

# Architecting Voltage Islands in Core-based System-on-a-Chip Designs

Jingcao Hu<sup>‡</sup>, Youngsoo Shin<sup>§</sup>, Nagu Dhanwada<sup>‡</sup>, and Radu Marculescu<sup>‡</sup>  
<sup>‡</sup>Department of ECE, Carnegie Mellon University, Pittsburgh, PA 15213  
<sup>§</sup>IBM T. J. Watson Research Center, Yorktown Heights, NY 10598  
<sup>†</sup>IBM EDA Laboratory, Hopewell Junction, NY 12533

## ABSTRACT

Voltage islands enable core-level power optimization for System-on-Chip (SoC) designs by utilizing a unique supply voltage for each core. Architecting voltage islands involves island partition creation, voltage level assignment and floorplanning. The task of island partition creation and level assignment have to be done simultaneously in a floorplanning context due to the physical constraints involved in the design process. This leads to a floorplanning problem formulation that is very different from the traditional floorplanning for ASIC-style design.

In this paper, we define the problem of architecting voltage islands in core-based designs and present a new algorithm for simultaneous voltage island partitioning, voltage level assignment and physical-level floorplanning. Application of the proposed algorithm to a few benchmark and industrial examples is demonstrated using a prototype tool. Results show power savings of 14%–28%, depending on the constraints imposed on the number of voltage islands and other physical-level parameters.

## Categories and Subject Descriptors

J.6 [Computer Applications]: Computer-Aided Engineering—*computer aided design (CAD)*

## General Terms

Algorithms, Design

## Keywords

System-on-a-Chip, voltage island, floorplanning, low-power, multiple  $V_{DD}$

## 1. INTRODUCTION

As the scale of process technologies steadily shrinks, more and more devices can be implemented on a single chip. This enables various applications to be realized as System-on-a-Chip (SoC) designs, especially by using pre-designed cores. Typical SoCs consist of programmable processors and peripheral cores that are connected to standard bus-based architectures. The SoC design process

starts with architectural design in conjunction with physical planning and performance/power analysis, along with estimation of die size and package selection [1]. This step is followed by mapping the design to a platform that is composed of a commonly used set of cores and their interconnections, or by changing legacy designs to comply with new design requirements. The final RTL description with timing assertions is submitted to the traditional chip-level design process consisting of logic synthesis, floorplanning and physical design, and physical synthesis-based timing closure. While meeting the timing requirements of modern SoC designs is difficult, power consumption has become another critical design metric due to increasing power density and wide use of portable systems. There are many techniques to reduce power consumption at each level of the design abstraction [2].

The core-base design using voltage islands [3] is a new technique which helps reducing both switching and standby components of power dissipation. Simply speaking, a voltage island is a group of on-chip cores powered by the same voltage source, independently from the chip-level voltage supply. The use of voltage islands permits operating different portions of the design at different voltage levels in order to optimize the overall chip power consumption. In the SoC context, the voltage island enables core-level power optimization by utilizing a power supply that is unique from the rest of the design.

Introducing voltage islands makes the chip design process even more complicated with respect to static timing, power routing, floorplanning, *etc.* In particular, the complexity grows significantly with the number of allowed islands. Thus, a designer using voltage islands needs to group together the cores powered by the same voltage source and ensure that the created grouping does not violate other design metrics such as timing and wiring congestion. The voltage islands need to be placed close to the power pins in order to minimize the power routing complexity and the IR drop. Since each island requires its own power grid and level converters to communicate with different other islands, the overhead with respect to area and delay is unavoidable. We may have additional area overhead due to potential dead spaces, if two or more cores are put into the same island but they cannot be packed perfectly. These additional requirements lend themselves into an interesting and unique floorplanning problem, which is the main objective of this paper.

Figure 1 shows an example of creating voltage islands in SoCs. Each core is associated with a list of voltages that can be used to operate it. For instance, the core  $c_2$  can operate at 1.0, 1.1 or 1.2V. All the cores have fixed shapes and, for this particular example, the cores  $c_1$ ,  $c_2$  and  $c_3$  are pre-placed to fixed positions. The chip-level voltage is assumed to be 1.2V meaning that we don't need to create a distinct voltage island for those cores that operate at 1.2V. If we want to minimize the power consumption, one obvious way is to

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ISLPED'04, August 9–11, 2004, Newport Beach, California, USA.  
Copyright 2004 ACM 1-58113-929-2/04/0008 ...\$5.00.

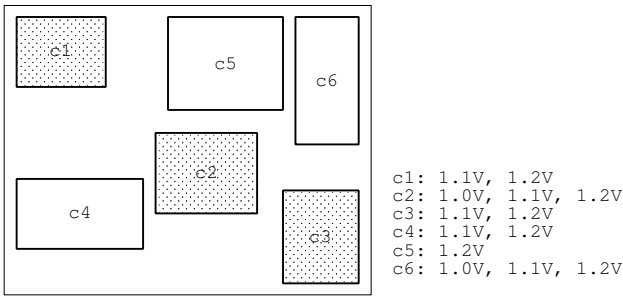


Figure 1: Example SoC for voltage island creation.

operate each core at its lowest voltage. This means that we need at least 3 voltage islands: one for  $c_2$  and  $c_6$ , one for  $c_1$  and  $c_4$ , and one for  $c_3$ . (Of course, other choices do exist. For instance, another choice could be: one island for  $c_2$  and  $c_6$ , one for  $c_1$ , and one for  $c_3$  and  $c_4$ .) Note that we cannot create a single voltage island with  $c_1$ ,  $c_3$ , and  $c_4$  because in such a case the bounding rectangle<sup>1</sup> would have to cover the entire chip image. Still, this is not a perfect solution: an island with  $c_2$  and  $c_6$  may not be allowed because the enclosing rectangle violates the constraint of proximity to the power pins in case they are located on the periphery of the image. A voltage island bringing  $c_1$  and  $c_4$  (or  $c_3$  and  $c_4$ ) would have a dead space inside of it. We may try to use more islands to alleviate some of these problems, but this is not easily possible because the number of islands that can be created is usually constrained from other design considerations. This example clearly shows a new *constrained floorplanning problem* with the objective very different from the traditional floorplanning in ASIC-style designs. As such, we address, for the first time to our knowledge, the floorplanning problem for voltage island creation with the objective of minimizing power consumption and area overhead.

The remainder of the paper is organized as follows. In the next section, we formulate the floorplanning problem for voltage island creation. In Section 3, we outline our algorithm and study some potential extensions. In Section 4, we present experimental results for several examples, and in Section 5 we conclude by summarizing our main contribution.

## 2. PROBLEM DESCRIPTION

We assume given a SoC consisting of a set of cores  $C$ . The chip image where the cores are to be floorplanned is also given implying that we do a fixed frame floorplanning as opposed to traditional min-area floorplanning. The choice depends on the overall chip design process [4], meaning that, in this paper, the floorplanning process is assumed to happen *after* the die size and package have been chosen.

For each core  $c_i \in C$ , the area is given as a product  $w_i h_i$ , where  $w_i$  and  $h_i$  are the width and the height of the core, respectively. The shape is fixed for the hard cores, yet rotation and mirroring are allowed. For soft cores, the acceptable aspect ratios are given as constraints. Each core is also associated with a *power table*, which specifies the legal voltage levels and the corresponding average power consumption values.

The legal voltage levels of a core can be characterized by core designers. For example, the designers may keep changing the supply voltage of a specific core, as long as the core-level timing assertions are still satisfied; this gives a list or range of voltages that

<sup>1</sup>We assume a rectangular outline for simplicity, though rectilinear outline may also be allowed.

can be supplied. Once the legal voltages are selected, the power consumption corresponding to each voltage can be characterized through simulation or estimation [5].

Let  $\pi_i$  denote any voltage island, which consists of a set of cores, i.e.  $\pi_i \subset C$ . Thus,  $C = \sum_i \pi_i \cup \sum_i c_i$ , where  $\sum_i c_i$  denotes those cores not assigned to any islands that are therefore operated by chip-level power supply. A voltage island has a unique voltage, denoted as  $v(\pi_i)$ , which is selected from the list of supply voltage levels, denoted as  $V(\pi_i)$ , which  $\pi_i$  can work at; thus  $v(\pi_i) \in V(\pi_i)$ .  $V(\pi_i)$  is equal to the intersection of the legal voltage levels of all  $\pi_i$ 's composing cores. As an example, if we create  $\pi_1$  with  $c_1$  and  $c_6$  in Figure 1,  $V(\pi_1) = \{1.1V, 1.2V\}$ . A voltage island  $\pi_i$  is said to be *compatible* with another voltage island  $\pi_j$  if and only if  $v(\pi_i) = v(\pi_j)$ , i.e. they have the same voltage level; it is *incompatible*, otherwise.

We assume an initial floorplan, which may be given upfront or may be the result of floorplanning with different design objectives such as performance. In the initial floorplan, some of cores may be pre-fixed or have assigned a certain area where they can be moved. This can be modeled by associating to each core a *move bound* defined by  $(lx_i, rx_i, by_i, uy_i)$ , where  $(lx_i, by_i)$  and  $(rx_i, uy_i)$  denote bottom-left and upper-right hand corner coordinates, respectively. The move bound may overlap with a core when it has a fixed location. Also, the move bound may become the entire chip image in case a core is not assigned any move bounds.

Our problem of *voltage island planning* consists of:

- partitioning  $C$  into a set of voltage islands and cores,
- area-planning for each voltage island,
- floorplanning of islands and cores.

Note that area-planning of each island involves another floorplanning step (namely, the voltage island-level floorplanning). Our objective is to simultaneously minimize power consumption and area overhead, while keeping the number of voltage islands less than or equal to a designer-specified threshold.

## 3. VOLTAGE ISLAND PLANNING ALGORITHM

### 3.1 Basic Data Structure

A *Voltage Island Compatibility Graph* (VICG)  $\mathcal{G} = G(\Pi, A)$  is a *complete undirected* graph which captures the current voltage island partitioning solution in an abstract way. Each vertex  $\pi_i \in \Pi$  represents a voltage island and each arc  $a_{i,j} \in A$  characterizes the “attraction” between the islands  $\pi_i$  and  $\pi_j$ . Each arc has a weight  $w(a_{i,j})$ , which is calculated using the following equation:

$$w(a_{i,j}) = \begin{cases} 1 + \alpha \times wires_{i,j}, & \text{if } v(\pi_i) = v(\pi_j) \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

where  $\alpha$  is a constant and  $wires_{i,j}$  denotes the number of wires between  $\pi_i$  and  $\pi_j$ . The weight  $w(a_{i,j})$  is used to guide the floorplanning by describing the potential savings when two islands are placed adjacently which, in turn, gives the possibility of further island merging. Intuitively, if the voltage islands  $\pi_i$  and  $\pi_j$  are *not* compatible, then they are independent since they can not be merged together even if they are placed adjacently. On the other hand, placing two compatible islands near each other increases the possibility of merging them into one single island, which eventually leads to a solution with a lower cost. Additionally, an island having more interconnections with its *compatible* islands should be given a higher priority in being placed nearby since more interconnections between separate voltage islands implies more level shifters, which leads to higher area costs. The constant  $\alpha$  is used to distribute the

### Algorithm

#### begin

```
L1: Given the initial island partitioning and floorplanning;
L2: Perturb the current solution;
L3: Update VICG;
L4: Floorplan for VICG edge weight minimization;
L5: Merge voltage islands;
L6: for each newly merged island do
L7:   Floorplan for area minimization;
L8: end do
L8: Cost calculation;
L9: if satisfactory solution found then
L10:   Output solution;
L11:   Return;
L12: else if meet exit criteria then
L13:   Output best solution so far;
L14:   Return;
L15: else Go to L2;
L16: end if
end
```

**Figure 2: Outline of floorplanning and island merging algorithm.**

effort in minimizing the level shifter area overhead versus the effort in minimizing the number of voltage islands. More specifically, given a larger value of  $\alpha$ , the algorithm will give a higher priority to minimizing the level shifter area overhead. On the other hand, given a smaller value of  $\alpha$ , minimizing the number of voltage islands become the primary goal in the optimization process.

The use of the VICG structure also offers enough flexibility in setting up different optimization objectives. For instance, we can easily add interconnect performance into our optimization by giving higher weights to those edges which lie on the critical path such that the length of the corresponding interconnects can be indirectly minimized.

## 3.2 Outline of Algorithm

Our approach is based on *simulated annealing* which guides the floorplanning and the island merging processes through the VICG graph. Figure 2 outlines the main steps of our approach.

As shown in Figure 2, given an initial voltage island partitioning and its associated floorplanning, the approach iteratively improves the solution quality by local perturbation, re-floorplanning and islands merging. Specifically, given the current solution, we first perturb it as described in subsection 3.4. This perturbation is then reflected back to the corresponding VICG. Next, a chip-level floorplanning is applied with the goal of finding a floorplan where *compatible* islands are likely to be placed adjacently. Following that, the island merging process consists of detecting the regions that contain potentially mergable islands (and eventually merging those islands), provided that the newly formed islands do not cause overlaps in the floorplan. In order to shrink the area of these islands, an island-level floorplanning is applied to each of the newly merged islands with the goal of minimizing its bounding box. Finally, this new solution is evaluated and its costs calculated. The above process is repeated until a satisfactory solution is found or, if this is not possible, a certain exit criteria (e.g. reaching a certain number of iterations) is met.

We use a unified cost function which is a weighted sum of different metrics including the number of islands, average power consumption, and area overhead. Note that given the flexibility of the simulated annealing algorithm, other cost functions (e.g. routing

congestion) can be also included, if necessary. This will be further discussed in subsection 3.6.

## 3.3 Integrated Floorplanning Process

As it appears from Figure 2, the speed of the newly proposed approach is determined mainly by the floorplanning process. More specifically, at each iteration, two successive floorplanning steps need to be applied:

- The *chip-level* floorplanning (L4 in Figure 2) tries to arrange the *compatible* islands in adjacent positions by minimizing the following cost function:

$$cost = \sum_{\forall i,j} w(a_{i,j}) \times d(\pi_i, \pi_j) \quad (2)$$

where  $d(\pi_i, \pi_j)$  represents the center-to-center Manhattan distance between islands  $\pi_i$  and  $\pi_j$  in the floorplan;  $w(a_{i,j})$  has been defined in (1).

- The *island-level* floorplanning (L7 in Figure 2) is applied to each newly merged island, as the composing cores inside the merged island may not be placed compactly enough. The floorplanning with the goal of area minimization helps not only in reducing the dead space inside these newly formed islands, but also in legalizing the newly generated floorplan by reducing the risk of islands overlapping.

Obviously, an efficient implementation of the floorplanner is critical for the performance of our approach. In this paper, we use a floorplanner based on sequence pair representation and evaluation [6]. The floorplanner uses simulated annealing on sequence pair data structure [7], and is capable of evaluating in  $O(n \log \log n)$  time, where  $n$  stands for the total number of blocks; typically it ranges from 5 to 100 for current practical designs. Move bound support is achieved by adding some dummy blocks. To exploit the property that a merged island contains just a few cores (typically under 5), the island-level floorplanner automatically switches between simulated annealing mode and the enumerating mode based on the number of composing cores in the target island; this helps reducing the run time.

## 3.4 Solution Perturbation

Perturbations to the current solution are performed at the beginning of each iteration. More precisely, one of the following three moves is probabilistically chosen:

- **Island split move (ISM):**

A voltage island, which consists of more than one cores, is randomly picked and split up into a set of islands.

- **Island voltage change move (IVCM):**

In this move, a voltage island which supports two or more legal supply voltages is randomly chosen and its supply voltage level randomly switched to one of its legal voltages.

- **Multi-island voltage change move (MIVCM):**

In this move, a voltage supply level  $l_i$  is randomly picked up and all the islands which supports  $l_i$  will be assigned to voltage  $l_i$ .

All three perturbations lead to changes in the VICG graph that corresponds to the current solution. More specifically, the IVCM and MIVCM moves change the relevant arc weights, while the ISM move changes not only the VICG's arc weights, but also its topology. Obviously, compared to IVCM, ISM and MIVCM have a more significant perturbation on the current solution and are thus more suitable for being applied when the annealing temperature is high.

### Algorithm begin

```

L1: Sort voltage levels by the number of related islands;
L2: for each supply voltage level  $l_i$  do
L3: Find all islands with voltage  $l_i$ ;
L4: Create rectangular region  $R$ ;
L5: Add  $R$  to split list  $L$ ;
L6: while  $L \neq \{\}$  do
L7: Get next region  $R$  from  $L$ ;
L8: if  $R$  contains no more than one island then
L9: Continue;
L10: if  $R$  overlaps with an incompatible island  $\pi_a$  then
L11: Cut  $R$  using  $\pi_a$  into smaller regions;
L12: Combine new generated regions;
L13: Add new regions to  $L$ ;
L14: else
L15: Merge islands in  $R$  to  $\pi_n$ ;
L16: Floorplan  $\pi_n$  for area minimization;
L17: end if
L18: end do
L19: end do
end

```

Figure 3: Outline of island merging process.

## 3.5 Island Merging

Following perturbation and chip-level floorplanning, the islands that are compatible with each other are likely to be placed in adjacent positions. The heuristic in Figure 3 is then applied to detect and merge the islands that can be merged.

As shown in Figure 3, the supply voltage levels used in the system are first sorted based on how many voltage islands are assigned to a given voltage level. More precisely, the voltage levels that are used by more voltage islands are given a higher priority. Intuitively, the more islands use the same voltage level, the higher the probability of finding mergable islands using that voltage is.

Next, we perform the following operation on each voltage level in this list. For each voltage level  $l_i$ , all the voltage islands that use  $l_i$  are selected and a rectangular region  $R$  is created by drawing the bounding box of these islands (step L4 in Figure 3). Obviously, to reduce the number of voltage islands in the system, we would like to merge all these islands at  $l_i$  into one single merged island with shape  $R$ . Unfortunately,  $R$  may overlap with other islands, thus rendering the floorplan infeasible. Additionally, creating a merged island with shape  $R$  may significantly increase the area overhead due to the dead space. The loop L6 to L18 in Figure 3 resolves the aforementioned problems by recursively cutting the region  $R$  into smaller regions together with island-level floorplanning.

To better describe our merging algorithm, Figure 4 provides a step by step illustration of the island merging process. Figure 4(a) gives an example of a piece of the initial floorplan for a chip image. As shown in step L4 of Figure 3, the initial rectangular region  $R$  is the external bounding box in Figure 4(a). Because it overlaps with an incompatible island ( $\pi_0$ ) which uses a different voltage level, creating a merged island with the size of this bounding box is not feasible. (Additionally, such a solution will also contain a significant amount of dead space.) Consequently, the steps L6 to L18 in Figure 3 are necessary to solve this issue. More specifically, the original region is divided into 8 new regions by cutting it around  $\pi_0$  (as shown in Figure 4(b))<sup>2</sup>. In the next steps, these newly gen-

<sup>2</sup>Depending on how an incompatible island overlaps with the region, the number of new regions may vary.

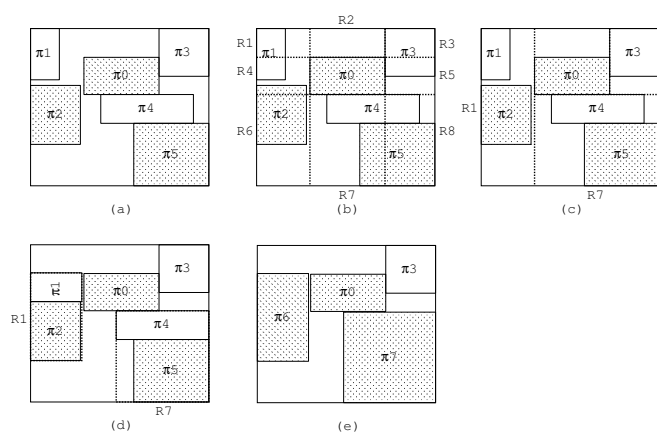


Figure 4: An example for island merging.  $\pi_1$ – $\pi_5$  are compatible islands.  $\pi_0$ ,  $\pi_2$ , and  $\pi_5$  are fixed.

erated regions are merged to form larger regions. Regions containing more islands are given a higher priority for combination, such that more islands will likely be merged in the following steps. For instance, region  $R_7$  will be combined with region  $R_8$  instead of region  $R_6$  since combining regions  $R_7$  and  $R_8$  generates a new region which contains two islands. On the other hand, if we combine regions  $R_6$  and  $R_7$  instead, the newly formed region would contain no island. Figure 4(c) shows the new regions ( $R_1$  and  $R_7$ ) left after the combining step.

The above procedure is repeated until the regions under consideration do not overlap with any incompatible island. For each of these remaining regions, a new island will be built by merging the islands inside the corresponding regions. Meanwhile, the island-level floorplanner is applied to each of these newly merged islands to minimize their outlines. For instance, Figure 4(d) shows the picture of this step, while Figure 4(e) gives the partitioning and the floorplan at the end of this iteration. Since islands  $\pi_6$  and  $\pi_7$  contain cores with fixed location, both of them will be marked as fixed islands.

It is worth mentioning that the existence of fixed cores (or cores with move bound) adds extra complexity to island-level floorplanning. In such cases, a minimal outline floorplan has to be found under the constraints of satisfying fixed core and/or specified move bounds. On the other hand, most of the previous work in floorplanning either assumes no move bound constraints when dealing with area minimization, or targets finding a floorplan under fixed outlines. Thus, previous results in floorplanning can *not* be directly applied to solve our problem.

We used the heuristic in Figure 5 for island-level floorplanning; this converts the problem of area minimization into the one of finding a feasible floorplan under a given outline. As shown in Figure 5, the procedure uses the bounding box obtained after the cutting (for instance, the rectangle  $R_1$  in Figure 4(c)) as the starting outline for floorplanning. If a feasible floorplan can be found, the procedure tries to decrease the size of the outline by incrementally shrinking it from the chosen boundary (north, east, south or west). The starting boundary for the shrinking process depends on the current slack in the floorplan, while taking into consideration the relative position of the move bounds or the fixed cores with regard to the current outline boundary. This process is repeated until the outline can not be shrunk any further; this minimum then corresponds to the bounding box with the minimal area. We note that although multiple floorplanning runs are needed in order to floorplan just one island, the

execution time is actually quite acceptable since the voltage island usually contains a small number of cores.

```

Algorithm
begin
L1:  outline = original bounding box;
L2:  while feasible floorplan exist do
L3:    floorplan = find floorplan with outline;
L4:     $b$  = decide boundary to shrink;
L5:    shrink outline from boundary  $b$ ;
L6:  end do
L7:  return floorplan;
end

```

Figure 5: Outline of island-level floorplanning.

### 3.6 Possible Extensions

The cost function used in this paper includes the area overhead, the number of voltage islands and the power consumption of the target system. However, our approach can be extended to incorporate other factors of interest in the design process. For instance, in designs where the routing congestion becomes a serious issue, the total wire length usually needs to be minimized during the floorplanning. The approach presented in this paper can easily be adapted to consider this scenario by modifying the arc weights equation of the VICG as follows:

$$w(a_{i,j}) = \begin{cases} 1 + \alpha \times wires_{i,j} + \beta \times wires_{i,j}, & \text{if } v(\pi_i) = v(\pi_j) \\ \beta \times wires_{i,j}, & \text{otherwise} \end{cases} \quad (3)$$

where  $\beta$  is a constant which is specified by the designer to control how much effort should be devoted to the wire length minimization.

Another important extension would be to provide support for bounded delay for the critical nets at this architecting stage. This is especially important as the interconnect delay has become a serious issue in deep sub-micron designs. To this end, we are currently working on providing this type of performance constraint using techniques similar to that proposed in [8]. By adding dummy blocks to force the move bound of the related cores, the floorplanner guarantees that these critical nets will *not* exceed their specified length, which is directly related to the wire delay. However, caution must be taken to guarantee the convergence of the optimization process where many constraints and optimization objectives need to be considered simultaneously.

Another way to address the interconnect timing issue in voltage island architecting is through the direct interaction with the designer. In this case, the designer would adaptively add additional constraints before applying the voltage island architecting tool. For instance, if it is found that in the current solution the timing constraints are violated for the links between two cores  $c_i$  and  $c_j$ , the designer can go back to the input specification, add a move bound  $mb_{i,j}$  for both  $c_i$  and  $c_j$ , and then re-apply the tool to this modified input, such that the upper bound of the distance between  $c_i$  and  $c_j$  can be guaranteed. This will automatically ensure that the timing constraints are satisfied. Additionally, if the signal between  $c_i$  and  $c_j$  is not registered at the core boundary, the designer can also restrict the allowed voltage levels used by  $c_i$  or  $c_j$ , such that the timing constraints of the interconnects between  $c_i$  and  $c_j$  can be relaxed. This process can be iterated until the results generated by the tool satisfy all the timing constraints, in addition to those imposed on power and area.

Table 1: Floorplanning results

	# cores	area util.	# vi	run time(s)	power savings	area overhead
i1	23 (10)	73.1%	3	154	16.9%	8.3%
i2	10 (3)	50.0%	2	197	14.0%	3.2%
n10	10 (2)	50.2%	2	327	27.6%	1.5%
n30	30 (3)	74.4%	4	291	19.6%	0.3%
n50	50 (3)	63.4%	5	607	18.7%	4.3%

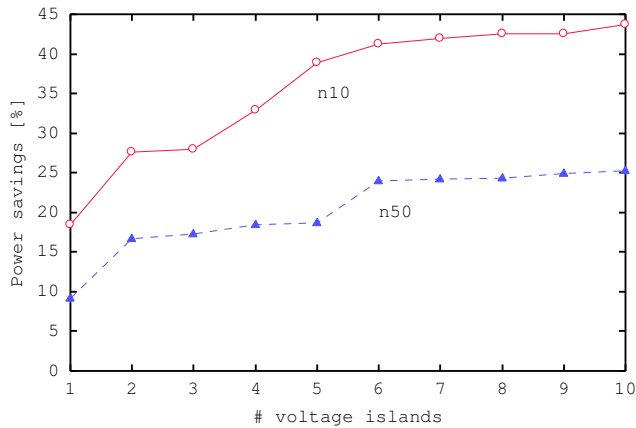
## 4. EXPERIMENTAL RESULTS

We implemented a prototype tool written in C++ and based on the algorithms presented in Section 3. Table 1 summarizes the examples we constructed for the experiment, which include two industrial (i1, i2) and three synthetic benchmark (n10, n30, n50) examples. The second column shows the number of cores. The initial floorplan is constructed by fixing the position of some cores and then running the automatic floorplanner. The number of cores that are pre-placed is shown in parenthesis in the second column. We assume that all cores are operated by a single supply voltage in the initial floorplan. We also assume a list of voltage levels available for each core in Table 1. In order to give the highest priority to minimizing the number of voltage islands, we assigned  $\alpha$  to be 0 for all these experiments.

For the industrial examples (i1 and i2), the power consumption has been obtained from a spreadsheet. In the case of the other three benchmark examples, we generated the power consumption values assuming that they are proportional to the area of each block and then scaled them appropriately with each supply voltage. The third column shows the area utilization for a given chip image. The fourth column shows the final number of islands that are created, which is equal to the constraint imposed on the maximum number of voltage islands for each example. The last two columns show the power savings and area overhead with respect to the initial floorplan where all cores operate at the chip level voltage. The area overhead includes the dead space inside the voltage islands and additional area due to the power rings. Note that we assume a fixed frame floorplanning: the unused space at chip-level is not considered to be a dead space because that area is a resource for chip-level logic such as test logic and buffers, and for those cores whose designs are still in flux.

The results in Table 1 show that our algorithm is very fast. It typically finishes in minutes with the actual run time depends on the size of the design and the constraints imposed on it. The results also suggest that our algorithm gives high-quality solutions with respect to power savings, area overhead, and the number of islands used in the design. Since the cost function is a weighted sum of different metrics, different solutions can be obtained by controlling the weights depending upon designer's assessment of the importance of each metric. For example, one can increase the weight for the power savings at the cost of area overhead if the target application is supposed to work under a tight energy budget.

Power savings that can be obtained are dependent on several factors: the constraints on the number of voltage islands, the pre-placed blocks that may interfere with creating voltage islands, the proximity constraints to power pins, the available voltage levels of each core. Thus, it is difficult to assess the quality of solutions for a given problem instance. However, by relaxing the constraints on the number of voltage islands and on the proximity to the power pins, we can plot the power savings as a function of the number of



**Figure 6: Power savings with the number of voltage islands for benchmark examples n10 and n50.**

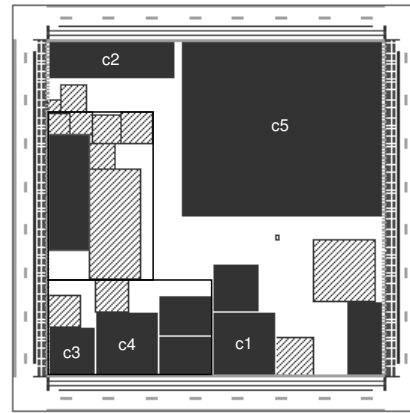
voltage islands; this can give a rough idea of how good the solution is. Figure 6 confirms that the power savings increase if more voltage islands are created for a particular benchmark n10. Note that 10 voltage islands correspond to the case when the maximum power savings are achieved; that is, it is an upper bound. The upper bound would be reached much earlier if, for instance, we increase the weight for the power savings at the cost of increased area overhead. The same process is repeated in Figure 6 for n50, which indeed shows the same trend.

Figure 7 shows the floorplanning result for the example i1, which is a platform-based design consisting of IBM PowerPC 405 and peripheral cores connected to the CoreConnect bus architecture [1], [9]. The solid boxes (shown in black) indicate the pre-placed cores in the initial floorplan; this topological arrangement is determined by the I/O requirements and the dataflow between a core and its associated buffers (SRAM cores). We assume that all cores are hard cores<sup>3</sup> and operate at a single 1.3V supply in the initial design. For voltage islands planning, we assign legal voltages within the range 1.0V to 1.3V for each core. For example, c1 can be operated at {1.1V, 1.2V, 1.3V}, c2 at {1.0V, 1.1V, 1.2V, 1.3V}, and c5 at {1.3V}. Three voltage islands are created as a result of running our tool: two shown with enclosing rectangles both with 1.1V supply, and the third one consisting of a single core (c1) powered by 1.0V. The experimental results are summarized in Table 1. Note that c2 is still at 1.3V although its minimum legal voltage is 1.0V. It could be operated at 1.1V instead of 1.3V if it is included in the voltage island on the left-hand side of the image, but that would lead to a significant dead space in the voltage island since c2 has a fixed location. c4 is powered by 1.1V, which is the supply of the enclosing voltage island, although its minimum legal voltage is 1.0V, while c3 is at its minimum supply.

## 5. CONCLUSION

In this paper, we addressed the problem of voltage islands planning for core-based SoC designs. This novel problem involves partitioning cores into several islands and floorplanning both at chip and island-level. By using a graph-based representation, we have shown that we can model the partitioning and floorplanning steps in an integrated fashion. We proposed a simulated annealing-based algorithm which gives high quality solutions with respect to power

<sup>3</sup>Note that, as indicated in the previous sections, the algorithm and the tool we implemented can handle soft cores as well.



**Figure 7: Floorplanning result for i1.**

savings, area overhead, and the number of voltage islands that are used in the design. The proposed cost function is flexible and can be extended to include other design metrics such as routing congestion and performance.

## References

- [1] R. A. Bergamaschi, Y. Shin, N. Dhanwada, S. Bhattacharya, W. E. Dougherty, I. Nair, J. Darringer, and S. Paliwal, "SEAS: A system for early analysis of SoCs," in *Proc. Int'l Conf. on Hardware/Software Codesign and System Synthesis*, Oct. 2003, pp. 150–155.
- [2] M. Pedram and J. Rabaey, *Power Aware Design Methodologies*, Kluwer Academic Publishers, 2002.
- [3] D. E. Lackey, P. S. Zuchowski, T. R. Bednar, D. W. Stout, S. W. Gould, and J. M. Cohn, "Managing power and performance for System-on-Chip designs using voltage islands," in *Proc. Int'l Conf. on Computer Aided Design*, Nov. 2002, pp. 195–202.
- [4] A. B. Kahng, "Classical floorplanning harmful?," in *Proc. Int'l Symp. on Physical Design*, Apr. 2000, pp. 207–213.
- [5] F. N. Najm, "A survey of power estimation techniques in VLSI circuits," *IEEE Trans. on VLSI Systems*, vol. 2, no. 4, pp. 446–455, Dec. 1994.
- [6] X. Tang and D. F. Wong, "FAST-SP: a fast algorithm for block placement based on sequence pair," in *Proc. Asia South Pacific Design Automat. Conf.*, Jan. 2001, pp. 512–526.
- [7] H. Murata, K. Fujiiyoshi, S. Nakatake, and Y. Kajitani, "VLSI module placement based on rectangle-packing by the sequence pair," *IEEE Trans. on Computer-Aided Design*, vol. 15, no. 12, pp. 1518–1524, Dec. 1996.
- [8] X. Tang and D. F. Wong, "Floorplanning with alignment and performance constraints," in *Proc. Design Automat. Conf.*, June 2002, pp. 848–853.
- [9] IBM Corp., "IBM platform-based design kit," [http://www-3.ibm.com/chips/products/asics/methodology/design\\_kit.html](http://www-3.ibm.com/chips/products/asics/methodology/design_kit.html).