# Voltage Island-driven Floorplanning

**MA, Qiang**

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science and Engineering

# Abstract

As the integration density continues to advance, energy efficiency has become one of the most important issues to be addressed in today's System-on-a-Chip (SoC) designs. One way to lower the power consumption is to reduce the supply voltage. *Multiple Supply Voltage* (MSV) is thus introduced to provide higher flexibility in controlling the power and performance trade-off. In region-based MSV, circuits are partitioned into "voltage islands" where each island occupies a contiguous physical space and operates at one supply voltage. These tasks of voltage island partitioning and voltage level assignments should be done simultaneously in the floorplanning process in order to take those important physical information into consideration.

In this thesis, we formulate the MSV driven floorplanning problem and solve it. Particularly, we address this problem in two stages. In the first stage, we solve a simpler version of this problem by assuming that timing requirements of the given circuit can be achieved by all different voltage levels of the modules; in the second stage, we try to consider this problem in more details, i.e., the working voltage levels of the modules are properly assigned to meet the timing requirements.

In the first stage, we focus on optimal voltage island partitioning and voltage assignment, where the voltage islands generated are rectangular in shape. An elegant voltage island-driven floorplanner is provided to generate a floorplan with rectangular voltage islands. Given a candidate floorplan solution represented by a normalized Polish expression, optimal voltage level assignment and island partitioning can be obtained simultaneously to minimize the total power consumption. Simulated annealing is used as the basic searching engine. By using this approach, significant power savings (up to 50%) can be achieved for all data sets. This floorplanner can also be extended to minimize the number of level shifters between different voltage islands and to simplify the power routing step by placing the islands in proximity to the corresponding power pins.

In the second stage, we focus on solving the sub-problem of voltage assignment on a given netlist under timing constraints. We first show that the voltage assignment task on a given netlist can be formulated as a convex cost dual network flow problem and can be solved optimally in polynomial time using a cost-scaling algorithm when the delay choices of each module are continuous in the real or integer domain. We can make use of this approach to obtain a feasible voltage assignment solution in the general cases with power consumption approximating the minimum one. Furthermore, we propose a framework to optimize power consumption and physical layout of a circuit simultaneously during the floorplanning stage, by embedding this cost-scaling solver into a simulated annealing based floorplanner. We compared our approach with the latest work [11] on the same problem, and the experimental results show that, using our framework, significant improvement on power saving (18% less power cost on average) can be achieved in much less running time (7X faster on average) for all the test cases, which confirms the effectiveness of this approach.

# 摘要

由於電路集成密度不斷增加，功耗逐漸成爲了當今片上系統(SoC)設計中最重要的問題之一。 降低供電電壓是降低功耗的一種有效措施。因此，多供應電壓(MSV)設計方法爲平衡電路的功耗和性能提供了更大的靈活性。在區域性多供應電壓設計中，電路被分爲若干個"電壓島"，其中每一個電壓島佔據一塊連續的區域，而且同一個電壓島裏的電路單元工作於同一個電壓。在 MSV 設計中，同時考慮電壓島的劃分，供應電壓的分配和電路佈局將會很有利於整個設計的優化。

在本論文中，我們提出並解決了多供電電壓驅動的電路佈局問題。 具體來說，我們將這個問題分爲兩個階段進行解決。在第一個階段， 我們認爲所有的電壓選擇都可以滿足電路的時序要求，從而解決了這個問題的一個比較簡單的版本；在第二個階段，我們通過在分配各個模組的供電電壓時考慮滿足電路時序要求，更深入的考慮了這個問題。

在第一個階段，我們提供了一個電壓島驅動的佈局工具，它可以產生一個具有矩形電壓島的電路佈局。 給出一個由標準化波蘭運算式表示的電路佈局，我們可以得到這個佈局上最優的電壓分配和電壓島的劃分，這裏最優指使得整個電路功耗最低。 搜索引擎採用了類比退火方法。 這種方法可以很有效地降低功耗(最大可以達到 50%). 另外,我們可以拓展這個佈局工具，使其同時考慮減少需要使用的電壓轉換器的數量，以及考慮電壓島和與其對應的供電引腳的毗鄰性。

在第二個階段， 我們主要考慮了解決在滿足時序要求下的電壓分配問題。在 MSV 設計中，電路單元的電壓分配一定要滿足電路時序要求。 我們論證了在一個給定的電路網表上滿足時序要求的電壓分配問題可以歸結爲最小開銷網路流問題. 當每一個電路單元的延遲的選擇範圍是一些連續整數時，cost-scaling 演算法可以用多項式時間得到該電壓分配問題的最優結果。 在一般情況下，即電路單元延遲的取值範圍不連續時，我們可以利用這種方法得到一個近似結果。 而且，通過把 cost-scaling solver 嵌入一個基於類比退火方法的電路佈局工具，我們給出了一個用來同時優化功耗和佈局的框架。 我們跟最近的針對相同問題的工作[11]做了比較，實驗結果顯示，我們的方法可以更在更短的時間內得到更好的結果，從而證明了我們的方法的優越性。

# Acknowledgement

This thesis owes much to the thoughtful and helpful comments of my supervisor, Professor Evangeline Fung Yu Young, who patiently motivated me to conceive and develop the main idea of the thesis. Here I would like to express to her my greatest gratitude for her supervision, advice and instructions from the very early stage of this research as well as giving me extraordinary experiences through out the work. Above all and the most needed, she provided me unflinching encouragement and support in various ways. Without her inspirational guidance, this thesis could not be successfully completed.

Also, I gratefully acknowledge Professor David Yu-Liang Wu and Professor Qiang Xu, the committee members during the oral defense. Thanks for having read a draft of this thesis and having made their precious comments and suggestions. As for any remaining errors, the responsibility for the text rests entirely upon the author. Special thanks go to Mr. Lei Shi for his kind help on the formatting of this thesis.

Finally, I would like to thank everybody who was important to the successful realization of thesis, as well as expressing my apology that I could not mention personally one by one.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Background

The technology of *integrated circuit (IC)* was widely adopted for computing devices like microprocessors, memory modules, and many other interface chips since 1960*s*. It is not surprising to find that we are surrounded by a huge number of computing devices in daily life, such as our personal computers, the ATM machines we use to withdraw cash, and many other electronic appliances. IC is one of the core components of those computing facilities.

As the *Very Deep Sub-Micron(VDSM)* technology advances, IC has evolved from *Small Scale Integration(SSI)* to *Very Large Scale Integration(VLSI)*, which means the integration density becomes much higher while the interconnections tend to be much more complicated, thus introducing many new challenges to VLSI design automation.

Producing a chip is quite a complicated process containing a couple of steps, each of which can be accomplished with the help of the specifically implemented CAD tools. Generally speaking, the process starts with a formal specification, and the final product is a fabricated chip. The VLSI design cycle traditionally consists of steps including Architecture Design, Functional Design, Logic Design, Circuit Design, Physical Design and finally, Fabrication. Among all these steps, Physical Design is of significant importance, whose task is to transform a circuit representation into a geometric representation. The geometric representation of a circuit is called a layout. During physical design, problems like where to place the modules, how the interconnections between the modules should be made, etc, will be addressed. As physical design is a crucial yet complex step in the design cycle, it can be further broken down into sub-steps, such as partitioning, floorplanning, placement, routing, and so on.

Floorplanning is an important step in the physical design cycle, as a good planning in the early design phase is crucial to avoid unnecessary iteration in the design cycle. During the step of floorplanning, the blocks are positioned on the chip roughly, so as to optimize the circuit size and performance according to the circuit specification. A compact design is favorable, but there are many other important aspects that have to be taken care of. For example, issues like the block dimensions, overall delay and power consumption should be taken into account. In addition, there may also be some special constraints needed to be satisfied, e.g., some blocks are pre-placed, while some others are required to be placed adjacent to each other, etc.

## 1.2 Floorplanning

The input to the floorplanning phase is a set of blocks, the area of each block, the possible shapes of each block, the number of terminals of each block, and the interconnections between blocks. In the floorplanning phase, we are going to plan the position and the shape of each block, together with the pin positions. The shapes for some blocks are fixed and cannot be altered. We called those blocks hard blocks. For other blocks, the shapes can be altered as long as their aspect ratios are within the desired value range. Those blocks are called soft blocks. A formal definition of the floorplanning problem is given as follows:

**Definition 1 Floorplanning** - *Given a set of n modules $\{M_1, M_2, \ldots, M_n\}$, where each module $M_i$ is associated with an area $A_i$, together with two aspect ratio bounds $r_i$ and $s_i$ such that $r_i \leq h_i/w_i \leq s_i$, where $h_i$ and $w_i$ is the height and the width of module i respectively. The output of the problem is a packing of the set of modules, i.e., the x and y coordinates and the dimension $(h_i, w_i)$ of each module. There should be no overlapping between modules, and the circuit performance should be optimized.*

There are several objectives to be optimized in floorplanning, like the total chip area, the total wire length, the critical path delay, etc.

1. *Chip Area* - Area minimization is one of the most commonly adopted objectives. Minimizing the chip area implies minimizing the wire length, and hence reducing the circuit delay.

2. *Total Wire Length* - In addition to minimizing the chip area, minimizing the total wire length directly is also another important goal. Beside the timing issues, using less wires to connect the modules means consuming less resources, and thus reducing the production cost.

3. *Delay* - In some cases, minimizing the total wire length is not enough. Timing is and important issue. The final circuit performance can be optimized by minimizing the delay on the critical path.

4. *Routability* - Routability refer to the possibility of completing all the connections. A non-routable floorplan is of no use even if it is area-optimized and delay-optimized. Enhancing the routability of a floorplan means to reduce

the chance of encountering routing problems in the downstream designing steps.

5. *Others* - There are still some other objectives in floorplanning, like minimizing heat dissipation, minimizing power consumption, etc.

## 1.3 Motivations

As technology scales for increased circuit density and performance, the need to reduce power consumption increases in significance as designers strive to utilize the advancing silicon capabilities. The total power consumed by conventional CMOS circuitry is composed of two sources, namely, dynamic power and leakage power. The leakage power refers to the sub-threshold current of the transistors in the circuit [1], while the dynamic power represents the power consumed by the intended work of the circuit to switch states and thus execute logic functions. Dynamic power is primarily composed of the power associated with the charging or discharging of the capacitance of the switching nodes [10]. Generally, the dynamic power consumption can be computed by

$$P_{dynamic} = k \cdot C_{load} \cdot V_{dd}^2 \cdot f$$

where $k$ is the circuit switching rate, $C_{load}$ is the loading capacitance, $V_{dd}$ is the supply voltage, and $f$ is the clock frequency. Dynamic power dominates the total power consumption in today's logic design. The dynamic power consumption is proportional to the square of $V_{dd}$, implying that lowering supply voltages can effectively reduce dynamic-power consumption, although the performance also degrades. Therefore, *Multiple supply voltage* (MSV) is introduced to provide higher flexibility in controlling power and performance trade-off.

MSV designs involve the partitioning of a chip into areas called "voltage islands" that can be operated at different voltage levels, or be turned off when idle.

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 significantly affect each other. In addition, there are other issues to be considered. For example, the voltage islands should be placed close to the power pins in order to minimize the power routing complexity and the 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 designs using voltage islands.

## 1.4 Design Implementation of Voltage Islands

An SoC architecture based on voltage islands requires additional design components to ensure reliable communications across island boundaries, distribute and manage power, and save and restore logic state during power-off and on.

A voltage island represents a level of hierarchy with unique powering, and it exists within a parent block which constitutes the physical region in which the island is placed. An island's parent block may be the top level of a chip design or even another island at the next higher level of the chip hierarchy. The circuits within a voltage island are primarily powered from the island voltage, called VDDI (VDD-island), while the circuits in the parent terrain are powered from a supply voltage called VDDO (VDD-outside). With deeper hierarchy, the VDDO of one island may be equivalent to the VDDI of a parent island in which it is contained.

When such a gate operates at a voltage sufficiently lower than the gate it drives, signal margins and performance will degrade, and the driven circuit will consume significantly higher power. Further increases in the voltage difference will eventually result in unreliable signal switching. Clearly, additional circuitry is necessary to handle the differences in both magnitude and timing that can occur between VDDI and VDDO at island boundaries. Circuits called Voltage Island Receivers perform this function for signals going from the parent block into the island, while Voltage Island Driver cells perform similar task for signals going from the island to the parent block. These drivers and receivers must provide reliable voltage level shifting between VDDI and VDDO for a wide range of operating voltages, and do so with minimal impact to signal delay or duty cycle.

Leakage or standby power can be reduced by lowering the voltage of functionally-inactive islands well below the level required for reliable operation. However, some logic states, prior to power-down, may need to be preserved to resume operation once the island is powered up again, at the end of the inactive period. Special state-saving latches provide a solution to this problem, eliminating the need to transfer logic states off-island and back in order to save and restore necessary logic states.

A voltage island requires at least one power source and complete isolation from all other supplies on the chip, in order to enable independent power sequencing. Physical planning of voltage islands includes determining the number of power sources that meets all islands' power requirements. The following are possible types of off-chip and on-chip power source:

- Solder bumps or C4 pads
- Wire bond pads
- Island power switches
- Voltage regulators

Some voltage island circuits require both the internal voltage island supply (VDDI) and the external voltage supply (VDDO) for proper operation. These include level-shifting circuits, state-saving latches, and power switches. Power-routing tools must be aware of these constraints and make efficient VDDI and

VDDO connections without creating electrical or wireability issues.

The P/G network of the chips using voltage islands can be much more complex than traditional one because many additional decisions must be made to fit the island design. These decisions may include the following: to use on-chip power regulator or off-chip power source, to use separate Vdd networks or use DC-DC converter in every local voltage island, and the topology of the power grid of each island (wire width, pitch, spacing, etc.). Many possibilities exist for powering voltage islands, driving further requirements for special design components. VDDI or VDDO may be supplied directly from a unique, non-switching power source. One or both may be the output of an on-chip voltage regulator, whose voltage value can be fixed or programmable. Finally, VDDI or VDDO may be a switched version of some other voltage supply, controlled by one or more PFET or NFET switches. A given SoC design may use one or more of these approaches depending upon the product design objectives. An example of power network design for a chip with voltage islands is shown below:

• Cells with different supply voltages share the same ground network, but use different power supply networks.

• There is a global power network, to which cells not in any voltage island are connected. For each voltage island, there is a separate Vdd network, connected to special power pads.

• The global power network is part of a uniform mesh, with the same width and pitch across the chip, except the parts over voltage islands are removed.

• There is a boundary power ring along the outer edges of the global power network. The width of the power ring is several times wider than that of the top-level wires in the global power network. The power ring connects the global network to power pads, and is very helpful to ensure power delivery integrity.

## 1.5 Problem Formulation

In the MSV driven floorplanning problem, we are given a set of $n$ modules with areas $A_1, A_2 \ldots A_n$ and aspect ratio bounds $[l_i, u_i]$ for $i = 1 \ldots n$. Each module $i$ is associated with a power table $T_i$ that specifies the legal voltage levels for the module and the corresponding average power consumption values. In MSV designs, the timing requirements of the circuits should be satisfied. Besides, the power network resource should also be minimized. Given a floorplan, the power network resource $\Phi_q$ for the modules working at voltage $v_q$ can be estimated by the half-perimeter of the bounding box of all those modules [11], while the total power network resource is the sum of $\Phi_q$, for $q = 1, 2, ..., k$ where $k$ is the number of different voltage levels available.

We can define the MSV driven floorplanning problem as follows:

**Definition 2 MSV Driven Floorplanning** - *Given a netlist, a timing constraint and a set of modules, each of which has multiple choices of supply voltage, generate a floorplan in which each module is assigned to work at a specific voltage level such that the timing constraint is satisfied and a weighted sum of the power consumption, power network resource, area and interconnect length is minimized.*

In this thesis, we address this problem in two stages. In the first stage, we solve a simpler version of this problem by assuming that timing requirements of the given circuit can be achieved by all different voltage levels of the modules; in the second stage, we try to consider this problem in more details, i.e., the working voltage levels of the modules are properly assigned to meet the timing requirements.

In the first stage, we focus on optimal voltage island partitioning and voltage assignment, where the voltage islands generated are rectangular in shape. In this work, the timing requirement is not explicitly taken into consideration, and we assume that the given voltage levels of each module can meet timing. 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 a weighted sum of the total power consumption, area and interconnect length is minimized. Each island will be supplied with the lowest possible voltage level common to all the modules in that island while the remaining modules not assigned to any island will be operated at the chip-level voltage.

In the second stage, we consider timing more explicitly. Each module $m_i$ is given $k_i$ choices of supply voltages $v_q^i$, for $1 \leq q \leq k_i$, and the power-delay tradeoff in $m_i$ is represented by a *delay-power curve* (DP Curve), $\{ (d_i^1, p_i^1), (d_i^2, p_i^2), ..., (d_i^{k_i}, p_i^{k_i}) \}$, where each pair $(d_i^q, p_i^q)$ is the corresponding delay and power consumption when $m_i$ is operated at $v_q^i$. Note that each $d_i^q$ is in $Z^+$ (positive integer set), since the delay of a module is measured in terms of the number of clock cycles. Fig. 1.1(a) is an example of a DP Curve, which contains three choices of supply voltages for this module. Moreover, we assume that for each module, power is a convex function of delay when each point $(d_i^q, p_i^q)$ is connected to its neighboring point(s) by a linear segment in the DP Curve. We are also given a timing requirement $T_{cycle}$, called clock cycle time, of this circuit and require that the critical path delay is at most $T_{cycle}$. We first focus on solving the sub-problem of voltage assignment on a given netlist under timing constraints, objective of which is to select a delay-power pair for each module in its DP Curve such that the sum of power consumption is minimized while the timing constraint is satisfied. Globally, the voltage assignment procedure is incorporated into a simulated annealing based floorplanner to solve the MSV driven floorplanning problem.

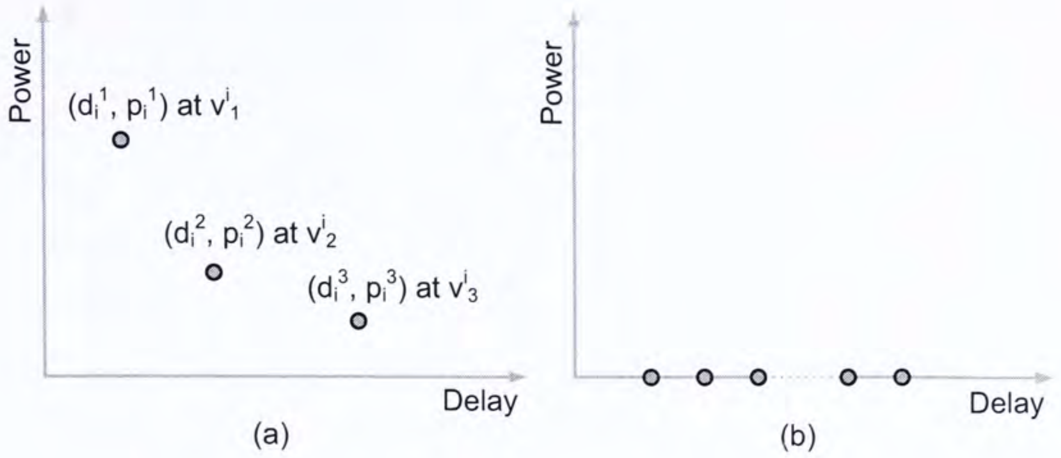


Figure 1.1: DP curve of $(a)$ a vertex and $(b)$ an edge in $G$

## 1.6 Progress on the Problem

A number of previous works addressed the voltage assignment and island generation problem in floorplanning and placement. In these previous works, MSV is considered at various design stages, including the floorplanning and placement stages [8], [9], [11], [13], [14], and the post-floorplanning and post-placement stages [5], [12], [16], [19], [20].

For the latter category, Wu *et al.* [19] minimized the number of voltage islands generated on a slicing floorplan using a dynamic programming based approach, Ching *et al* [5] then considered voltage islands that are non-rectangular in shape, and they also targeted at minimizing the number of voltage island generated by iteratively partitioning a tree structure, which is initially constructed according to an adjacency graph of a candidate floorplan to reflect the similarities of required working voltage levels among adjacent blocks. Mak *et al.* [16] formulated the voltage assignment and island generation problem as an *integer linear program* (ILP). In this 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 an ILP 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. [20] and [12] take timing constraints into consideration. In [20], Wu *et al.* proposed an approximation algorithm based on a *zero slack algorithm* to minimize the power cost and to simplify the power network resource under timing constraints; Lee *et al.* [12] tackled the voltage assignment problem using an ILP based approach with a set of linear inequalities representing the timing requirements. An inevitable deficiency of all these post-floorplanning and post-placement works is that physical layout and power optimization are done separately, hence the solution quality is restricted to a certain extent.

Concurrent physical layout and power optimization will be much more favorable. Hu *et al.* [8] considered this simultaneous voltage assignment, voltage island generation, 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. The large search space made their approach relatively timing consuming. In one recent work [11], Lee *et al.* handled the voltage assignment problem under timing constraints by dynamic programming, and developed a power network aware floorplanner to generate floorplans with voltage islands after the voltage assignment step. Given a netlist represented by a directed acyclic graph (DAG), voltage assignment is first performed according to the timing requirements before the floorplanning step. Level shifters are then inserted into the nets according to the voltage assignment result. At last, a power-network aware floorplanner is invoked to pack the blocks such that the power-network resource, estimated as the sum of the half-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.

There are other works addressing issues like reliability [21] and temperature reduction [9] in SoC voltage island partitioning and floorplanning.

## 1.7 Contributions

In this thesis, we address the MSV driven floorplanning problem (definition 2 in section 1.5) in two stages, and propose two approaches to solve it.

In the first stage, we propose a floorplanning method for MSV designs that is tightly integrated with the island partitioning and voltage assignment steps. Simulated annealing is used with normalized Polish expression [18] as the floorplan representation. Normalized Polish expression is used because the slicing tree is a suitable data structure on which island partitioning and voltage assignment can be done *optimally and efficiently* given one slicing floorplan. Simulated annealing is adopted to perform the random search. 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 update 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 [8]). In this floorplanning framework, we can also generate islands with power down mode to optimize the total power consumption further. Our floorplanner can be extended to consider the number of level shifters and the ease of power network routing (proximity to power pins and shapes of voltage islands). By using this approach, we can achieve significant power savings (up to 50%) for all data sets, without any significant increase in area and wire length.

In the second stage, we explicitly take timing requirements into account in the MSV driven floorplanning problem by proposing another framework, which is able to simultaneously optimize power consumption and physical layout of the circuit with the given timing constraints being satisfied. We will first show that the voltage assignment problem under timing constraints can be solved optimally under some conditions, i.e., given a netlist and a number of working voltage levels for each module, a specific voltage level can be assigned to each module on the netlist in such a way that maximum power saving can be achieved without violating the timing constraints, if a feasible solution exists. This problem can be solved optimally when the delay choices (each delay choice corresponds to a choice of the working voltage level) are continuous in the real or integer domain. We will present a general formulation of this voltage assignment problem as a set of linear inequalities to represent the timing requirements, with an objective to minimize the sum of the power cost in each cell. This problem can be transformed to a convex cost network flow problem [2], after applying the Lagrangian relaxation technique and can be solved optimally by a cost-scaling algorithm [7] in polynomial time. We can then make use of this method to obtain a feasible voltage assignment solution in which each module has just a discrete number of delay choices. Because of the short running time of this cost-scaling algorithm and, particularly, its capability of handling efficiently incremental changes on edge costs, we can embed this voltage assignment step in a simulated annealing based floorplanner, so that power consumption and physical information can be taken into consideration simultaneously. We compared our approach with [11], the most updated previous work on this problem, on the data sets provided by the authors of [11]. Experimental results show that our approach performs much better than [11] in terms of both power saving and running time. An average of 24.61% (v.s. 6.68% by [11]) power saving can be achieved in an average running time of $273s$ (v.s. $1911s$ by [11]) with our approach on those data sets.

## 1.8 Thesis Organization

The remaining of this thesis is organized as follows. In chapter 2, a number of previous works addressing the voltage island generation problem in the floorplanning context are briefly discussed. A voltage island driven floorplanner is provided in chapter 3, and another floorplanning framework for MSV designs is proposed in chapter 4 to further take timing constraints into consideration. Finally, chapter 5 concludes this thesis.

□ **End of chapter.**

# Chapter 2

# Literature Review on MSV

## 2.1 Introduction

Voltage islands enable core-level power optimization for 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 floorplan design. A number of previous works have addressed this problem. According to the design stage at which MSV is applied, these previous works basically fall into two categories.

## 2.2 MSV at Post-floorplan/Post Placement Stage

### 2.2.1 "Post-Placement Voltage Island Generation under Performance Requirement"

This work [19] considered power versus design cost trade-off for the post-placement voltage island generation problem.

In this paper, a candidate placement is viewed as a 2-D array, and the value at each entry of the array is set to be the square of the required working voltage level of the cell occupies the corresponding entry position. Let $A$ be an $m \times n$ array, and $A[i][j]$ denotes the value of the element at position $(i, j)$. A subarray $A[l \dots r, b \dots u]$ is the rectangular region in $A$ with $(l, b)$ (resp. $(r, u)$) as the bottom-left (resp. upper-right) corner. A subarray is refereed as a rectangle region. Let $\mu(R) = \max_{(i,j)\in R} A[i][j]$ be the maximum value of all elements in a subarray rectangle $R$. The weight of a rectangle $R$ is defined as

$$\omega(R) = \sum_{(i,j)\in R} (\mu(R) - A[i][j])$$

The weight of a rectangle $R$ represents the power wastage resulted if rectangle $R$ form a voltage island, since the cells in one voltage island are restricted to work at the same voltage level, which should be $\sqrt{\mu(R)}$ as the voltage level assigned to a cell can not be lower than its required working voltage level.

A partitioning of A is a set of disjoint rectangles (subarrays) $\Pi = \{R_1, R_2, \dots, R_k\}$ that cover $A$; $k = |\Pi|$ is called the size of this partitioning. The weight of a partitioning $\Pi$, is defined as

$$\omega(\Pi) = \sum_{1 \le t \le k} \omega(R_t)$$

In their voltage island generation problem, they want to subdivide the placement region into a small number of voltage islands (where every cell in the same voltage island will eventually receive the same voltage), while keeping the total power consumption low.

They defined the Voltage-partitioning problem as follows:

**Definition 3 Voltage-partitioning Problem, or VPP** *Given an $m \times n$ array $A$ and an error threshold $\delta$, among all partitionings whose weight is at most $\delta$, find one with the smallest size. Let $\kappa(A, \delta)$ be the size of this optimal partitioning.*

They also defined the dual version of VPP, or DVPP. The dual version of this problem (DVPP) is defined as the problem of minimizing the weight of the partitioning with a bound on the size.

They first the dual DVPP problem under the slicing model can be solved optimally by dynamic programming, then proposed an approximation approach to handle primal VPP by using binary search.

Let $\omega_s^*(l \dots r, b \dots u)$ denote the minimum weight of any rectangle (subarray) $R$ in $A$ with at most $s$ sub-rectangles, where $R = A[l \dots r, b \dots u]$. The ultimate goal is to compute $w_k^*(l \dots r, b \dots u)$. At the base level, if $s = 1$, or if $\omega(R) = 0$, and $\omega_s^*(l \dots r, b \dots u)$ is set as $\omega(R)$. Otherwise, the recursion can be written as follows:

$$\omega_s^*(l \dots r, b \dots u) = \min_{1 \le t < s} \{$$
$$\min_{l \le i < r \le j < u} \left\{ \begin{array}{l} \omega_t^*(l \dots i, b \dots u) + \omega_{s-t}^*(i+1 \dots r, b \dots u), \\ \omega_t^*(l \dots r, b \dots j) + \omega_{s-t}^*(l \dots r, j+1 \dots u) \end{array} \right\} \}$$

The second minimization term enumerates all horizontal and vertical cuts, and the first term $(1 \le t < s)$ enumerates all possible ways to assign the number of rectangles ($t$ and $s-t$) for the two separated parts obtained by some horizontal or vertical cut.

This provides an optimal solution for the dual DVPP problem under the slicing model. Then, the approximation approach for the primal VPP problem under the same model can be easily obtained by guessing the optimal number of rectangles $\bar{k}$ in $O(log\bar{k})$ time by a binary search, starting with $\bar{k} = 1$.

### 2.2.2 "Post-Placement Voltage Island Generation"

Similar with the paper in the previous sub-section, this work [5] also targeted at minimizing the number of voltage islands generated on a candidate placement, while keeping the power wastage under a certain threshold. The main difference between this work and the previous one lies in that non-rectangular voltage islands are allowed in this work.

The non-rectangular voltage-partitioning problem (NVPP) is defined as follows:

**Definition 4 Non-rectangular Voltage-partitioning Problem, or (NVPP)** *Given an $m \times n$ array $A$ and an error threshold $\delta$, find a partitioning $\Pi$ of connected regions whose weight $\omega(\Pi)$ is at most $\delta$ and the size $|\Pi|$ is as small as possible.*

The input grid (the candidate placement) size $m \times n$ is usually huge and it is inefficient to work on it directly. Therefore, they first coarsen the grid $G$ to a size of $O(N)$ where $N$ is the number of cells, after which a new grid $G'$ is obtained, where each cell in the original grid $G$ belongs to one and only element of the new grid $G'$. At the second step, a tree $T$ of the elements in the coarsened grid is built according to their adjacencies and their differences in power requirement. Basically, this is a bottom-up recursive clustering of $G'$. At each level, pairs of adjacent nodes are clustered to form a "super-node" at the next level according to the amount of power wastage incurred. The binary tree $T$ is constructed to represent this multi-level clustering process, in which the leave nodes are the nodes in $G'$ and the internal nodes are the super-nodes. Subsequently, a dynamic programming will be applied to partition this tree optimally into subtrees (each subtree will correspond to a connected region) such that the total power wastage is minimized. Since the NVPP also targeted at minimizing the number of voltage islands generated, this dynamic programming procedure starts with $k = 1$, where parameter $k$ denotes the size of the partitioning. The procedure stops if the power wastage is below threshold $\delta$; else, $k$ is increased by 1 and the procedure is invoked again. Furthermore, this tree $T$ will be reconstructed and partitioned repeatedly, so that the nodes can be redistributed among subtrees to refine the solution.

### 2.2.3 "Timing-Constrained and Voltage-Island-Aware Voltage Assignment"

This paper [20] is also a work on post-floorplanning voltage island generation. They present an approximation algorithm to produce an initial voltage assignment that meets timing, while considering proximity of high voltage cells on a candidate floorplan.

They formulate the voltage assignment problem as a delay budgeting problem, with the special objective of physical proximity. Based on the framework of a classic delay budgeting algorithm, the zero slack algorithm (ZSA), they design an algorithm to allocate delay budget and assign voltage to cells towards an objective of physical proximity.

ZSA iteratively assigns delay budget to non-critical cells in the circuit. At each iteration, it first computes the slacks of all cells and find the minimum positive slack $s_{min}$; then it finds a path with this minimum positive slack, and distributes $s_{min}$ evenly among the cells on the path. Then it updates the delay of these cells and go to the next iteration. The algorithm stops when the slacks of all cells become zero. They adopted the iterative framework of ZSA to solve the voltage assignment problem. The major difference lies in how the path slack is allocated among the cells on each selected path, since the physical proximity of the high voltage cells has to be taken into consideration.

At first, they compute for each cell $c$ the delay budget $r_c$ it needs for voltage reduction. Let the slack of this cell be $s_c$. Those cells satisfying $s_c < r_c$ will be assigned high voltage because they can not have any voltage reduction. Then at each iteration, for each cell $c$ on the selected path, they calculate its distance to the closest high voltage cell, denoted as $d_c$. The larger this distance value is, the more likely it's sitting in a potentially non-critical low-voltage region, the more cost there is if this cell becomes a high voltage cell, thus, they give higher priority to it for voltage reduction. Therefore they sort all cells on the selected path with decreasing value of $d_c$, and allocate the path slack in the order of this sorted list, giving each cell the delay budget it needs for voltage reduction. If at certain cell in the list, the remaining slack is less than its needed delay budget, then it will be skipped and the next one is proceeded. The allocation stops when it reaches the end of the list, or when there is no more slack left. Such allocation guarantees that high voltage are always assigned to cells as close to the existing high voltage cells as possible, thereby maintains the physical proximity of the high voltage cells.

### 2.2.4 "Voltage Island Generation under Performance Requirement for SoC Designs"

In this work [16], they considered how to perform voltage island floorplanning with multiple supply voltages for SoC design.

In their approach, a few candidate floorplan solutions are first generated based on area and interconnect cost using an ordinary floorplanner, then voltage assignment and partitioning are performed on these candidate floorplans using the ILP approach to identify the best candidate solution. Therefore, this work actually belongs to the post-floorplanning voltage island generation category, although floorplanning has been considered in their framework.

In their approach, the voltage assignment and island partitioning problem is formulated as a 0-1 integer linear program, when an input floorplan is given. They presented two formulations for the voltage assignment and voltage island generation problem. The first is exact but takes longer time to compute a solution. The second use some approximation techniques, and it can be solved much faster with insignificant sacrifice on solution quality. The goal of voltage level assignment is to reduce the total power consumption while controlling the level conversion area overhead and the power network complexity without compromising system performance. A fragmentation cost (number of adjacent cores operating at different voltages) is used to model the power network complexity when assigning working voltage levels to cores, but this cost is not related to the number of islands directly.

The separation between the floorplanning step and the voltage assignment step largely confined their solution quality. Besides, the ILP formulation made their approach relatively time consuming.

### 2.2.5 "An ILP Algorithm for Post-Floorplanning Voltage-Island Generation Considering Power-Network Planning"

In this paper [12], they presented a general formulation of this problem that considers level-shifter planning and power-network routing resources at post-floorplanning stage.

The problem inputs include a floorplan with a number of blocks, a netlist specifying the interconnections between the blocks, a timing constraint $T_{cycle}$ such that the delay of the critical path in the netlist can not exceed $T_{cycle}$, a constant $l$ that defines the relaxed block adjacency, and the legal working voltage levels of each block. The objective is to assign each block with an available supply voltage and insert necessary level shifters, such that the total power consumption and the total power-network routing resource are minimized, the total size of level shifters do not exceed the white space in the floorplan, and the timing constraint is satisfied.

They proposed a general problem formulation which considers level-shifter planning and power-network routing resource. Moreover, the cost function is dynamic in nature, since they do not have the information on the number of voltage islands during the optimization. To capture the block adjacency in a floorplan for voltage-island generation, they proposed a graph-based representation such that the number of nodes in the graph is linear to the number of blocks. To tackle the addressed problem, an ILP formulation was employed, in which they proposed (1) a box-growing wirelength estimation method to predict the timing overhead incurred by level shifters, (2) a set of linear voltage-island-clustering inequalities to avoid complicated constraint transformations, and (3) a set of linear inequalities to capture the usage of the power-network routing resource. Solving this ILP would give a solution to the problem addressed.

## 2.3 MSV at Floorplan/Placement Stage

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

In this paper [8], the authors addressed the problem of voltage islands planning for core-based SoC designs, which involves partitioning cores into several islands and floorplanning both at chip and island-level. A graph-based representation is used so that the partitioning and floorplanning steps are modeled in an integrated fashion.

Their approach is based on simulated annealing which guides the floorplanning and the island merging processes through a Voltage Island Compatibility Graph (VICG) graph $G(V,E)$, a complete undirected graph which captures the current voltage island partitioning solution in an abstract way. Each vertex $v_i \in V$ denotes a voltage island, and each edge $e_{ij}$ characterize the "attraction" between island $v_i$ and island $v_j$ by its weight, which is dependent on and interconnections between the two islands as well as their working voltage levels. In this way, the floorplanner can be guided in such a way that two compatible voltage islands (working at the same voltage level) with heavy interconnections are placed adjacently.

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. 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. The island merging process consists of detecting the regions that contain potentially mergeable islands, and eventually merging them. 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. The cost function is a weighted sum of different metrics including the number of islands, average power consumption, and area overhead.

The solution perturbations during their annealing process contain island split move, island voltage change move, and multi-island voltage change move, which produce a large search space and thus make their approach quite timing consuming.

### 2.3.2 "Voltage Island Aware Floorplanning for Power and Timing Optimization"

In this work [11], Lee *et al.* handled the voltage assignment problem under timing constraints by dynamic programming, and developed a power network aware floorplanner to generate floorplans with voltage islands after the voltage assignment step. In the following the approach of this paper is explained in more details, as we will mainly compare with this paper in chapter 4. Here is their problem formulation.

**Definition 5 Multi-Voltage Floorplanning [MVF] Problem** *Given multiple supply-voltage choices, a set of blocks, a netlist, a static-timing and a fixed-outline constraints, assign each block with a supply voltage and its coordinate in a floorplan so that the power consumption and the power-network resource requirement are minimized and both the static-timing and fixed-outline constraints are satisfied.*

### Algorithm Flow

The flow consists of three phases: (I)voltage assignment, (II) level-shifter (block) insertion, and (III) power-network aware floorplanning. For Phase I, they present a dynamic programming (DP) based method to solve the voltage assignment problem. As supply voltages are assigned to the circuit blocks in Phase I, they check in Phase II whether a net needs a level shifter and insert one as a soft block if needed. Finally in Phase III, they transform the precomputed slack into the wire-length constraint and perform floorplanning on all blocks, circuit blocks and level shifters (soft blocks), to minimize the power-network resource requirement. The floorplanning is based on simulated annealing (SA) using the $B^*$-tree floorplan representation [4].

In Phase I, they first represent the netlist by a DAG, and then handle voltage assignment by dynamic programming (DP) on the DAG. Given a netlist without reconvergent fanouts, the DP can guarantee an optimal solution for the voltage assignment. Phase I contains mainly the following steps.

*Step 1. DP-curve initialization.* Represent the delay-power properties of a block as a Delay-Power-curve (DP-curve). DP-curve is a monotonic decreasing chain as shown in Figure 2.1.

*Step 2. Joint Curve Generation.* For an unvisited node in the DAG, they will

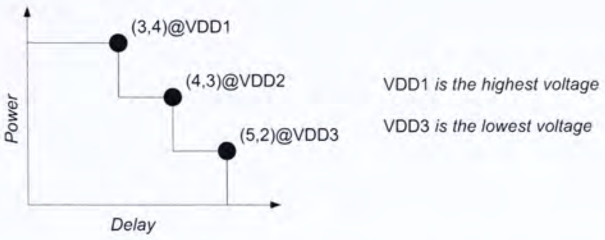


Figure 2.1: An Example DP-Curve (The three points of the DP-curve represent the delay-power characteristics of different supply voltages)

calculate the joint curve of this node and each of its fanins, after which a set of joint curves will be generated.

*Step 3. Lower-Bound Merge Operation.* During this step the set of previously generated joint curves will be merged into one curve.

*Step 4. Redundant Point Pruning.* After all the points on the DP-curve have been produced after the merging operation, a pruning procedure will be employed to delete those redundant points which are dominated by others in terms of both power and delay, so that a monotonic decreasing DP-curve can be obtained for the node that is currently visited.

*Step 5. Solution Backtracing.* Having generated the new accumulative DP-curve at POs, they trace a netlist to get an good solution of voltage assignment. Among all the points that satisfy the timing constraints in the accumulative DP-curve at a PO, the one with lowest power consumption will be chosen, so the delay and power of the whole circuit will be determined. Consequently, they repeat tracing solutions until PIs to assign a voltage level for each core.

In the Phase II of their algorithm flow, level shifters are inserted into a net that connects two blocks in different power domains. After voltage assignment, they trace the circuits from PI's to PO's to search the nets that need level shifters by

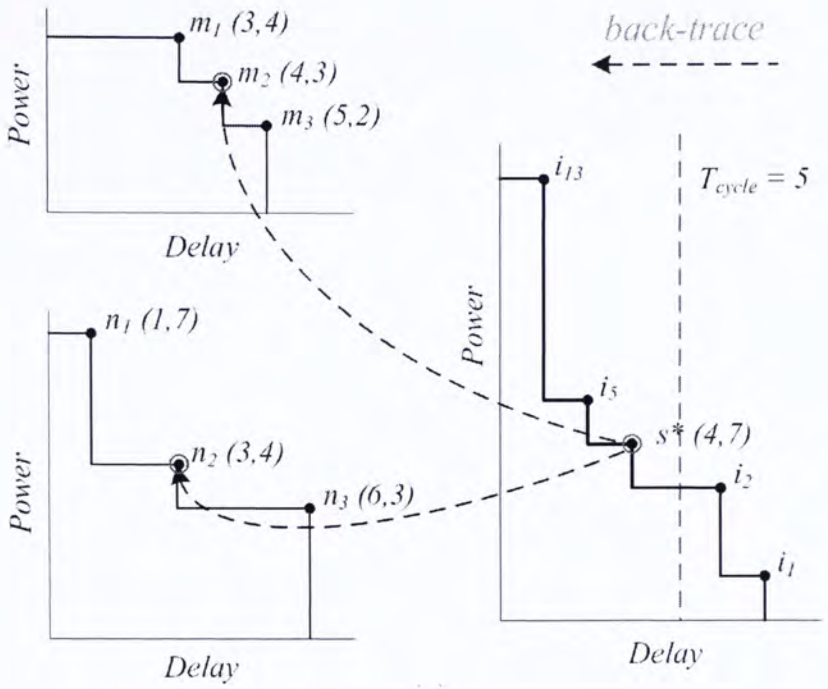


Figure 2.2: The backtracing procedure in [11] for getting a solution

breadth-first search. Those level shifters are treated as soft blocks during floor-planning stage.

In Phase III, they conduct power-network aware floorplanning for the original hard blocks and the additional level-shifter (soft) blocks together to make the critical paths satisfy the timing constraint. The objective in this phase is to find a floorplan which simultaneously minimizes the power-network resource requirement and satisfies the timing and the fixed-outline constraints. During the annealing process of floorplanning, they adopted the following cost function to minimize the powernetwork resource without violating the constraints.

$$\Phi(T) = \alpha\Phi_{PNR} + (1-\alpha)\Phi_{area} + \Phi_{timing} + \Phi_{outline}$$

where $\Phi_{PNR}$ denotes the cost of power-network resource, $\Phi_{area}$ is the area of the floorplan, and $\alpha$ is a weighting factor. $\Phi_{timing}$ and $\Phi_{outline}$ are the penalty for timing and outline violation, respectively.

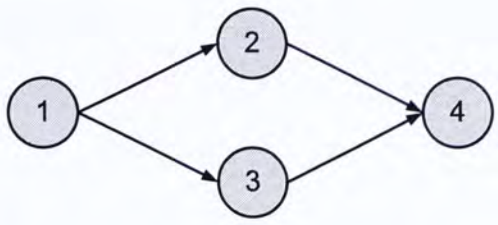


Figure 2.3: A reconvergent structure in the netlist

This paper proposed a reasonable design flow, however, their solution quality suffers in several aspects. First of all, the optimality of its voltage assignment step can only be guaranteed when there are no reconvergent structures in the netlist, but there are surely many of such structures in real circuits, which can significantly worsen the quality of their voltage assignment result. Fig. 2.3 shows an example of a reconvergent structure, in which the signals emanating from vertex one re-converges at vertex four, after traversing two disjoint paths. Secondly, physical information is not taken into account while performing the voltage assignment step. Lastly, the length of an interconnect is restricted to stay within a certain range during floorplanning in order to satisfy the timing constraints, which may degrade the solution quality.

## 2.4 Summary

In these previous works, MSV is considered at various design stages, including the floorplanning and placement stages [8], [11], and the post-floorplanning and post-placement stages [5], [12], [16], [19], [20]. For the latter category, Wu *et al.* [19] and Ching *et al* [5] targeted at minimizing the number of voltage islands generated, and Mak *et al.* [16] formulated the voltage assignment and island generation problem as an *integer linear program* (ILP). The other two works also considered timing. In [20], Wu *et al.* proposed an approximation algorithm based on a *zero* *slack algorithm* to minimize the power cost and to simplify the power network resource under timing constraints; Lee *et al.* [12] tackled the voltage assignment problem using an ILP based approach with a set of linear inequalities representing the timing requirements. An inevitable deficiency of all these post-floorplanning and post-placement works is that physical layout and power optimization are done separately, hence the solution quality is restricted to a certain extent. Concurrent physical layout and power optimization will be much more favorable. In [11], voltage islands are generated during floorplanning, however, the voltage assignment step is done prior to floorplanning, which largely restricted their solution quality. Hu *et al.* [8] considered this simultaneous voltage assignment, voltage island generation, and floorplanning problem, however, the large solution space of their approach rendered their approach quite time consuming, besides, they didn't explicitly considered timing in their problem formulation.

---

□ **End of chapter.**

# Chapter 3

# MSV Driven Floorplanning

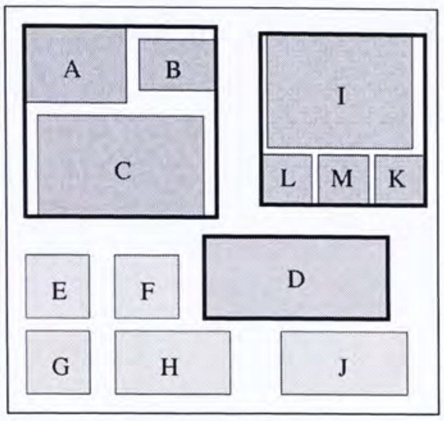
The content of this chapter is included in the proceedings of *IEEE/ACM International Conference of Computer-Aided Design (ICCAD)* 2007 [14].

## 3.1 Introduction

Energy efficiency has become one of the most important issues to be addressed in today's System-on-a-Chip (SoC) designs because of the increasing power density and the wide use of portable systems. There are two kinds of power consumption: dynamic and leakage. Dynamic power is caused by the charging and discharging of the load capacitance during switching. Leakage power is due to the sub-threshold currents when a device is turned off. There are many techniques to reduce power consumption. One of the most effective methods is by lowering the voltage supply. Multi-voltage design is thus introduced to provide "just enough" power to support different functional operations. Both dynamic and leakage power consumption can be reduced in multi-voltage designs. For dynamic power, since the consumption is proportional to the square of the voltage, a minor adjustment to the voltage level can result in a significant reduction. For leakage power, the consumption can be reduced by powering down parts of a chip when the functions are inactive.

Multi-voltage designs involve the partitioning of a chip into areas called "voltage islands" that can be operated at different voltage levels, or be turned off when idle. 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 significantly affect each other. In addition, there are other issues to be considered. For example, the voltage islands should be placed close to the power pins in order to minimize the power routing complexity and the 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 designs using voltage islands. An example is shown in Figure 3.1. 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 1.0V, core *D* on its own as one island at voltage 1.5V and cores *I*, *K*, *L* and *M* as one island at 1.2V. Notice that other cores will be operated at the chip-level voltage and the island containing *I*, *K*, *L* and *M* can be powered down during sleep. A candidate floorplan solution for such a partitioning and voltage assignment is shown on the left hand side.

In this chapter, we propose a floorplanning method for SoC designs that is tightly integrated with the island partitioning and voltage assignment steps. Simulated annealing is used with normalized Polish expression [18] as the floorplan representation. Normalized Polish expression is used because the slicing tree is



A: {1.0V, 1.1V}
B: {1.0V, 1.1V, 1.2V}
C: {1.0V, 1.1V}
D: {1.5V}
E: {1.1V, 1.5V}
F: {1.1V, 1.2V}
G: {1.1V, 1.2V}
H: {1.0V, 1.1V}
I: {1.2V}
J: {1.0V, 1.1V}
K: {1.1V, 1.2V}
L: {1.1V, 1.2V}
M: {1.1V, 1.2V}

Idle State Grouping:
Group 1: {D, I, K, L, M}
Power saving = 50%
Group 2: {H, J}
Power saving = 30%

Figure 3.1: An example of the voltage island driven floorplanning problem

a suitable data structure on which island partitioning and voltage assignment can be done *optimally and efficiently* given one slicing floorplan. Note that the restricted use of slicing floorplan can also give satisfactory results since many input cores have flexibilities in shape at this stage. Simulated annealing is adopted to perform the random search. 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 update 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 [8]). In this floorplanning framework, we can also generate islands with power down mode to optimize the total power consumption further. Our floorplanner can be extended to consider the number of level shifters and the ease of power network routing (proximity to power pins and shapes of voltage islands). By using this approach, we can achieve significant power savings (up to 50%) for all data sets, without any significant increase in area and wire length.

We will define the problem in section 3.2, then the methodology used will be discussed in section 3.3. Experimental results will be reported in section 3.6 before the conclusion and discussion in the last section.

## 3.2 Problem Formulation

In this problem, we are given a set of $n$ cores with areas $A_1, A_2 \dots A_n$ and aspect ratio bounds $[l_i, u_i]$ for $i = 1 \dots n$. 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. The power table of a core can be characterized by a core designer. For example, they can run simulations that try applying different supply voltages to the core, and a voltage level will be regarded as a legal one as long as the timing assertion can be satisfied [8]. We assume that the given voltage levels of each core can meet timing, so we do not consider timing explicitly in our formulation. The power consumption corresponding to each legal voltage can then be estimated. In this work, we compute the power cost of a core $i$ operated at voltage $v$ as $v^2 A_i$. 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.

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. 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.

## 3.3 Algorithm Overview

Our floorplanner is based on simulated annealing using normalized Polish expression (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.

## 3.4 Optimal Island Partitioning and Voltage Assignment

Given a candidate floorplan solution represented by a normalized Polish expression, 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 pseudo code 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 the 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 normalized Polish expression 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 non-zero $k$, 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: 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 $V_c$; 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 $V_c$; 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 sub-section); case 4 (line 8), the entire subtree $u$ is regarded as one voltage island and the power consumption $cost(\{u\})$ will be computed as $(v_{\{u\}})^2 A_{\{u\}}$, 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 (line 14-16). The minimum one will be returned and recorded in the table.

### 3.4.1 Voltage Islands in Non-subtrees

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 operational type. An example is shown in Figure 3.2. Therefore, we need the procedure *nonSubtree*() to enumerate these cases. The pseudo code 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 Figure 3.2).

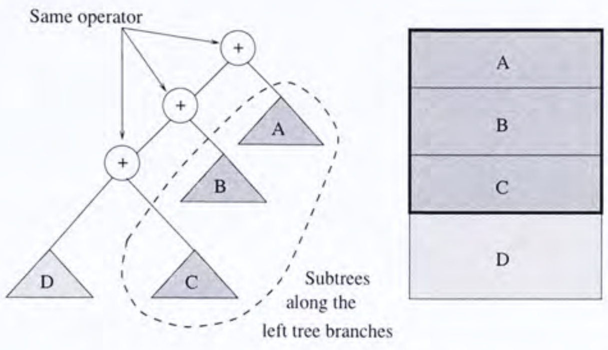
**Pseudocode** *nonSubtree(u, k)*

```
// 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. $min\_cost = \infty$
2. $S = rchild(u)$
3. $op = operator(u)$
4. While $operator(lchild(u))$ is op,
5.     $u = lchild(u)$
6.     $S = S \cup rchild(u)$
7.     $C = TreePart(lchild(u), k-1) + cost(S)$
8.     If $min\_cost > C$, $min\_cost = C$
9. Return($min\_cost$).
```

### 3.4.2 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 $T_u$. Three of the 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



A, B and C, not in one subtree, can form one voltage island.

Figure 3.2: An example of forming an island across subtrees

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.

### 3.4.3 Handling Island with Power Down Mode

Voltage 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 procedures

*TreePart*() and *nonSubtree*(), we only need to check if all the cores within this island belong to one group $G_i$ for some $i = 1 \dots p$. If this is true, the island formed can be shut down during sleep and have an additional power saving of $s_i\%$. 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.

### 3.4.4 Speedup in Implementation and Complexity

A table $cost\_table[v][j]$ for $j = 1 \dots K$ 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 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 (line 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.

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.

### 3.4.5 Varying Background Chip-level Voltage

In the problem formulation, it is assumed that a background chip-level voltage $V_c$ is given. Our approach can also be applied when there is no such voltage and the final chip-level voltage is determined by the minimum feasible voltage level among all the cores which are not grouped into any voltage island. To achieve this, we only need to expand the table at each node $L$ times where $L$ is the 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 $1 \leq l \leq L$. Notice that all entries in the tables corresponding to different chip-level voltages can be updated simultaneously during the recursive calls and the run time is only linearly scaled up by $L$. This is very affordable in practice since the number of possible voltage levels $L$ is usually small.

## 3.5 Simulated Annealing

### 3.5.1 Moves

There are three kinds of moves to change the normalized Polish expression of a candidate floorplan solution in the annealing process. 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.

### 3.5.2 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 method and $P$ is the total power consumption. The parameters $\lambda_w$ and $\lambda_p$ 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. This cost function can be modified to consider the fixed-outline constraint by replacing the area term $A$ (since we are not minimizing the area) by $\lambda_\infty(\max\{0, w - W'\} + \max\{0, h - H'\})$ where $\lambda_\infty$ 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.

## 3.6 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 {1.0V, 1.1V, 1.2V, 1.3V, 1.5V} and 1.5V is assumed to be the chip-level voltage. In each data set, groups of blocks with similar inactive periods are also randomly generated. Table 3.1 shows the details of each data set, the fourth column indicates the grouping information of similar inactive periods, e.g., for n30, there are two groups: one contains five cores with an additional 30% power saving if being grouped together, and the other

Table 3.1: Data sets for MSV driven floorplanning

| 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(30%), 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%) |

one contains five cores with an additional 20% power saving if being grouped together. Our algorithm is implemented in the C programming language and all the experiments were performed on a Sun Blade 2500 with a 1.6 GHz CPU and 2 GB RAM.

The results are shown in Table 3.3. For each data set, we performed voltage island driven floorplanning with the number of voltage islands generated ranges from zero to six. We can 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 Figure 3.3 and Figure 3.4. Figure 3.3 is a resultant packing of data set $n100$, with four voltage islands generated. In Figure 3.4, we aim at testing a particular situation in which some cores can be operated at very low voltages (compared with the chip-level voltage). We use *playout* of the MCNC benchmark as the testing data set, and assign 0.6V to cores $\{2\text{-}9\}$ and 0.8V to cores $\{12\text{-}19\}$ respectively as their minimum working voltages, while the other cores' working voltages are all set to 1.5V. The floorplanning procedure is then performed with $K = 2$ and 1.5V 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 one 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 being operated at a higher voltage.

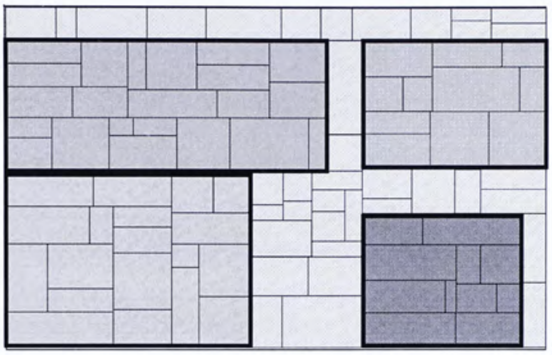


Figure 3.3: One Resultant Floorplan of $n100$ with Four Voltage Islands.

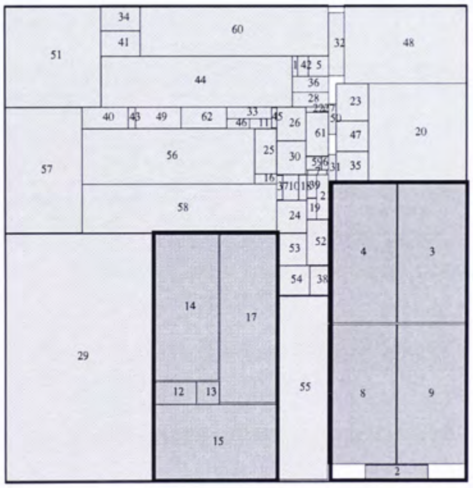


Figure 3.4: One resultant floorplan of *playout* with some blocks operated at very low voltages

Table 3.2: Comparisons of our MSV driven floorplanner with [16]

| Data Set | Power Saving (%) | | | Dead Space (%) | | | Run Time* (s) | | |
|---|---|---|---|---|---|---|---|---|---|
| | with idle island | without idle island | [16] | with idle island | without idle island | [16] | with idle island | without idle island | [16] |
| apte | 52.47 | 47.91 | 53.78 | 1.264 | 1.232 | 3.422 | 3.016 | 3.002 | 5.482 |
| xerox | 50.21 | 41.76 | 22.85 | 1.137 | 1.085 | 5.259 | 3.235 | 3.185 | 7.079 |
| hp | 47.85 | 39.19 | 25.37 | 1.324 | 1.318 | 5.700 | 3.827 | 3.367 | 122.9 |
| ami33 | 54.26 | 46.96 | 44.12 | 3.865 | 3.582 | 5.784 | 40.69 | 36.84 | 89.39 |
| ami49 | 52.63 | 45.50 | 41.13 | 3.791 | 3.805 | 6.440 | 98.36 | 91.23 | 90.46 |
| 2xerox | 50.86 | 41.76 | 33.53 | 2.062 | 3.067 | 3.765 | 10.64 | 9.026 | 74.75 |
| 2hp | 55.13 | 39.19 | 17.47 | 2.451 | 2.238 | 5.650 | 12.74 | 11.12 | 26.86 |
| ami75 | 49.32 | 40.05 | 39.06 | 4.379 | 4.871 | 6.330 | 276.1 | 257.2 | 316.5 |
| ami99 | 48.68 | 45.57 | 41.16 | 6.388 | 7.036 | 7.666 | 441.4 | 428.3 | 684.4 |
| ami200 | 43.87 | 39.88 | 41.90 | 8.116 | 8.011 | 10.88 | 1657 | 1598 | 3851 |
| ami300 | 42.54 | 40.54 | 40.69 | 10.08 | 11.42 | 12.02 | 3369 | 3243 | 7380 |
| Average | 49.78 | 42.57 | 36.46 | 4.078 | 4.333 | 6.655 | 537.8 | 516.7 | 1150 |

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

In order to compare with the previous related work on SoC designs [16], we have done another set of experiments with the benchmarks provided by the authors of [16]. In their data sets, the available voltage levels for each cell are chosen from the set $\{1.1V, 1.3V, 1.5V, 1.8V\}$. The comparisons are displayed in Table 3.2. Note that two different comparisons have been done, one of which enables the idle island option while the other disables it. Result shows that our approach is much more efficient and is able to save more power in most cases, while with less area overhead. In this set of experiments, we set $K = 4$ for all data sets, i.e., four voltage islands are generated for each test case.

Table 3.3: Experimental results of MSV driven floorplanning

| Data Set | K | Total Power | Power Saving (%) | Area | Dead Space (%) | Wire Length | Idle Island No. | Run Time (s) |
|---|---|---|---|---|---|---|---|---|
| n10 | 0 | 498718 | 0.000 | 223588 | 0.854 | 1832 | 0 | 3.236 |
| | 1 | 406653 | 18.47 | 223918 | 1.003 | 1865 | 1 | 3.259 |
| | 2 | 357723 | 28.28 | 224457 | 1.238 | 1986 | 1 | 3.343 |
| | 3 | 303755 | 39.16 | 224546 | 1.277 | 1929 | 1 | 3.351 |
| | 4 | 277569 | 44.35 | 224335 | 1.184 | 1897 | 1 | 3.374 |
| | 5 | 268342 | 46.22 | 224314 | 1.175 | 2106 | 1 | 3.388 |
| | 6 | 267145 | 46.44 | 224355 | 1.193 | 2075 | 1 | 3.454 |
| n30 | 0 | 469330 | 0.000 | 211720 | 1.476 | 8659 | 0 | 29.65 |
| | 1 | 322429 | 31.30 | 214040 | 2.543 | 8823 | 0 | 31.34 |
| | 2 | 251420 | 46.43 | 213453 | 2.278 | 8425 | 1 | 32.47 |
| | 3 | 243300 | 48.16 | 214763 | 2.874 | 9016 | 1 | 32.83 |
| | 4 | 237621 | 49.37 | 216481 | 3.645 | 8969 | 2 | 37.14 |
| | 5 | 233623 | 50.22 | 220572 | 5.432 | 9113 | 2 | 39.88 |
| | 6 | 228047 | 51.41 | 217022 | 3.885 | 9084 | 2 | 42.64 |
| n50 | 0 | 446802 | 0.000 | 200465 | 0.941 | 15324 | 0 | 90.26 |
| | 1 | 326612 | 26.90 | 200852 | 1.132 | 16233 | 0 | 94.25 |
| | 2 | 286310 | 35.92 | 202101 | 1.743 | 17659 | 0 | 95.73 |
| | 3 | 255838 | 42.74 | 202332 | 1.855 | 16983 | 1 | 95.46 |
| | 4 | 229835 | 48.56 | 202875 | 2.118 | 17241 | 2 | 100.3 |
| | 5 | 217369 | 51.35 | 202495 | 1.934 | 16815 | 3 | 113.7 |
| | 6 | 216788 | 51.48 | 206309 | 3.747 | 17927 | 2 | 134.1 |
| n100 | 0 | 417928 | 0.000 | 187899 | 1.146 | 17638 | 0 | 325.8 |
| | 1 | 365143 | 12.63 | 190173 | 2.338 | 17962 | 0 | 333.7 |
| | 2 | 350558 | 16.12 | 195880 | 5.174 | 18267 | 0 | 367.6 |
| | 3 | 327864 | 21.55 | 196858 | 5.645 | 18663 | 1 | 396.7 |
| | 4 | 307553 | 26.41 | 198599 | 6.472 | 18135 | 1 | 438.1 |
| | 5 | 282686 | 32.36 | 200187 | 7.214 | 19579 | 2 | 451.5 |
| | 6 | 250064 | 40.17 | 200429 | 7.326 | 18851 | 2 | 492.4 |
| n200 | 0 | 419223 | 0.000 | 188092 | 0.942 | 21501 | 0 | 1489 |
| | 1 | 382449 | 8.761 | 195633 | 4.762 | 22419 | 0 | 1523 |
| | 2 | 363843 | 13.21 | 196582 | 5.224 | 22946 | 1 | 1568 |
| | 3 | 350804 | 16.32 | 199979 | 6.831 | 23017 | 1 | 1626 |
| | 4 | 325484 | 22.36 | 203808 | 8.586 | 22838 | 1 | 1648 |
| | 5 | 295216 | 29.58 | 206267 | 9.670 | 23519 | 1 | 1723 |
| | 6 | 271572 | 35.22 | 209939 | 11.26 | 23485 | 2 | 1838 |
| n300 | 0 | 668290 | 0.000 | 302007 | 1.652 | 27924 | 0 | 2843 |
| | 1 | 624303 | 6.582 | 317022 | 6.310 | 28318 | 0 | 2869 |
| | 2 | 598988 | 10.37 | 325061 | 8.627 | 28254 | 0 | 2913 |
| | 3 | 563502 | 15.68 | 326493 | 9.028 | 29247 | 1 | 2966 |
| | 4 | 515518 | 22.86 | 331086 | 10.29 | 28761 | 1 | 3202 |
| | 5 | 483508 | 27.65 | 335348 | 11.43 | 29388 | 1 | 3635 |
| | 6 | 453167 | 32.19 | 342858 | 13.37 | 29289 | 2 | 3828 |

Table 3.4: Extending for minimization of level shifter number

| Data | No. of LS | | Power Saving (%) | | Run Time (s) | |
|---|---|---|---|---|---|---|
| | w/o LS Opt. | with LS Opt. | w/o LS Opt. | with LS Opt. | w/o LS Opt. | with LS Opt. |
| n10 | 43 | 27 | 44.35 | 38.25 | 3.374 | 4.635 |
| n30 | 79 | 45 | 49.37 | 35.89 | 37.14 | 42.82 |
| n50 | 192 | 137 | 48.56 | 40.63 | 100.3 | 108.9 |
| n100 | 267 | 62 | 26.41 | 17.82 | 438.1 | 440.6 |
| n200 | 345 | 96 | 22.36 | 15.34 | 1648 | 1702 |
| n300 | 569 | 126 | 22.86 | 12.86 | 3202 | 3228 |
| Avg | 249.2 | 82.1 | 35.65 | 26.80 | 904.8 | 920.2 |
| Diff(%) | -67.1 | | -24.82 | | +1.7 | |

### 3.6.1 Extension to Minimize Level Shifters

Level shifters (LS) are needed for connections between two blocks in different power domains. These level shifters will lead to area and delay overhead and should be minimized. We can extend our floorplanner to minimize the usage of level shifters by having an additional term in the cost function (with a weight determined by random walk before the annealing process) to represent the number of level shifters used. We assume that a level shifter is needed whenever a wire is connecting two blocks operating at different voltages[1]. 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 $C$. Since the operating voltage of each block is known after the voltage assignment and voltage island partitioning procedure, the number of level shifters needed can be counted trivially. The result is shown in Table 3.6.1. In this set of experiments, the numbers of voltage islands $K$ are all four. The results have shown that our method can reduce the number of level shifters by 67.1% on average with some penalty in power saving and run time.

[1]We can also consider adding a level shifter only when one core with a lower voltage level drives another core with a higher voltage level.

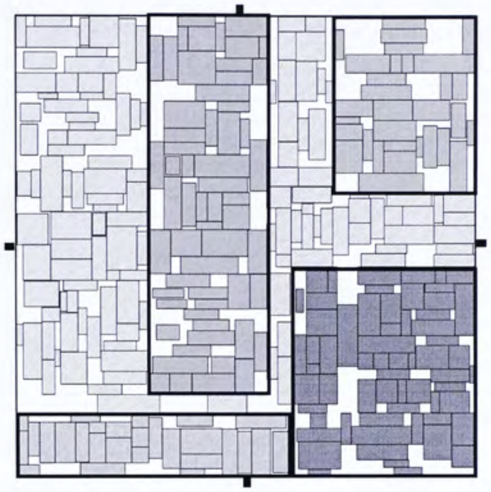
### 3.6.2 Extension to Consider Power Network Routing

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 [11]. 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* data set consisting of 300 blocks are shown in Figure 3.5, and they are produced with the fixed-outline constraint. The packing in Figure 3.5($a$) is obtained by the original floorplaner, without taking into consideration the power network issues, while the one in Figure 3.5($b$) 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. Besides the islands are closer to square in shape that will favor IR drop reduction and power network routing.

## 3.7 Summary

In this chapter, 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


($a$) Without considering proximity to power pins

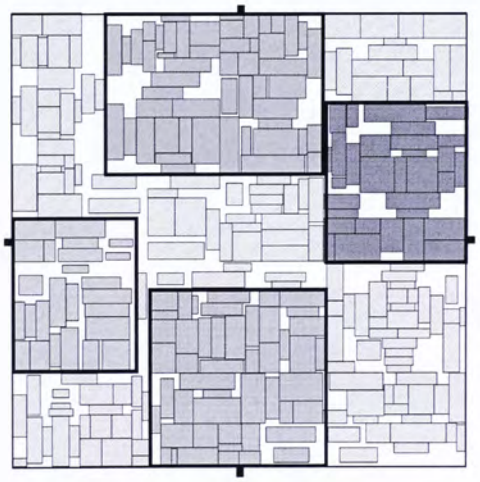

($b$) Considering proximity to power pins

Figure 3.5: Resultant floorplans considering proximity constraints to power pins

floorplan are concurrently taken into consideration. The experiment results have shown that we are able to achieve a significant power saving of up to 50% for the testing data sets.

In addition, when extended to minimize the number of level shifters, our method can reduce 67.1% of the LS used on average, with some penalty in power saving. It also functions well to generate voltage islands in proximity to their corresponding power pins, by having additional terms in the cost function to minimize the total distance between voltage islands and power pins, and to restrict the islands to be more square-like in shape.

---

□ **End of chapter.**

# Chapter 4

# MSV Driven Floorplanning with Timing

The content of this chapter will appear in the 26*th IEEE/ACM International Conference of Computer-Aided Design (ICCAD)* 2008 [15].

## 4.1 Introduction

In MSV designs, one prerequisite is an initial voltage assignment at the standard cell level that meets timing. Therefore, the timing slacks of the cells are traded for power saving by assigning appropriate supply voltage levels to the cells while still satisfying the timing constraints. Basically, high voltage level will be assigned to critical cells and low voltage level is assigned to non-critical cells, so that power can be saved without violating the timing constraints. Therefore, an effective and efficient voltage assignment method is desirable to minimize power consumption under timing requirements. Moreover, it is beneficial to consider the problem of voltage assignment, voltage island generation and floorplanning simultaneously under the timing, power, area and other physical constraints, since these steps will significantly affect each other.

A number of previous works addressed the voltage assignment and island generation problem in floorplanning and placement. In these previous works, some of them explicitly considered the timing constraints [11], [12] [13] [20], while others did not [5], [8], [9], [16], [19].

In one recent work [11], Lee *et al.* handled the voltage assignment problem under timing constraints by dynamic programming, and developed a power network aware floorplanner to generate floorplans with voltage islands after the voltage assignment step. Given a netlist represented by a directed acyclic graph (DAG), voltage assignment is first performed according to the timing requirements before the floorplanning step. Level shifters are then inserted into the nets according to the voltage assignment result. At last, a power-network aware floorplanner is invoked to pack the blocks such that the power-network resource, estimated as the sum of the half-perimeters of the voltage islands, will be minimized. As a result, blocks in the same voltage island will be placed close to each other. This paper proposed a reasonable design flow, however, their solution quality suffers in several aspects. First of all, the optimality of its voltage assignment step can only be guaranteed when there are no reconvergent structures in the netlist, but there are surely many of such structures in real circuits, which can significantly worsen the quality of their voltage assignment result. Secondly, physical information is not taken into account while performing the voltage assignment step. Lastly, the length of an interconnect is restricted to stay within a certain range during floorplanning in order to satisfy the timing constraints, which may degrade the solution quality.

In this chapter, we will first show that the voltage assignment problem under timing constraints can be solved optimally under some conditions, i.e., given a netlist and a number of working voltage levels for each module, a specific voltage level can be assigned to each module on the netlist in such a way that maximum power saving can be achieved without violating the timing constraints, if a feasible solution exists. This problem can be solved optimally when the delay choices (each delay choice corresponds to a choice of the working voltage level) are continuous in the real or integer domain. We will present a general formulation of this voltage assignment problem as a set of linear inequalities to represent the timing requirements, with an objective to minimize the sum of the power cost in each cell. This problem can be transformed to a convex cost network flow problem [2], after applying the Lagrangian relaxation technique and can be solved optimally by a cost-scaling algorithm [7] in polynomial time. We can then make use of this method to obtain a feasible voltage assignment solution in which each module has just a discrete number of delay choices. Because of the short running time of this cost-scaling algorithm and, particularly, its capability of handling efficiently incremental changes on edge costs, we can embed this voltage assignment step in a simulated annealing based floorplanner, so that power consumption and physical information can be taken into consideration simultaneously. We compared our approach with [11], the most updated previous work on this problem, on the data sets provided by the authors of [11]. Experimental results show that our approach performs much better than [11] in terms of both power saving and running time. An average of 24.61% (v.s. 6.68% by [11]) power saving can be achieved in an average running time of $273s$ (v.s. $1911s$ by [11]) with our approach on those data sets.

The remainder of this chapter is organized as follows. We define the problem in section 4.2, then the methodology used will be discussed in section 4.3. Experimental results will be reported in section 4.6 before the conclusion and discussion in the last section.

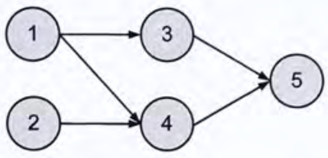


Figure 4.1: The DAG representation of a netlist

## 4.2 Problem Formulation

We are given a set of $n$ modules $m_1, m_2, ..., m_n$ with areas and aspect ratio bounds. Each module $m_i$ is given $k_i$ choices of supply voltages $v_q^i$, for $1 \leq q \leq k_i$, and the power-delay tradeoff in $m_i$ is represented by a *delay-power curve* (DP Curve), $\{(d_i^1, p_i^1), (d_i^2, p_i^2), ..., (d_i^{k_i}, p_i^{k_i})\}$, where each pair $(d_i^q, p_i^q)$ is the corresponding delay and power consumption when $m_i$ is operated at $v_q^i$. Note that each $d_i^q$ is in $Z^+$ (positive integer set), since the delay of a module is measured in terms of the number of clock cycles. Fig. 1.1(a) is an example of a DP Curve, which contains three choices of supply voltages for this module. Moreover, we assume that for each module, power is a convex function of delay when each point $(d_i^q, p_i^q)$ is connected to its neighboring point(s) by a linear segment in the DP Curve. We are also given a timing requirement $T_{cycle}$, called clock cycle time, of this circuit and require that the critical path delay is at most $T_{cycle}$.

We denote a netlist by a *directed acyclic graph* (DAG), $G = (V, E)$. Each vertex $i \in V$ denotes a module $m_i$, while each directed edge $e(i, j) \in E$ denotes an interconnect through which a signal is sent from $m_i$ to $m_j$. Fig. 4.1 is an example of the DAG representation of a netlist. In this DAG, each vertex is associated with a DP Curve representing the power-delay tradeoff of the corresponding module, while each edge is attributed with a wire delay.

To facilitate our problem formulation, we will first transform $G = (V, E)$ into $\bar{G} = (\bar{V}, \bar{E})$ in such a way that each vertex $i \in V$ is split into an input vertex and an

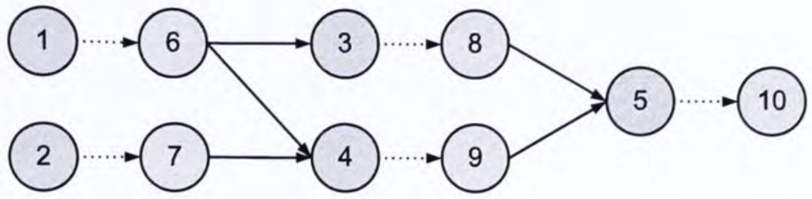


Figure 4.2: The transformed DAG $\bar{G}$ of the graph $G$ in Fig. 4.1

output vertex, and the input vertex is connected to the output vertex by a directed edge. We denote this set of newly created edges and the set of original edges by $E_1$ and $E_2$, respectively. Hence $\bar{E} = E_1 \cup E_2$. Fig. 4.2 illustrates the resultant DAG of the graph in Fig. 4.1 after this transformation, where dashed edges are in $E_1$ and solid edges are in $E_2$. After this transformation, the DP Curve of a vertex in $G$ will be associated with its corresponding edge in $E_1$. For example, after this transformation, vertex one and vertex six in $\bar{G}$ of Fig. 4.2 are, respectively, the input and output vertex of vertex one in $G$ of Fig. 4.1, thus the DP Curve of vertex one in $G$ is now associated with the edge $e(1,6)$ in $\bar{G}$, and we will use $k_{1,6}$ to denote the number of voltage level choices of this DP Curve. We can assume that each edge $e(i,j) \in E_2$ (corresponding to a wire) is also associated with a DP Curve in which the delay is an integer ranging from the wire's minimum delay to the maximum allowed delay $T_{cycle}$, and here we assume the power cost is zero at all points (the power cost on interconnects can be estimated as a function of total wire length and will be considered at the floorplanning level). Fig. 1.1$(b)$ is an example DP Curve corresponding to an edge $e(i,j) \in E_2$. Therefore, each edge $e(i,j) \in \bar{E}$ of $\bar{G}$ is now associated with one and only DP Curve, while the vertices in $\bar{G}$ simply represent the starting or ending point of a time interval and are not associated with any DP Curves. For each edge $e(i,j) \in \bar{E}$, we use $d_{ij}$ to denote the delay on it, and use $\bar{P}_{ij}$ to represent the function that maps delay to power consumption on edge $e(i,j)$, i.e., $\bar{P}_{ij}(d_{ij}^q) = p_{ij}^q$ for $1 \leq q \leq k_{ij}$ if $e(i,j) \in E_1$, and $\bar{P}_{ij}(d_{ij}) = 0$ if $e(i,j) \in E_2$. Moreover, we use $l_{ij}$ and $u_{ij}$ to denote the lower and upper bound of the delay $d_{ij}$

on edge $e(i,j)$, i.e., $l_{ij} \le d_{ij} \le u_{ij}, \forall e(i,j) \in \bar{E}$. For each edge $e(i,j) \in E_1$, $l_{ij} = d_{ij}^1$ and $u_{ij} = d_{ij}^{k_{ij}}$. For each edge $e(i,j) \in E_2$, $u_{ij}$ is $T_{cycle}$, while $l_{ij}$ is the minimum wire delay of the corresponding connection, and is estimated by scaling wire length to delay according to equation (4.1), when physical information is available.

$$l_{ij} = \delta \cdot wl_{ij} \tag{4.1}$$

where $wl_{ij}$ is the estimated length of the wire connecting module $i$ and module $j$, and $\delta$ is a constant scaling factor. Let $t_i \in Z^+$ denote the arrival time at vertex $i$ in $\bar{V}$, then it follows that

$$t_j - t_i \ge d_{ij}, \quad \forall e(i,j) \in \bar{E} \tag{4.2}$$

and

$$0 \le t_i \le T_{cycle}, \quad \forall i \in \bar{V} \tag{4.3}$$

The static timing constraint and the voltage assignment problem can be defined as follows:

**Definition 6 Static Timing Constraint** - *Given a clock cycle time $T_{cycle}$ and a DAG, $\bar{G} = (\bar{V}, \bar{E})$ corresponding to a netlist, the static timing constraint of the netlist is that the arrival time $t_i$ satisfies equation* (4.2) *and* (4.3), $\forall i \in \bar{V}$.

**Definition 7 Voltage Assignment Problem** - *Given a clock cycle time $T_{cycle}$ and a DAG, $\bar{G} = (\bar{V}, \bar{E})$ where each edge e is associated with a DP Curve, select a delay-power pair for each edge in its DP Curve such that the sum of power consumption is minimized while the static timing constraint is satisfied.*

According to the above definitions and notations, the voltage assignment problem can be easily formulated into the following mathematical program.

$$
\begin{array}{lllr}
\text{Minimize} & \sum_{e(i,j)\in\bar{E}} \bar{P}_{ij}(d_{ij}) & & (1a) \\
\text{Subject to} & & & \\
& t_j - t_i \geq d_{ij}, & \forall e(i,j)\in\bar{E} & (1b) \\
& 0 \leq t_i \leq T_{cycle}, & \forall i\in\bar{V} & (1c) \\
& l_{ij} \leq d_{ij} \leq u_{ij}, & \forall e(i,j)\in\bar{E} & (1d) \\
& d_{ij} \in \{d_{ij}^1, d_{ij}^2, \ldots, d_{ij}^{k_{ij}}\}, & \forall e(i,j)\in E_1 & (1e) \\
& d_{ij} \in Z^+, & \forall e(i,j)\in E_2 & (1f) \\
& t_i \in Z^+ & \forall i\in\bar{V} & (1g)
\end{array}
$$

Problem $(1)$ without the discrete choice constraint $(1e)$ is a *convex cost integer dual network flow problem* (or simply a *dual network flow problem*) [2], since its dual can be transformed to a minimum cost flow problem. This problem can be solved optimally in polynomial time, and we will show it in the following sections. Then, we will make use of the optimal solution to this problem to generate a feasible solution to the general cases of discrete choices for the module delay, with a power consumption approximating the optimal one.

In MSV designs, the power network resource should also be minimized. Given a floorplan, the power network resource $\Phi_q$ for the modules working at voltage $v_q$ can be estimated by the half-perimeter of the bounding box of all those modules [11], while the total power network resource is the sum of $\Phi_q$, for $q = 1, 2, ..., k$ where $k$ is the number of different voltage levels available. Our ultimate goal in this work is to solve the MSV driven floorplanning problem ( 2 in section 1.5).

## 4.3 Algorithm Overview

Globally, we use a simulated annealing based FAST-SP [17] floorplanner to explore the solution space. The voltage assignment problem (definition 7) is a subproblem of the MSV-driven floorplanning problem (definition 2), since the former is needed in order to evaluate a candidate solution for the latter problem. After each random move of the annealing process, the length of some interconnects may be changed which will cause changes in the minimum delay of the corresponding wires, thus $l_{ij}$ for those affected wires $e(i,j) \in E_2$ of $\bar{G}$ should be re-computed with equation (4.1) after the floorplan is realized. With these changes, the operating voltage level of each module is re-assigned by computing another solution to problem $(1)$. The power network resource is evaluated afterwards. In the following sub-section, we will first focus on solving the voltage assignment problem before discussing how this voltage assignment solver can be incorporated into a MSV-driven floorplanner.

## 4.4 Voltage Assignment Problem

The voltage assignment problem is equivalent to the mathematical program $(1)$ in Section 4.2. We will show in the following that problem $(1)$ without the discretization constraint $(1e)$ can be transformed into a minimum cost flow problem and can be solved optimally in polynomial time [2].

To simplify the problem, we will first make some modifications to the DP Curves. First of all, we connect each pair of neighboring points in a DP Curve by a linear segment, resulting in a piecewise linear convex function of power versus delay. It can be shown that there always exists an optimal solution to the dual network flow problem (problem $(1)$ without constraint $(1e)$) with integral variables [2]. Therefore, the constraints $(1f)$ and $(1g)$ can be removed.

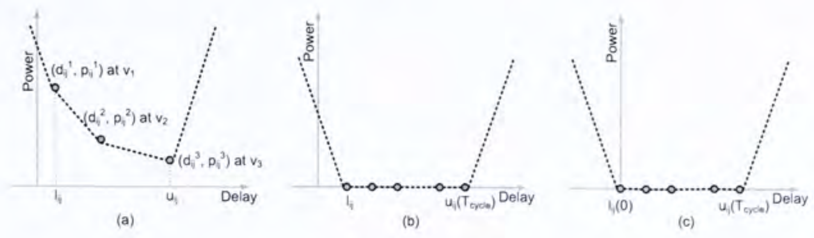


Figure 4.3: Delay power curve $P_{ij}(d_{ij})$ of $(a)$ an edge $e(i,j) \in E_1$; $(b)$ an edge $e(i,j) \in E_2$; $(c)$ an edge $e(i,j) \in E_3$

In the second step of simplification, the lower and upper bounds on variables $d_{ij}$ and $t_i$ in constraints $(1c)$ and $(1d)$ are eliminated. We first eliminate bounds on $d_{ij}$ by defining a new power delay function for each edge $e(i,j) \in \bar{E}$ as follows:

$$P_{ij}(d_{ij}) = \begin{cases} \bar{P}_{ij}(u_{ij}) + M \times (d_{ij} - u_{ij}), & if \quad d_{ij} > u_{ij} \\ \bar{P}_{ij}(d_{ij}), & if \quad l_{ij} \le d_{ij} \le u_{ij} \\ \bar{P}_{ij}(l_{ij}) - M \times (d_{ij} - l_{ij}), & if \quad d_{ij} < l_{ij} \end{cases}$$

where $M$ is a sufficiently large number, and here, we set $M = D + 1$, where $D = \sum_{e(i,j)\in E_1} \bar{P}_{ij}(l_{ij}) - \sum_{e(i,j)\in E_1} \bar{P}_{ij}(u_{ij})$. Actually, $M$ can be viewed as a penalty for violating the lower and upper bounds of the variables, and we set the penalty to be large enough to guarantee that a solution with variables violating the bounds is impossible to be an optimal one, if a feasible solution of problem $(1)$ exists. Therefore, by replacing $\bar{P}_{ij}(d_{ij})$ with $P_{ij}(d_{ij})$, we are able to remove the constraint $(1c)$ and $(1d)$, with variables in the optimal feasible solution guaranteed to stay in the range specified by $(1c)$ and $(1d)$. An example transformed DP Curve of an edge in $E_1$ and that of an edge in $E_2$ are illustrated in Fig. 4.3$(a)$ and Fig. 4.3$(b)$, respectively.

Similarly, we define $B_i(t_i)$ as follows and put it into the objective function so that the lower and upper bounds on $t_i$ can also be eliminated.

$$B_i(t_i) = \begin{cases} M \times (t_i - T_{cycle}), & if \quad t_i > T_{cycle} \\ 0, & if \quad 0 \le t_i \le T_{cycle} \\ -M \times (t_i), & if \quad t_i < 0 \end{cases}$$

After all the above simplifications, problem (1) without the discretization constraint $(1e)$ can be transformed into problem (2) as follows.

$$\begin{array}{lll} \text{Minimize} & \sum_{e(i,j)\in \bar{E}} P_{ij}(d_{ij}) + \sum_{i\in \bar{V}} B_i(t_i) & (2a) \\ \text{Subject to} & t_j - t_i \ge d_{ij}, \quad \forall e(i,j) \in \bar{E} & (2b) \end{array}$$

In the following sub-section, problem (2) is further simplified by using the Lagrangian relaxation technique.

## 4.4.1 Lagrangian Relaxation

The Lagrangian relaxation technique is useful to eliminate difficult constraints. Here, we want to eliminate the constraints $t_j - t_i \ge d_{ij}$ in $(2b)$. We apply the Lagrangian relaxation technique to problem (2), by introducing a nonnegative Lagrangian multiplier vector $\vec{x}$, and the corresponding Lagrangian sub-problem is to compute:

$$L(\vec{x}) = \min_{d,t} \sum_{e(i,j)\in\bar{E}} P_{ij}(d_{ij}) + \sum_{i\in\bar{V}} B_i(t_i) - \sum_{e(i,j)\in\bar{E}} (t_j - t_i - d_{ij})x_{ij}$$

where $x_{ij}$ is the nonnegative Lagrangian multiplier associated with the constraint $t_j - t_i \ge d_{ij}$. Note that:

$$\sum_{e(i,j)\in\bar{E}} (t_i - t_j)x_{ij} = \sum_{i\in\bar{V}} \Big( \sum_{j:e(i,j)\in\bar{E}} x_{ij} - \sum_{j:e(j,i)\in\bar{E}} x_{ji} \Big) t_i$$

We define

$$x_{0i} = \sum_{j:e(i,j)\in \bar{E}} x_{ij} - \sum_{j:e(j,i)\in \bar{E}} x_{ji}, \quad \forall i \in \bar{V}$$

Then we can rewrite $L(x)$ as follows:

$$L(\vec{x}) = \min_{d,t} \sum_{e(i,j)\in \bar{E}} (P_{ij}(d_{ij}) + x_{ij}d_{ij}) + \sum_{i\in \bar{V}} (B_i(t_i) + x_{0i}t_i)$$

In order to convert this problem to a network flow problem, we add an extra vertex 0 into $\bar{G}$, and add an edge $e(0,i)$ for each vertex $i \in \bar{V}$. Lets denote this set of newly added edges by $E_3$ and denote the resultant DAG by $G^* = (V^*, E^*)$, so that $V^* = \{0\} \cup \bar{V}$ and $E^* = E_1 \cup E_2 \cup E_3$. Particularly, for each newly added edge $e(0,i)$, we put $d_{0i} = t_i$, $l_{0i} = 0$, $u_{0i} = T_{cycle}$ and $P_{0i} = B_i$. The delay power curve $P_{ij}(d_{ij})$ of an edge $e(i,j) \in E_3$ is shown in Fig. 4.3$(c)$. Upon these notations, the Lagrangian sub-problem (problem (3)) can be restated as to compute:

$$L(\vec{x}) = \min_{d} \sum_{e(i,j)\in E^*} \{P_{ij}(d_{ij}) + x_{ij}d_{ij}\} \quad (3a)$$

Subject to

$$\sum_{j:e(i,j)\in E^*} x_{ij} - \sum_{j:e(j,i)\in E^*} x_{ji} = 0, \qquad \forall i \in V^* \qquad (3b)$$

$$x_{ij} \geq 0, \quad \forall e(i,j) \in E_1 \cup E_2 \quad (3c)$$

Since each $P_{ij}(d_{ij})$ is a piecewise linear convex function, $P_{ij}(d_{ij}) + x_{ij}d_{ij}$ is also a convex function of $d_{ij}$ for a given value of $x_{ij}$. It is important to note that, for a given value of vector $\vec{x}$, each term $\min_{d_{ij}}\{P_{ij}(d_{ij}) + x_{ij}d_{ij}\}$ can be optimized separately, since $d_{ij}$ is now independent of each other among all the edges $e(i,j) \in E^*$. We define function $H_{ij}(x_{ij})$ for each $e(i,j) \in E^*$ as follows:

$$H_{ij}(x_{ij}) = \min_{d_{ij}}\{P_{ij}(d_{ij}) + x_{ij}d_{ij}\}$$

Then problem (3) can be rewritten as to compute:

$$L(\vec{x}) = \sum_{e(i,j)\in E^*} H_{ij}(x_{ij}), \quad \text{subject to (3b) and (3c).}$$

The Lagrange dual problem is to determine $x^*$ such that

$$L(x^*) = \max_x L(x) = \max_x \sum_{e(i,j)\in E^*} H_{ij}(x_{ij}) \quad (4a)$$

Subject to

$$\sum_{j:e(i,j)\in E^*} x_{ij} - \sum_{j:e(j,i)\in E^*} x_{ji} = 0, \qquad \forall i \in V^* \qquad (4b)$$

$$x_{ij} \geq 0, \quad \forall e(i,j) \in E_1 \cup E_2 \quad (4c)$$

The following well-known theorem [3] establishes a connection between problem (2) and problem (4):

**Theorem 1** *Let $x^*$ be a solution to the Lagrange dual problem, then $L(x^*)$ equals the optimal objective value of the original convex dual network flow problem.*

## 4.4.2 Transformation into the Primal Minimum Cost Flow Problem

We first derive an explicit expression for $H_{ij}(x_{ij})$. As shown in Fig. 4.4, $P_{ij}(d_{ij})$ is a piecewise linear convex function. Let $(d_{ij}^q, P_{ij}(d_{ij}^q))$ be a break point (where the slope changes) on this function, and let $s^-$ and $s^+$ denote the left slope and the right slope at this point, respectively. We can prove the following two lemmas.

**Lemma 1** $H_{ij}(x_{ij}) = P_{ij}(d_{ij}^q) + x_{ij}d_{ij}^q$, *if* $-s^+ \leq x_{ij} \leq -s^-$.

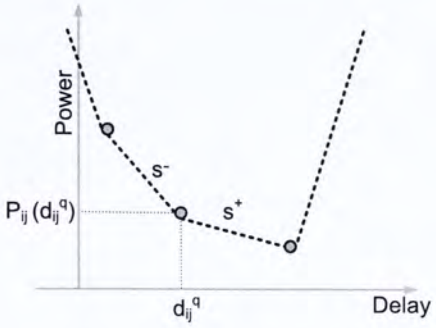


Figure 4.4: Left slope $s^-$ and right slope $s^+$ at $(d_{ij}^q, P_{ij}(d_{ij}^q))$ in a DP curve

**Proof**: When $d_{ij} \le d_{ij}^q$, let $f(d_{ij}) = s^- d_{ij} + c$ ($c$ is a constant) be the equation of the straight line on the left hand side of $d_{ij}^q$ (with slope $s^-$). Due to the convexity of $P_{ij}$, $P_{ij}(d_{ij}) \ge f(d_{ij})$, for $d_{ij} \le d_{ij}^q$. Thus $P_{ij}(d_{ij}) + x_{ij}d_{ij} \ge f(d_{ij}) + x_{ij}d_{ij} = (s^- + x_{ij})d_{ij} + c$. Note that $s^- + x_{ij} \le 0$, so $(s^- + x_{ij})d_{ij} + c$ is monotonically decreasing. Therefore, for $d_{ij} \le d_{ij}^q$, $P_{ij}(d_{ij}) + x_{ij}d_{ij} \ge (s^- + x_{ij})d_{ij} + c \ge (s^- + x_{ij})d_{ij}^q + c = P_{ij}(d_{ij}^q) + x_{ij}d_{ij}^q$. Similarly, when $d_{ij} \ge d_{ij}^q$, let $f(d_{ij}) = s^+ d_{ij} + c'$ ($c'$ is a constant) be the equation of the straight line on the right hand side of $d_{ij}^q$ (with slope $s^+$). Then $P_{ij}(d_{ij}) + x_{ij}d_{ij} \ge f(d_{ij}) + x_{ij}d_{ij} = (s^+ + x_{ij})d_{ij} + c'$. Since $s^+ + x_{ij} \ge 0$, $(s^+ + x_{ij})d_{ij} + c'$ is monotonically increasing. Therefore, $P_{ij}(d_{ij}) + x_{ij}d_{ij} \ge (s^+ + x_{ij})d_{ij} + c' \ge (s^+ + x_{ij})d_{ij}^q + c' = P_{ij}(d_{ij}^q) + x_{ij}d_{ij}^q$, for $d_{ij} \ge d_{ij}^q$. Combining the two cases completes the proof. □

**Lemma 2** *$H_{ij}(x_{ij}) = -\infty$ if $x_{ij} < -M$ or $x_{ij} > M$.*

**Proof**: Consider the DP Curve $P_{ij}(d_{ij})$ of an edge $e(i,j)$ in $E^*$. If $x_{ij} < -M$, $P_{ij}(d_{ij}) + x_{ij}d_{ij}$ is minimum (with a value of $-\infty$) when $d_{ij}$ tends to $\infty$ since $x_{ij}d_{ij}$ dominates $P_{ij}(d_{ij})$ when $d_{ij}$ tends to $\infty$ (the slope of the rightmost segment of $P_{ij}$ is $M$). Similarly, if $x_{ij} > M$, $P_{ij}(d_{ij}) + x_{ij}d_{ij}$ is minimum (with a value of $-\infty$)

when $d_{ij}$ approaches $-\infty$ since $x_{ij}d_{ij}$ dominates $P_{ij}(d_{ij})$ as $d_{ij}$ tends to $-\infty$ (the slope of the leftmost segment of $P_{ij}$ is $-M$). Therefore, we can conclude that $H_{ij}(x_{ij}) = \min_{d_{ij}}\{P_{ij}(d_{ij}) + x_{ij}d_{ij}\} = -\infty$, if $x_{ij} < -M$ or $x_{ij} > M$. $\square$

Recall that the objective of the Lagrange dual problem is to maximize $\sum_{e(i,j)\in E^*} H_{ij}(x_{ij})$ over $\vec{x}$. Therefore, we can safely add the constraint $-M \le x_{ij} \le M\ \forall e(i,j) \in E^*$, according to Lemma 2.

Note that, for the function $H_{ij}(x_{ij})$ corresponding to an edge $e(i,j) \in E_1$, the Lagrangian multiplier $x_{ij}$ must be nonnegative, therefore, it can be further bounded as $0 \le x_{ij} \le M\ \forall e(i,j) \in E_1$. Together with Lemma 1 and the DP Curve in Fig. 4.3$(a)$, we are able to express $H_{ij}(x_{ij})$ of each edge $e(i,j) \in E_1$ in the following manner.

$$H_{ij}(x_{ij}) = \begin{cases} P_{ij}(d_{ij}^{k_{ij}}) + d_{ij}^{k_{ij}}x_{ij}, & 0 \le x_{ij} \le b_{ij}(k_{ij}) \\ P_{ij}(d_{ij}^{k_{ij}-1}) + d_{ij}^{k_{ij}-1}x_{ij}, & b_{ij}(k_{ij}) \le x_{ij} \le b_{ij}(k_{ij}-1) \\ \vdots & \\ P_{ij}(d_{ij}^{q}) + d_{ij}^{q}x_{ij}, & b_{ij}(q+1) \le x_{ij} \le b_{ij}(q) \\ \vdots & \\ P_{ij}(d_{ij}^{1}) + d_{ij}^{1}x_{ij}, & b_{ij}(2) \le x_{ij} \le M \end{cases}$$

where $b_{ij}(q) = \frac{P_{ij}(d_{ij}^{q-1}) - P_{ij}(d_{ij}^{q})}{d_{ij}^{q} - d_{ij}^{q-1}}$. Note that it directly follows from the convexity of $P_{ij}(d_{ij})$ that $b_{ij}(q+1) \le b_{ij}(q)$.

Similarly, the constraint $0 \le x_{ij} \le M$ can also be added for each function $H_{ij}(x_{ij})$ that corresponds to an edge $e(i,j) \in E_2$, since the variable $x_{ij}$ of an edge $e(i,j) \in E_2$ is also a nonnegative Lagrangian multiplier. According to Lemma 1 and the power delay curve $P_{ij}(d_{ij})$ in Fig. 4.3(b), the corresponding $H_{ij}(x_{ij})$ of an edge $e(i,j) \in E_2$ can be written as follows:

$$H_{ij}(x_{ij}) = l_{ij}x_{ij}, \quad 0 \le x_{ij} \le M$$

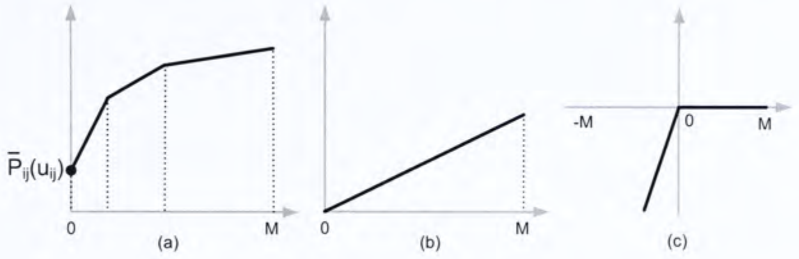


Figure 4.5: Function $H_{ij}(x_{ij})$ of (a) an edge in $E_1$; (b) an edge in $E_2$; and (c) an edge in $E_3$

For the function $H_{ij}(x_{ij})$ corresponding to an edge $e(i,j) \in E_3$, the variable $x_{ij}$ is not a Lagrangian multiplier, and it is bounded by $-M \leq x_{ij} \leq M$, according to Lemma 2. The corresponding function $H_{ij}(x_{ij})$ can be written as follows, according to Lemma 1 and the delay power curve in Fig. 4.3($c$).

$$H_{ij}(x_{ij}) = \begin{cases} T_{cycle}x_{ij}, & -M \leq x_{ij} \leq 0 \\ 0, & 0 \leq x_{ij} \leq M \end{cases}$$

Fig. 4.5 shows the bounds and shapes of all these functions $H_{ij}(x_{ij})$. It is easy to observe that these functions $H_{ij}(x_{ij})$ are piecewise linear concave. Then, we let $C_{ij}(x_{ij}) = -H_{ij}(x_{ij})$, so that $C_{ij}(x_{ij})$ is a piecewise linear convex function. We can subsequently transform problem (4) into problem (5) as shown below, by replacing $H_{ij}(x_{ij})$ with $-C_{ij}(x_{ij})$.

$$\text{Minimize} \quad \textstyle\sum_{e(i,j)\in E^*} C_{ij}(x_{ij}) \quad (5a)$$

Subject to

$$\sum_{j:e(i,j)\in E^*} x_{ij} - \sum_{j:e(j,i)\in E^*} x_{ji} = 0, \qquad \forall i \in V^* \qquad (5b)$$

$$-M \le x_{ij} \le M \qquad \forall e(i,j) \in E_3 \qquad (5c)$$

$$0 \le x_{ij} \le M \quad \forall e(i,j) \in E_1 \cup E_2 \quad (5d)$$

## 4.4.3 Cost-Scaling Algorithm

Problem (5) is a convex cost flow problem in which the cost associated with the flow on an edge $e(i,j)$ is a piecewise linear convex function containing a number of linear segments. We can transform problem (5) into a minimum cost flow problem in an expanded network $G' = (V', E')$, and solve it using a cost-scaling algorithm [7].

In the transformation from $G^*$ to $G'$, each edge $e(i,j)$ in $G^*$ will be replaced by one or more edges, depending on the number of linear segments in the corresponding function $C_{ij}(x_{ij})$. There are three categories of edges to consider, namely, $E_1$, $E_2$, and $E_3$:

• *Edge in* $E_1$: An edge in $E_1$ represents an input and output relationship of a module. For each edge $e(i,j) \in E_1$, $k = k_{i,j}$ edges are introduced in $G'$, with one edge corresponding to one linear segment in the cost function $C_{ij}$. These edges have costs $-d_{ij}^k$, $-d_{ij}^{k-1}$, $-d_{ij}^{k-2}$, ..., $-d_{ij}^1$, upper capacities $b_{ij}(k)$, $b_{ij}(k-1) - b_{ij}(k)$, $b_{ij}(k-2) - b_{ij}(k-1)$, ..., $M - b_{ij}(2)$, respectively, and lower capacities of zero for all of them.

• *Edge in* $E_2$: An edge in $E_2$ represents an interconnect. Each edge $e(i,j) \in E_2$ is directly copied to $G'$, with its cost, lower and upper capacity set as $-l_{ij}$, 0 and $M$ respectively.

• *Edge in* $E_3$: Edges belonging to this category emanate from vertex 0 in $G^*$. For each of them, two edges are introduced in $G'$. The cost, lower capacity and upper capacity are respectively $-T_{cycle}$, $-M$ and 0 for one edge, and 0, 0 and $M$ for the other edge.

It is well known that solving the convex cost flow problem in $G^*$ is equivalent to solving the minimum cost flow problem in $G'$.[1] Particularly, due to the fact that the costs of the edges (corresponding to delay) are all integers in our problem, we can directly apply the cost-scaling algorithm on $G'$ to find a minimum cost flow (as the correctness of the cost-scaling algorithm in [7] relies on the fact that all edge costs must be integer). Since this cost-scaling algorithm is a commonly known method to solve the minimum cost network flow problem, we will just briefly explain its major steps in the following, and more details can be found from [7], [6].

The cost-scaling algorithm maintains a pseudoflow $x$ at each step such that $x$ satisfies the upper and lower capacity constraints for all the edges but might violate the flow conservation constraint. For any pseudoflow $x$, the excess of a vertex $i$ is defined as $exc(i) = \sum_{j:e(j,i)\in E'} x_{ji} - \sum_{j:e(i,j)\in E'} x_{ij}$ for all $i \in V'$. Vertex $i$ is called an excess vertex if $exc(i) > 0$. A potential $\pi(i)$ is maintained for each vertex $i$. The cost-scaling algorithm proceeds by constructing and manipulating a residual network $G'(x)$ with respect to a pseudoflow $x$. The residual network consists only of edges with positive residual capacity. For a given residual network $G'(x)$ and a potential vector $\pi$, the reduced cost of an edge $e(i,j)$ is defined as $c^{\pi}_{ij} = c_{ij} - \pi(i) + \pi(j)$, where $c_{ij}$ is the cost of edge $e(i,j)$. A flow or a pseudoflow $x$ is said to be $\varepsilon$-optimal for some $\varepsilon \geq 0$ if for some potential vector $\pi$, $c^{\pi}_{ij} \geq -\varepsilon$ for every edge $e(i,j)$ in the residual network $G'(x)$. The cost-scaling algorithm treats $\varepsilon$ as a parameter and iteratively obtains $\varepsilon$-optimal flows for successively smaller values of $\varepsilon$. Initially, $\varepsilon = \max\{|c_{ij}| : e(i,j) \in E'\}$ ($T_{cycle}$ in our problem) and any feasible flow is $\varepsilon$-optimal. The algorithm then performs cost-scaling phases by repeatedly applying an improve-approximation procedure that transforms an $\varepsilon$-optimal flow into an $\varepsilon/2$-optimal flow. The improve-approximation procedure terminates when there are no excess vertices. After $O(log(n'T_{cycle}))$ cost-scaling

[1]There is a constant difference ( $\sum_{e(i,j)\in E_1} \bar{P}_{ij}(u_{ij})$ ) between the cost of minimum convex cost flow in $G^*$ and the cost of minimum cost flow in $G'$.

phases, $\varepsilon < 1/n'$ and the algorithm terminates with an optimal flow where $n'$ is the number of vertices in the network $G'$.

**Time complexity analysis**: The cost-scaling algorithm solves the minimum cost flow problem in $O(n'm'log(n'^2/m')log(n'U'))$ time, with $n'$, $m'$ and $U'$ being the number of vertices, the number of edges and an upper bound of the edge costs, respectively. As for our problem, it is not difficult to see that there are $O(nk)$ vertices and $O(m+nk)$ edges in the expanded network $G'$ where $n$ and $m$ are the numbers of vertices and edges in the original DAG $G = (V, E)$, and $k$ is the maximum number of voltage level choices for each module. Therefore, the time complexity of solving our power optimization problem under timing constraint by this approach is $O(nk(m+nk)log(k^2n^2/m)log(knT_{cycle}))$, where $T_{cycle}$ is the circuit timing constraint. The overhead is still acceptable since $k$ is usually a very small number.

### 4.4.4 Solution Transformation

The convex cost-scaling algorithm upon termination gives an optimal flow $x^*$ and the corresponding optimal node potentials $\pi^*$. Both the solutions $x^*$ and $\pi^*$ may be non-integer. Since all the edge costs are integers, there always exist integral optimal vertex potentials $\pi$. To determine them, we construct $G'(x^*)$ and determine the shortest path distance $sd(i)$ from vertex 0 to every other vertex $i \in V'$. Since all edge costs in $G'(x^*)$ are integers, each $sd(i)$ is also an integer. Then $\pi(i) = -sd(i)$ for each $i \in V'$ gives an integral optimal set of vertex potentials for problem (5). According to [2], an optimal solution to problem (2) can be obtained by first assigning $t_i = \pi_i$, then set $d_{ij} = t_j - t_i$ for each $e(i,j) \in E_1$. Note that we do not need to assign the value of $d_{ij}$ for each $e(i,j) \in E_2$, since the power consumption on a wire is zero, and the timing constraints will be automatically satisfied if there exists a feasible solution to the original voltage assignment problem (problem (1)). If no

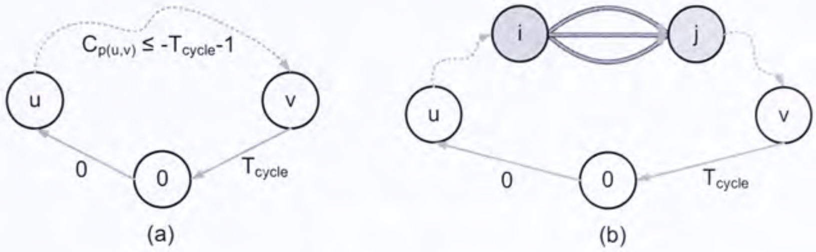


Figure 4.6: Paths in the proof of Lemma 3

feasible solutions of the voltage assignment problem exist, the total flow passing through vertex 0 will be larger than $D$, as stated in the following lemma.

**Lemma 3** *Given an instance of the voltage assignment problem, there exists a feasible solution to this instance if and only if the flow value through vertex* 0 *in the min-cost network flow problem in* $G'$ *as constructed above is such that* $f_0 \leq D$, *where* $D = M - 1 = \sum_{e(i,j)\in E_1} \bar{P}_{ij}(l_{ij}) - \sum_{e(i,j)\in E_1} \bar{P}_{ij}(u_{ij})$.

**Proof:** *(only if part)* Suppose a feasible solution exists for the voltage assignment problem. Let $C$ be the total cost of the flow network. Initially, $C = 0$ when the flow on each edge is zero, and this is automatically the minimum cost flow if there is no negative cycle. Note that every negative cycle must pass vertex 0, since the original netlist is a DAG. Now suppose, without loss of generality, the edge $e(0,u)$, the path from vertex $u$ to vertex $v$ (denoted by $p(u,v)$) and the edge $e(v,0)$ form a negative cycle, as shown if Fig. 4.6$(a)$. Recall that the edge cost are all integers, thus the cost through $p(u,v)$ (denoted by $C_{p(u,v)}$) must be integral, i.e., $C_{p(u,v)} \leq -T_{cycle} - 1$, otherwise the above mentioned negative cycle cannot be formed. This yields that one unit flow through vertex 0 will decrease the total cost $C$ by at least one, and hence $C \leq -f_0$, where $f_0$ denotes the flow value through vertex 0. Note that the optimal objective value of the Lagrange dual problem (problem (4)) $L(x^*) = (-C) + \sum_{e(i,j)\in E_1} \bar{P}_{ij}(u_{ij})$. If $f_0 > D$, then $C < -D$, thus $L(x^*) > D + \sum_{e(i,j)\in E_1} \bar{P}_{ij}(u_{ij}) = \sum_{e(i,j)\in E_1} \bar{P}_{ij}(l_{ij})$. According to theorem 1, $L(x^*)$ equals the optimal objective value of problem (2) (the dual network flow problem), so the objective value of problem (2) is also greater than $\sum_{e(i,j)\in E_1} \bar{P}_{ij}(l_{ij})$, which corresponds to the highest possible total power consumption of all the modules. This is impossible and we can thus conclude that no feasible solutions exist for the original voltage assignment problem exists, contradictory to our original assumption.

*(if part)* Suppose the flow through vertex 0 is less than or equal to $D$, i.e., $f_0 \le D$. Again, we prove by contradiction. If there is no feasible solution to the original voltage assignment problem, it means that the timing constraint $T_{cycle}$ will be violated by some critical path $p(u,v)$ even if each module on $p(u,v)$ is assigned to work at its highest legal voltage level. Given this, there must be at least one module with its input vertex as $i$ and output vertex as $j$ in the expanded network $G'$, such that all the edges between vertex $i$ and vertex $j$ are saturated, since a negative cycle will be formed with vertex 0 otherwise (as shown in Fig. 4.6($b$), all the red edges are saturated). Note that the sum of the capacities of these thickened red edges is $M = D + 1$ and the flow through these edges must go through vertex 0. Therefore, we can conclude that $f_0 > D$, contradictory to our original assumption. $\square$

If the flow through vertex 0 is not more than $D$, we can construct a feasible solution for the original problem (problem (1)) with discretization constraint ($1e$) from this optimal solution to problem (2), by taking the largest possible delay choice $d_{ij}^q$ smaller than or equal to $d_{ij}$ for each edge $e(i,j) \in E_1$. It can be easily shown that this solution is feasible to our original problem, and we can see from the experimental results that this transformation method can approximate the optimal power consumption well and gives large power savings.

## 4.5 Simulated Annealing

A candidate floorplan will be generated by a FAST-SP [17] floorplanner in each iteration of the annealing process, after which the wire delays are estimated and the voltage levels of the modules are assigned by solving the dual network flow problem. The quality of the current solution will then be evaluated by an objective function. When the annealing process terminates, the best found solution will be returned.

### 4.5.1 Moves

There are three kinds of moves to perturb the sequence pair representation of a candidate floorplan solution in the annealing process. This set of moves has been proven to be complete to change any arbitrary solution to any other arbitrary solution.

- *Change aspect ratio* - The aspect ratio of an arbitrary block $m_i$ is changed.
- *Swap two blocks in the $\alpha$ sequence* - Two randomly chosen blocks are swapped in the $\alpha$ sequence.
- *Swap two blocks in both sequences* - Two randomly chosen blocks are swapped in both the $\alpha$ and $\beta$ sequences.

### 4.5.2 Speeding up heuristic

The cost-scaling algorithm is capable of re-computing a minimum cost flow upon incremental changes of the edge costs with relatively less effort, in comparison with solving the whole problem once again. This property can potentially speedup the whole engine when we embed the cost-scaling solver into the annealing process. In practice, only a small portion of nets will get their lengths changed after a random move. This results in a high chance that the minimum cost flow found previously is also optimal (or close to optimal) after the random move. Therefore, in our implementation, we will first check whether the previous solution is still optimal in the new iteration. If this is true, the previous solution will be directly returned; otherwise, refinement will be conducted based on the previous solution. This heuristic has proven to be very helpful in improving the efficiency of our approach, according to the experiments.

### 4.5.3 Cost Function

We use the following cost function to evaluate a candidate floorplan.

$$\psi = A + \lambda_w W + \lambda_{pn}\Phi + \lambda_p P$$

In this function, $A$ is the area of the floorplan; $W$ is the total wire length estimated by the half perimeter bounding box method; $\Phi$ represents the power network resource; $P$ denotes the total power consumption, and particularly, $P$ is set to a large number to give a heavy penalty when the resultant floorplan violates timing (when the amount of flow passing through vertex 0 exceeds $D$). Note that there exists a possibility of accepting a move that results in a floorplan violating timing in our searching process, and this strategy allows us to explore the whole solution space effectively. In addition, it is reasonable to estimate the total power cost on interconnects as a function of the total wire length, and this part of power cost can be taken into consideration by adding a corresponding term into the cost function. In our implementation, we assume that the power cost on interconnects are zero, but this can be easily taken care of according to our discussion above. The parameters $\lambda_w$, $\lambda_{pn}$, and $\lambda_p$ can be used to adjust the relative weighting between the contributing factors, and they are determined in a random walk with 1000 random moves before the annealing process starts, in such a way that the four factors, area, wire length, power network resource and power cost, are contributing similarly to the cost function.

### 4.5.4 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, $n/2$ random moves are performed, where $n$ is the number of modules in the test case. The annealing process stops when the temperature falls below $10^{-5}$.

## 4.6 Experimental Results

The previous work [11] is the most updated one in handling the floorplanning problem under timing constraint in the context of MSV. In order to compare with [11], we performed a set of experiments on the test cases provided by the authors of [11]. There are totally six test cases, with the number of blocks ranging from 10 to 300. These test cases are based on the GSRC benchmarks, with power and delay specifications added for each block. In these test cases, every block has two legal working voltage levels, each of which is associated with both a power consumption value and a delay value.

Note that the approach in [11] performed level shifter insertion into the netlist, after the voltage assignment step. Whenever there is a net through which signal is sent from a module at low voltage level to a module at high voltage level, a level shifter will be inserted into this net. Each level shifter is associated with fixed delay and power values, and will affect the timing and power consumption of the whole circuit. In order to consider level shifters so as to have a fair comparison with [11], we extended our floorplanner with a conservative approach to take level shifters into consideration. When the DAG representation of the input netlist is given, we will first compute the longest path $L$ (in terms of the number of edges on a path) from primary input to primary output in the DAG. It is trivial that at most $L$ level shifters will be inserted on any path of the DAG after voltage assignment.

This indicates that after level shifter insertion, the delay of the whole circuit will be increased by $L \cdot d_{ls}$ units in the worst case where $d_{ls}$ denotes the delay of a level shifter. Therefore, in order to guarantee that the timing constraint can still be satisfied after level shifter insertion, we simply put $T_{cycle} - L \cdot d_{ls}$ as the timing constraint of the circuit while solving the voltage assignment problem.

The comparisons are displayed in Table 4.1. The column *Max Power (MaxP)* denotes the total power consumption when each module of the circuit works at its highest legal voltage level, and the column *Power Cost with LS (P)* shows the sum of the actual power consumption of each module plus the power consumption of all the level shifters inserted. The power saving is calculated by $(MaxP - P)/MaxP$. We can see that more power saving can be achieved in much less running time by using our approach. An average of 24.61% (v.s. 6.68% by [11]) power saving can be achieved in an average running time of 273$s$ (v.s. 1911$s$ by [11]) with our approach. Besides, we further demonstrated the effectiveness of our approach by performing another set of experiments in which the same set of circuits is used but the number of legal working voltage levels for each module is augmented to either three or four. For this second set of experiments, the resultant floorplans of all the six circuits are displayed in Fig 4.7, and the detailed results are listed in Table 4.2, from which we can see that an average of 28.62% power saving can be achieved. Our algorithm was implemented in the C programming language and all the experiments were performed on a Linux machine with a 3.20 GHz CPU and 2 GB RAM.

## 4.7 Summary

In this chapter, we proposed a framework for power optimization under timing constraints in floorplanning. We show that the voltage assignment problem under timing constraint can be formulated as a dual network flow problem that can

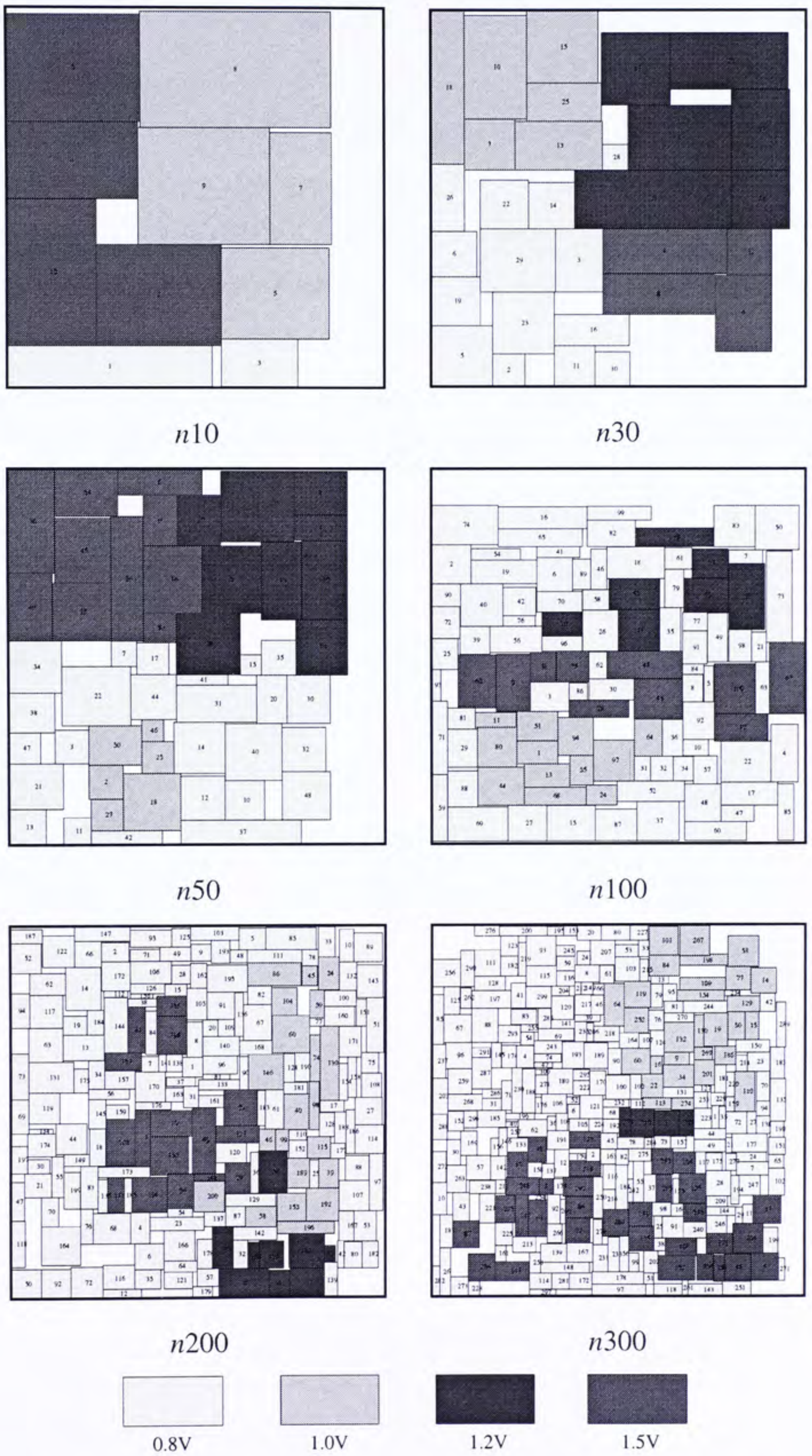


Figure 4.7: Resultant floorplanns with each module having multiple legal working voltage levels

Table 4.1: Comparisons of our framework with previous work [11]

| Data Set | Max Power (MaxP) | Power Cost with LS (P) | | Power Saving (%) | | Power Network Resource | | LS Number | | Dead Space (%) | | Run Time (s) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | ours | [11] | ours | [11] | ours | [11] | ours | [11] | ours | [11] | ours | [11] |
| n10 | 216841 | 189942 | 216841 | 12.40 | 0 | 1516 | 965 | 4 | 0 | 2.32 | 4.87 | 1.09 | 6.001 |
| n30 | 205650 | 153526 | 190717 | 25.35 | 7.26 | 1738 | 1369 | 26 | 57 | 7.74 | 9.03 | 9.31 | 115.069 |
| n50 | 195140 | 152684 | 172884 | 21.76 | 11.40 | 1589 | 1514 | 34 | 119 | 8.96 | 21.10 | 25.34 | 569.360 |
| n100 | 180022 | 120450 | 179876 | 33.09 | 0.10 | 1612 | 1671 | 77 | 92 | 7.53 | 34.07 | 121.61 | 1768 |
| n200 | 177633 | 135250 | 174818 | 23.86 | 1.58 | 1619 | 2040 | 129 | 399 | 7.19 | 46.52 | 454.60 | 4212 |
| n300 | 273499 | 188113 | 219492 | 31.22 | 19.75 | 1844 | 2147 | 151 | 452 | 18.11 | 44.10 | 1026.68 | 4800 |
| Average | - | 156661 | 192438 | 24.61 | 6.68 | 1653 | 1617.7 | 70 | 186 | 8.64 | 26.61 | 273.10 | 1911.74 |
| Difference | - | 1 | 1.23 | 1 | 0.27 | 1 | 0.98 | 1 | 2.66 | 1 | 3.08 | 1 | 7 |

Table 4.2: Results with more legal working voltage levels

| Data Set | Max Power (MaxP) | Power Cost with LS | Power Saving (%) | Power Network Resource | LS Number | Dead Space (%) | Run Time (s) |
|---|---|---|---|---|---|---|---|
| n10 | 216841 | 167012 | 22.98 | 1818 | 9 | 4.08 | 1.46 |
| n30 | 205650 | 142717 | 30.60 | 2012 | 37 | 16.80 | 13.79 |
| n50 | 195140 | 143562 | 26.43 | 1945 | 43 | 10.66 | 43.44 |
| n100 | 180022 | 124428 | 30.88 | 2377 | 108 | 8.98 | 195.85 |
| n200 | 177633 | 133360 | 24.93 | 2382 | 176 | 11.16 | 780.74 |
| n300 | 273499 | 175281 | 35.91 | 2413 | 188 | 12.83 | 1812.37 |
| Average | - | 147726 | 28.62 | 2157 | 94 | 10.75 | 474.61 |

be transformed into a primal minimum cost network flow problem using the Lagrangian relaxation technique and can subsequently be solved optimally by a cost-scaling algorithm in polynomial time when the delay choices of each module are continuous in the real or integer domain. We can make use of this approach to obtain a feasible voltage assignment solution in the general cases with power consumption approximating the minimum one. Globally, a simulated annealing based FAST-SP floorplanner is employed to pack the modules providing physical information to the voltage assignment engine. In each iteration, the cost-scaling solver is invoked to assign voltage levels, after updating the wire delays. Experimental results show that our approach is very effective and efficient in reducing power consumption and power network resource. In comparison with previous work [11], our framework achieves an additional 18% power saving on average, in much less running time.

---

□ **End of chapter.**

# Chapter 5

# Conclusion

As technology scales for increased circuit density and performance, the need to reduce power consumption increases significantly as designers strive to utilize the advancing silicon capabilities. The consumer product market further drives the need to minimize chip power consumption. As power consumption is proportional to the square of supply voltage, reducing supply voltage can significantly reduce power consumption. Multiple Supply Voltage (MSV) has previously been introduced to provide higher flexibility in controlling power and performance trade-off. In this thesis, we are interested in handling the simultaneous voltage island partitioning, voltage level assignment and floorplanning problem under various constraints in MSV designs, so as to optimize both the power consumption and physical layout of the circuitry.

We first propose a simulated annealing based approach to handle the floorplanning problem with simultaneous island partitioning and voltage assignment, where the three factors area, wire length and power consumption of the resultant floorplan are concurrently taken into consideration. In each step of the annealing process, a candidate floorplan solution is generated on which optimal island partitioning and voltage assignment can be performed to compute the smallest possible power consumption for that candidate floorplan solution. This is done by dynamic programming with an efficient cost table update technique. The experiment results demonstrate the effectiveness of our approach.

One important requirement of MSV design is that timing constraints of the circuit must be satisfied after voltage assignment of the cells. Basically, high voltage level will be assigned to critical cells and low voltage level is assigned to non-critical cells, so that power can be saved without violating the timing constraints. In order to take timing constraints into account, we propose another floorplanning framework for MSV designs. We show that the voltage assignment problem under timing constraint can be formulated as a dual network flow problem that can be transformed into a primal minimum cost network flow problem using the Lagrangian relaxation technique and can subsequently be solved optimally by a cost-scaling algorithm in polynomial time when the delay choices of each module are continuous in the real or integer domain. We can make use of this approach to obtain a feasible voltage assignment solution in the general cases (when the delay choices are discrete) with power consumption approximating the minimum one. Globally, a simulated annealing based FAST-SP floorplanner is employed to pack the modules providing physical information to the voltage assignment engine. In each iteration, the cost-scaling solver is invoked to assign voltage levels, after updating the wire delays. Experimental results show that our approach is very effective and efficient in reducing power consumption and power network resource.

---

□ **End of chapter.**

# Bibliography

[1] A. O. Adan and K. Higashi. Off-state leakage current mechanisms in bulksi and soi mosfets and their impact on cmos ulsis standby current. *IEEE Transactions on Electron Devices*, 48(9):2050–2057, September 2001.

[2] R. K. Ahuja, D. S. Hochbaum, and J. B. Orlin. Solving the convex cost integer dual network flow problem. *Management Science*, 49(7):950–964, July 2003.

[3] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Application*. Prentice Hall, 1993.

[4] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, and S.-M. Wu. A non-redundant repreentation for general non-slicing floorplan. In *Proceedings of the 37th ACM/IEEE Design Automation Conference*, pages 458–463, 2000.

[5] R. L. S. Ching and E. F. Y. Young. Post-placement voltage island generation. In *Proceedings of the International Conference on Computer-Aided Design*, 2006.

[6] A. V. Goldberg. An efficient implementation of a scaling minimum-cost flow algorithm. *Journal of Algorithms*, 22:1–29, 1997.

[7] A. V. Goldberg and R. E. Tarjan. Solving minimum cost flow problem by successive approximation. In *Proceedings of the 19th ACM Symposium on the Theory of Computing*, pages 7–18, 1987.

[8] J. Hu, Y. Shin, N. Dhanwada, and R. Marculescu. Architecting voltage islands in core-based system-on-a-chip designs. In *Proceedings of International Symposium on Low Power Electronics and Design*, pages 180–185, 2004.

[9] W.-L. Hung, G. Link, Y. Xie, N. Vijaykrishnan, N. Dhanwada, and J. Conner. Temperature-aware voltage islands architecting in system-on-chip design. In *Proceedings of the Computer Design*, 2004.

[10] 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 *Proceedings of International Conference on Computer-Aided Design*, pages 195–202, 2002.

[11] W.-P. Lee, H.-Y. Liu, and Y.-W. Chang. Voltage island aware floorplanning for power and timing optimization. In *Proceedings of International Conference on Computer-Aided Design*, 2006.

[12] W.-P. Lee, H.-Y. Liu, and Y.-W. Chang. An ilp algorithm for post-floorplanning voltage-island generation considering power-network planning. In *Proceedings of International Conference on Computer-Aided Design*, 2007.

[13] B. Liu, Y. Cai, Q. Zhou, and X. Hong. Power driven placement with layout aware supply voltage assignment for voltage island generation in dual-vdd designs. In *Proceedings of Asia and South Pacific Design Automation Conference*, 2006.

[14] Q. Ma and E. F. Y. Young. Voltage island-driven floorplanning. In *Proceedings of International Conference on Computer-Aided Design*, pages 644–648, 2007.

[15] Q. Ma and E. F. Y. Young. Network flow based power optimization under timing constraints in msv-driven floorplanning. To appear in *Proceedings of International Conference on Computer-Aided Design*, 2008.

[16] W. K. Mak and J. W. Chen. Voltage island generation under performance requirement for soc designs. In *Proceedings of the Asian South Pacific Design Automation Conference*, 2007.

[17] X. Tang and D. F. Wong. Fast-sp: a fast algorithm for block placement based on sequence pair. In *Proceedings of Asia and South Pacific Design Automation Conference*, 2001.

[18] D. F. Wong and C. L. Liu. A new algorithm for floorplan design. In *Proceedings of the 23rd ACM/IEEE Design Automation Conference*, pages 101–107, 1986.

[19] H. Wu, I.-M. Liu, D. F. Wong, and Y. Wang. Post-placement voltage island generation under performance requirement. In *Proceedings of the International Conference on Computer-Aided Design*, 2005.

[20] H. Wu, D. F. Wong, and I.-M. Liu. Timing-constrained and voltage-island-aware voltage assignment. In *Proceedings of Design Automation Conference*, pages 429–432, 2006.

[21] S. Yang, W. Wolf, N. Vijaykrishnan, and Y. Xie. Reliability-aware soc voltage islands partition and floorplan. In *Proceedings of the Emerging VLSI Technologies and Architectures*, 2006.