國 立 交 通 大 學

電子工程學系 電子研究所碩士班

碩 士 論 文

考慮存在電壓島的緩衝器繞線樹架構的一個有效率的演算法

An Efficient Algorithm for Voltage Island Aware Buffered Routing Tree Construction

研 究 生：曾柏欽

指導教授：陳宏明 博士

中 華 民 國 九 十 五 年 七 月

存在電壓島的緩衝器繞線樹架構的一個有效率的演算法

An Efficient Algorithm for Voltage Island Aware Buffered Routing Tree Construction

研究生：曾柏欽　　Student: Bruce Tseng

指導教授：陳宏明 博士　　Advisor: Prof. Hung-Ming Chen

國 立 交 通 大 學

電子工程學系　　電子研究所碩士班

碩士論文

A Thesis
Submitted to Department of Electronics Engineering & Institute of Electronics
College of Electrical Engineering and Computer Science
National Chiao Tung University
in Partial Fulfillment of Requirements
for the Degree of
Master of Science
in
Electronics Engineering
July 2006
Hsinchu, Taiwan, Republic of China

中華民國九十五年七月

考慮存在電壓島的緩衝器繞線樹架構的一個有效率的演算法

研究生：曾柏欽　　指導教授：陳宏明 博士

國立交通大學

電子工程學系 電子研究所碩士班

摘要

由於低功率的方法論在超大型積體電路(VLSI)和系統單晶片(SoC)的需求，電壓島(voltage island)越來越吸引設計者的注意。然而，相對應在電子自動化設計(EDA)工具方面，考慮到電壓島的繞線樹建構仍然非常稀少。[7] 是第一個深入研究應用兩種不同 Vdd 電壓的緩衝器在繞線樹的建構上，並且藉由限制緩衝器的擺放順序而忽略電壓轉換器(level converter)需求的考量。然而，因為有這些限制，所以這個方法並不能應用到具有電壓島的設計上面。

本篇論文提出一個演算法以解決具有電壓島的低功率設計上緩衝器以及電壓轉換器放置的問題。既然[7](使用[9]中的技巧)無法應用在電壓島的設計上，我們改進[9]中 RMP 的方法以適用在此種設計上。我們更進一步的發展我們的方法來做比較。藉由一些貪婪探索(greedy heuristic)的技巧，我們的方法不僅非常有效率，而且能維持解的品質。

實驗結果顯示相較於改良的 RMP 方法，我們的方法是非常有效率，甚至於會有較低的功率和延遲(delay)。再者，當汲點(sink)的個數增加，改良的 RMP 方法無法在一合理的時間內找到一組解答,我們的方法可以更有效率的找到一組適當的解。

An Efficient Algorithm for Voltage Island Aware Buffered Routing Tree Construction

Student: Bruce Tseng Advisor: Prof. Hung-Ming Chen

Department of Electronics Engineering
& Institute of Electronics
National Chiao Tung University

Abstract

Due to the need of low power methodology in VLSI and SoC designs, voltage island architecture is attracting attentions in design community. However, the corresponding EDA tools development regarding routing tree construction is still very few. [7] is the first in-depth study on applying dual Vdd buffers in routing tree construction, with the restriction on the ordering of buffers and the lack of level converter consideration. However this approach cannot be applied on a design with voltage islands due to these restrictions.

This paper presents an algorithm to solve the buffer insertion and level converter assignment problem in the presence of voltage island in a low-power design. Since [7] (use techniques in [9]) cannot be performed on voltage island designs, we have modified RMP approach in [9] to perform on those designs. We then develop our approach for comparison. With some greedy heuristics and prune techniques, our approach is very efficient and it still keeps the quality of solutions.

The experimental results show that we can obtain massive speedup over modified RMP approach, and even with lower power and delay. Furthermore, as number of sink increases, modified RMP cannot find solutions within a reasonable CPU time, while our approach can efficiently find feasible solutions.

誌謝

首先要特別感謝的人，是我的指導教授陳宏明老師，帶領我進入一個新的領域 EDA，感謝老師的指導與包容，讓我可以完成本篇論文並且學習到許多 EDA 相關的知識。

此外，要感謝的是 VDA LAB 實驗室所有的成員，兩年來相互的勉勵、幫忙，一起歡樂，有壓力時相互扶持，讓我有兩年充實的生活。

家人對我的支持、鼓勵更是我研究路上最大的依靠，對他們的感謝，更是筆墨難以形容。

最後由衷感謝所有幫助過、關懷過我的人，謝謝！

曾柏欽

民國九十五年七月 於新竹

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In CMOS digital circuits, power dissipation mainly consists of dynamic and static components. Dynamic and static power both have direct relationship with supply voltage V_{dd}. One of the techniques to reduce power consumption is voltage island methodology, proposed from IBM [16]. A voltage island is a group of on-chip cores powered by the same voltage source, independently from the chip-level voltage supply. Voltage island architecture can achieve power saving and this technique becomes more and more popular [14]-[19].

Nevertheless, there are very few development in corresponding EDA tools regarding voltage-island-aware routing tree construction. Based on the state-of-the-art buffered tree methodologies, we can put them into two categories. The first type is tree based algorithms [1]-[6]. They first generate routing tree topology, and use van Ginneken's approach [1] to insert buffers on this tree. The second type is graph based algorithms [7]-[13]. They deal with both routing and buffer insertion at the same time. Consider the circuit of a voltage island design, such as shown in Figure 1.1, we must use some level converters/shifters to maintain signal integrity when a low V_{dd} device drives a high V_{dd} device. If we use one of the tree based algorithms to do the buffer insertion and level converter assignment, there will be two concerns. First, the generated routing tree may not have feasible level converter location. As a result, signal integrity will not be able to be kept. Second, if we regard level converter as a standard cell and place it during placement stage, signal integrity can be maintained though, the location of level converter will affect total wirelength, delay and power. On the contrary, if we regard a level converter as a kind of buffer, a graph based algorithm will provide better solutions. But the complexity analysis in [9][10] has shown that graph based algorithms have a serious problem in exponential increase of runtime when

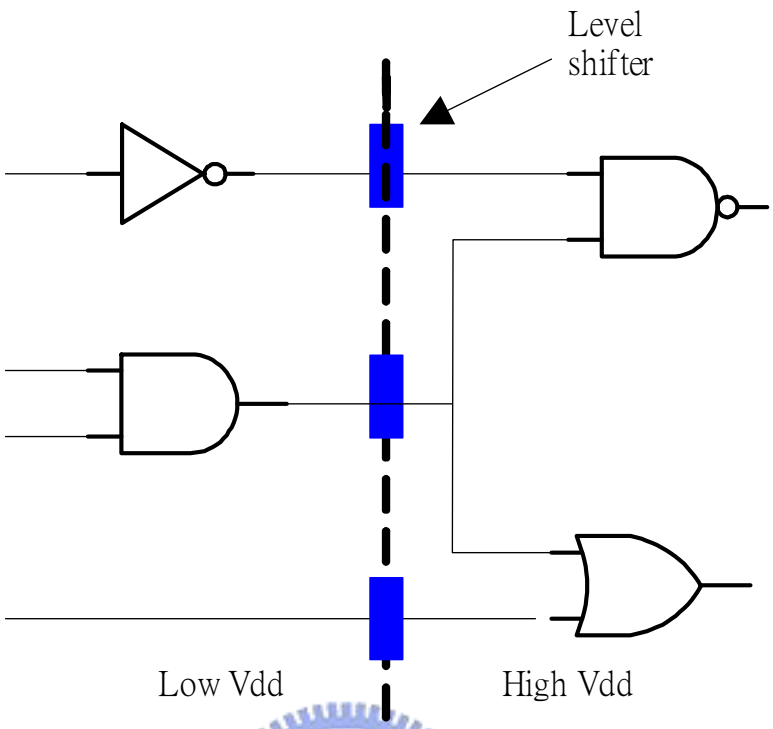

Figure 1.1: For a voltage island design, we always need level converter to maintain signal integrity. Our approach views level converter as a kind of buffer, and buffer insertion operation will place level converter at a better location than conventional tree based algorithm.

the number of sinks increases. The experimental results shown in [7] also report the same observation.

1.1 Previous Works

Some algorithms about buffer insertion problem have been proposed in [1] -[13]. Some of these methods regard the routing tree as an input, and buffer insertion algorithm intends to find a minimum delay for this routing tree. Because the input of the algorithm is a tree, we classify these algorithms as **tree based buffer insertion algorithms**. On the other hand, others simultaneous bulid a routing tree and perform buffer insertion during buliding routing trees. Although these kinds of approaches always lead to less delay than tree based buffer insertion algorithms, most of these algorithms need to spend more time on computing

a better result. Because the algorithms always create a graph for buffer insertion, we classify these algorithm as **graph based buffer insertion algorithms**. Here we introduce some of algorithms to show the differences. Then we describe the DVB algorithm in [7] for dual-Vdd buffer insertion.

1.1.1 Treed base buffer insertion algorithms

Almost all the tree based algorithms are based on the algorithm proposed by van Ginneken[1]. van Ginneken's algorithm deals with a routing tree which has multiple sinks and considers delay minimization only. It traverses a tree with a bottom up approach, and calculates delay with Elmore delay model during traversing a tree. For a tree with n feasible buffer location and buffer library has only one buffer, there should be 2^n kinds of buffer tree. But in van Ginneken's algorithm, a prune technique is adopted such that there will be only n kinds of buffer tree and results in a time complexity of $O(n^2)$.

- For 2 candidates $(D_1, C_1), (D_2, C_2)$, where D states the accumlated delay and C states the accumlated load capacitance. If $D_1 > D_2$ and $C_1 > C_2$, then (D_1, C_1) can be pruned.

After [1] has proposed, some algorithms intends to model the delay more accurately, such as [2]. Just like we demonstrate in Figure 1.2, [2] uses a π-model and changes the load capacitance part as (C_n, R, C_f). Besides, some of the papers, such as [3] [4] [5], use either a balanced tree for storing the candidates or some aggressive pruning methods to make the algorithm even faster. Recently, [4] has improved the time complexity of a buffer insertion algorithm for a 2-pin net from $O(n^2)$ to $O(nlogn)$, and to $O(nlog^2n)$ for multi-pin nets.

Because tree based algorithms regard routing tree as an input, the routing tree could not be changed during buffer insertion stage. And routing algorithms usually intends to find a Minumum Reclinear Steiner Tree (MRST) with minimum total wirelength. Therefore, if the routing tree has less feasible buffer locations, the buffer insertion algorithm can only improve the delay with these locations and will result in a poor delay while comparing with a graph based buffer insertion algorithm. Moreover, for a design with voltage island, if the MRST has none of feasible level converter location, the signal integrity can not be maintained and a large leakage current will be generated.

1.1.2 Graph based buffer insertion algorithms

For a 2-pin net, Lai et. al. proposes an elegant formulation of the maze routing with

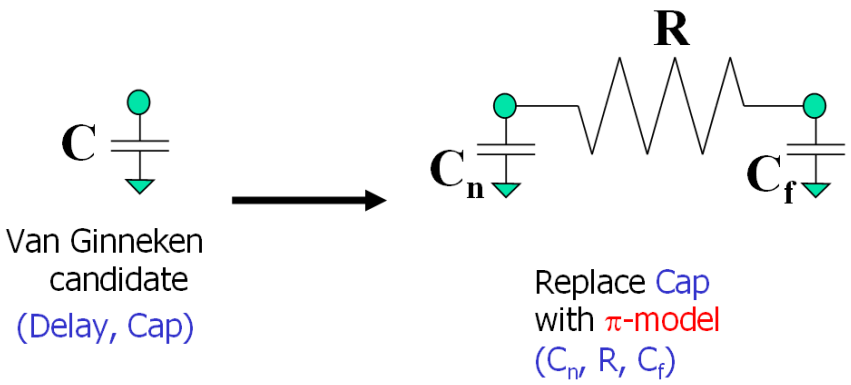

Figure 1.2: [2] uses a π-model instead of Elmore delay model and changes the load capacitance *cap* in Van Ginneken's original candidate as (C_n, R, C_f).

buffer insertion and wire sizing problem as a theoretic shortest path problem in [13]. But it still needs to consume much time on solving both maze routing and buffer insertion problem for a multiple sink net. Such as [11] [12], the maze routing with buffer insertion problem is converted into a graph collection problem. Various kinds of buffer routing tree subsets are pre-computed as a table. And the buffer routing tree is constructed through a dynamic programming approach with combining these subsets. Although the dynamic programming approaches consume less time, their algorithms still need much time on table computation.

J. Cong et. al. proposed a RMP (Recursively Merging and pruning) algorithm in [9]. Different from [11] [12], RMP simultaneously bulids the routing topology with considering the buffer insertion at the same time. RMP first creates a grid graph and solutions with factors of capacitance *cap*, required arrival time RAT, reachable sink set RE and buf for stating whether a buffer which had been placed is filled into each node in the created graph. With defining the formulation of solution propagation, various kind of solution could be generated during solution propagation. And each solution corresponds to a buffer routing tree. Therefore, various kinds of buffer routing tree is generated during computation, and the solutions at the source node states all the possible buffer trees.

1.1.3 DVB algorithm[7]

DVB algorithm which is the first in-depth study on applying dual V_{dd} buffers to buffer insertion. With the restriction of low and high V_{dd} buffer's ordering, their algorithm

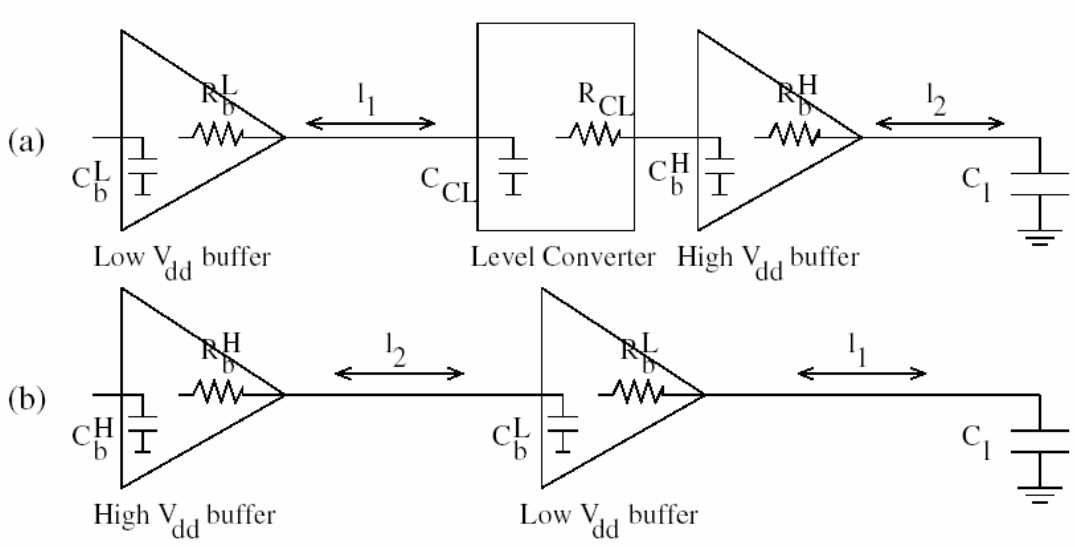

Figure 1.3: Refer from [7]. DVB algorithm states that to have the delay of case (b) larger than case (a), Cl must be larger than 0.5pF or an equivalent 9mm interconnect wire. Therefore, DVB algorithm restricts the buffers ordering as case (b).

could neglect the level converter. Their algorithm is realized with both tree based and graph based method. And the experimental result shows that their algorithm reduces 18~26% power consumption while comparing with using signal Vdd buffers for buffer insertion. For DVB algorithm with graph based method, their algorithm is similiar to RMP algorithm. And a time consuming problem is also occurred in their experiment result. That is their algorithm needs more than 20 minutes to perform both routing and buffer insertion for a net with 6 sink.

For Figure 1.3, which DVB algorithm states that to have the delay of case (b) larger than case (a), they find that C_l must be larger than 0.5pF or an equivalentlly a 9mm interconnect wire. Therefore, they restrict buffer's ordering by only high Vdd buffers could drive a low Vdd buffers, and thus none of level converter is needed in their algorithm. Alghough, this could reduce the complexity of a dual Vdd buffer insertion problem, but the assumption makes the algrotihm not realistical enough. First, for Figure 1.3 (b), if the sink device which C_l states is an high V_{dd} device, there will still need another level converter between the low V_{dd} buffer and C_l, or else a large leakage current occurs at sink device. Under this assumption, we shall need an extra level converter in Figure 1.3(b). Second, if the DVB

algorithm is applied on a design with dual V_{dd} voltage, and their algorithm inserts both low and high Vdd buffers anywhere in this design. It will make P/G line routing becomes a tough problem. Besides this, most dual Vdd design uses a voltage island floorplan, and a level converter is widely used. Without considering the level converter, it makes the DVB algorithm becomes not realistical.

1.2 Our Contributions

To deal with this problem, we adopt prune technique and a greedy heuristic in the proposed algorithm FVABT (Fast Voltage island Aware Buffered Tree construction). This algorithm uses a graph based algorithm to deal with both buffer insertion and level converter assignment problem, and also improves the time-consuming problem in graph based algorithm.To the best of our knowledge, this paper is the first work on the buffered routing tree construction in the presense of voltage islands. The contributions presented in this paper are as follows:

- Since current dual-Vdd buffer insertion approach cannot be performed on the designs with voltage islands, we have modified the approach in [9] (RMP) so that it can be used with designs that contain voltage islands.
- Our approach has obtained massive speedup over modified RMP, and even produced lower power buffered tree. Furthermore, as number of sink increases, our approach can find feasible solutions effectively and efficiently.

1.3 Organization

The rest of this paper is organized as follows. Section II introduces the power and delay estimation model used and problem formulation. Section III shows modified RMP algorithm and Section IV introduces our greedy heuristic FVABT to apply on modified RMP algorithm for runtime reduction. Section V shows our experimental results and Section VI concludes the paper.

Chapter 2

Preliminaries

We introduce the models that we use in this thesis, and formulate our problem as follows.

2.1 Power and Delay Estimation Models

We adopt the power and delay model introduced in [7]. The delay model is a distributed Elmore delay model. For the circuit shown in Figure 2.1, the delay of a wire D_w and delay of a buffer D_{buf} are defined as:

$D_w(L) = (\frac{1}{2} \cdot c_w \cdot L + C_{load}) \cdot r_w \cdot L$

$D_{buf} = D_b + R_b \cdot C_{load}$

where c_w is unit length capacitance, r_w is unit length wire resistance, L is the downstream wire length, V_{dd} is the voltage level of the device or the signal on the wire, D_b is the instrinsic delay of a buffer, R_b is the output resistance of a buffer, C_{load} is the downstream load capacitance.The interconnect power consumption P_w measured by energy per switch is defined as:

$P_w(L) = \frac{1}{2} \cdot c_w \cdot L \cdot V_{dd}^2$

We give an example as follows to further illustrate our estimation model.

Example 1 *For the circuit shown in Figure 2.2, a buffer drives two wire segments with* L_1*=100um,* L_2*=200um, and the load capacitance on these two segments are* C_1*=30fF,* C_2*=40fF, with the paramenters* r_w*=0.186Ω/um,* c_w*=0.0519fF/um,* D_b*=72ps and* R_b*=4.7kΩ. The delays in the circuit are computed as follows:*

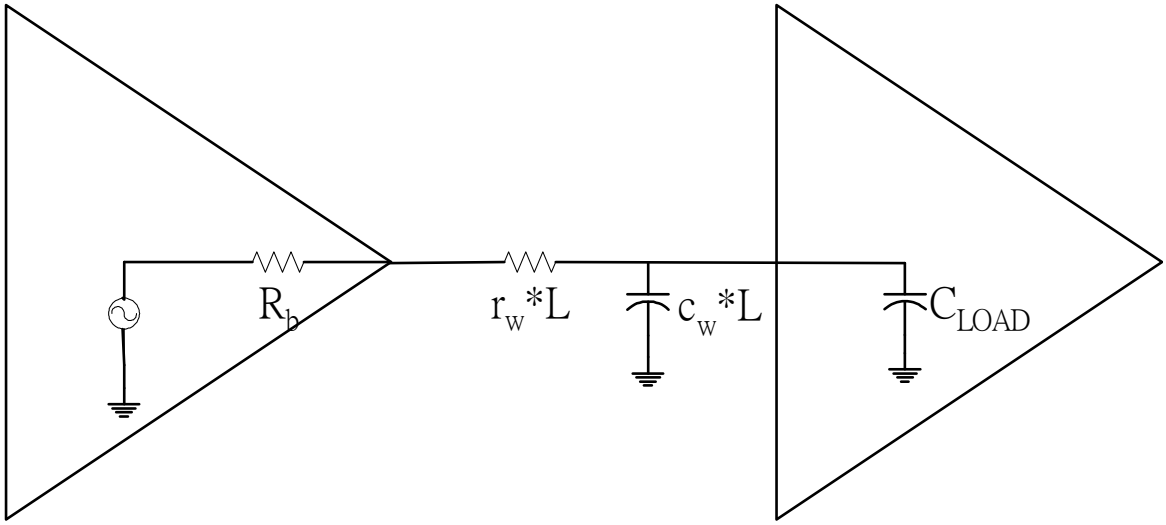

Figure 2.1: Power and delay model while a buffer drives a wire with length L and a capacitance load with C_{LOAD}, where c_w is unit length capacitance, r_w is unit length wire resistance, and R_b is the output resistance for buffers.

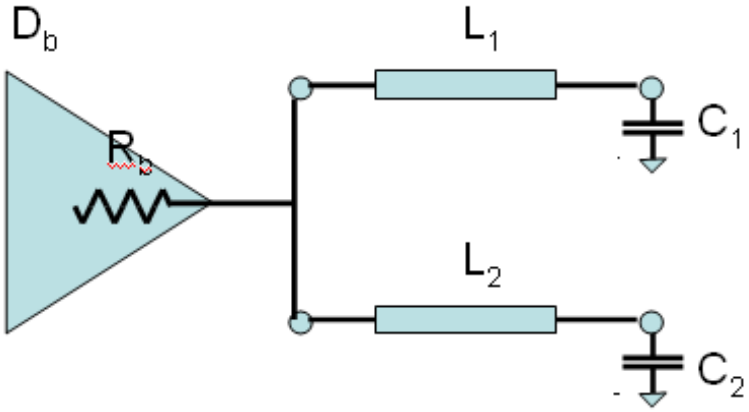

Figure 2.2: An example of delay computation: a buffer drives two wire segments L_1, L_2 with capacitance load C_1 and C_2.

The delay due to a buffer:

$$\begin{aligned} D_{buf} &= D_b + R_b \cdot (C_1 + L_1 \cdot c_w + C_2 + L_2 \cdot c_w) \\ &= 72 + 4.7 \cdot (30 + 100 \cdot 0.0519 + 40 + 200 \cdot 0.0519) \\ &= 474.179ps \end{aligned}$$

The delay due to the wires:

$$\begin{aligned} D_w &= \max\{D_w(L_1), D_w(L_2)\} \\ &= \max\{(\tfrac{1}{2} \cdot c_w \cdot L_1 + C_1) \cdot r_w \cdot L_1, (\tfrac{1}{2} \cdot c_w \cdot L_2 + C_2) \cdot r_w \cdot L_2\} \\ &= \max\{606.267, 1681.068\} \\ &= 1681.068ps \end{aligned}$$

The total delay:

$$\begin{aligned} D_{tot} &= D_{buf} + D_w \\ &= 474.179 + 1681.068 \\ &= 2155.247ps \end{aligned}$$

2.2 Problem Formulation

Our problem is based on the following assumptions. There are two types of buffers (buffers with high and low V_{dd}) and one type of level converters with various kinds of size in the buffer library, level converters must be drived by a high V_{dd} supply voltage so as to raise voltage level from low to high. Hence both level converters and high V_{dd} buffers can only be placed within a high V_{dd} region, low V_{dd} buffers can only be placed within low V_{dd} region. Note that a voltage island might be turned off while the chip is operating at power saving mode. The problem of buffer insertion and level converter assignment on a dual V_{dd} voltage island design can be specified as follows:

Problem 2 *Given a design with voltage island(s), a net with a source node, multiple sink node with RAT (required arrival time) at each sink, feasible buffer location, buffer library and wire obstacles (such as hard IPs), we want to construct buffered routing tree with buffer insertion and level converter assignment under the following constraints:*

- *RAT at each sink should be met.*
- *Signal integrity of nets can be maintained.*

- *The design works during power saving mode, such that the buffered routing tree has minimized power consumption.*

Chapter 3

Modified RMP algorithm

Since there are no approaches which can be used in routing tree construction for designs with voltage islands, we modify the original RMP in [9], which is used in [7], to fit a voltage island design, and try to speed up the algorithm as well. In order to fit a voltage island design, we use an indicator to show the possible voltage level of a signal ($signalV$) and keep on maintaining this indicator in this methodology.

Instead of fetching the solution with the maximal RAT among all the solutions, modified RMP algorithm classifies solutions with their reachable sinks, and store these solutions to a container called *wave*. A wave is popped and solutions of nodes in this wave are propagated to its neighboring nodes. The new generated solutions are also categorized with their reachable sinks, and these new solutions are stored in a new wave. During each iteration, if the new wave does not contain the source node, we store this new wave. If it contains the source node, this new wave is ignored. There are four steps in the modified RMP algorithm, and the details of these modifications are shown in the following subsections.

3.1 Construct Grid Graph

First, we create bounding box with the minimum rectangle covering all the source and sink nodes. Then we keep on enlarging the bounding box such that there exists none of wire obstacles being cut. During box enlarging, if source node of the net is out of the voltage island, we also regard voltage island as an obstacle. We then partition the bounding box into a grid graph by using the vertical and horizontal line intersect at source and sink nodes, buffer locations, and four corners of the wire obstacle. An example of the grid graph

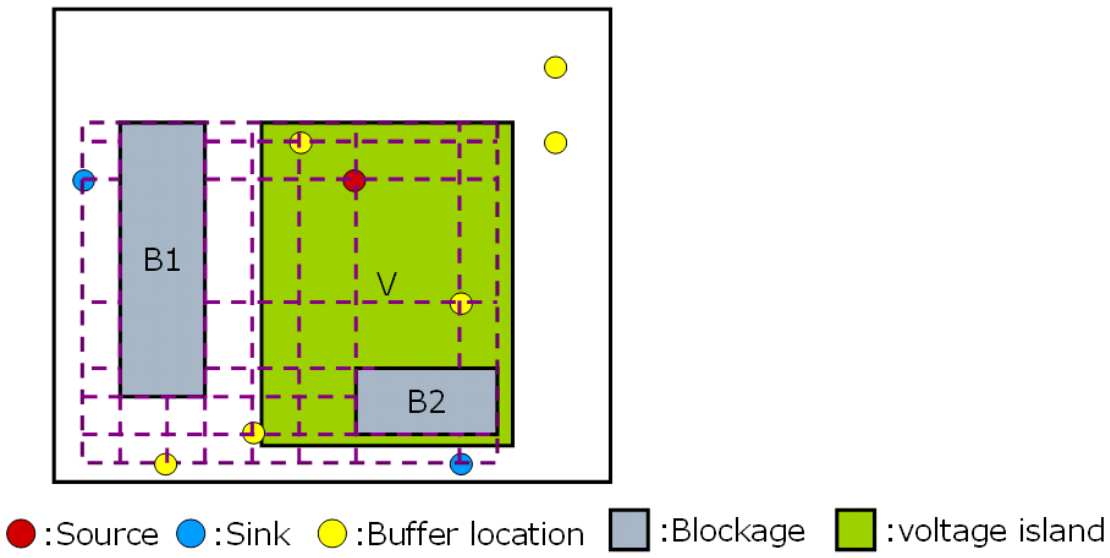

Figure 3.1: Grid graph construction. We define a bounding box that covers all the source, sink nodes and none of obstacles is cut by the bounding box. Then we partition the bounding box into a grid graph.

is shown in Figure 3.1.

3.2 Fill Initial Solutions

During this step, we fill initial solutions for each cross point in the grid graph. There are ten items (*cap*, *rat*, *pow*, *rn*, *rs*, *B*, *signalV*, *Cbl*, *bend*, *totLength*) in each solution, listed as follows:

cap: The accumulated capacitive load.

rat : Required arrival time.

pow : The accumulated power consumption.

rn (reachable node): The nodes that passed through to prevent from following the same path.

rs (reachable sink): The sink that the solution can reach.

B: The location where a buffer is placed and the size of the buffer placed at this location.

$signalV$ (signal voltage): Possible signal voltage level. If it is high, the solution can only be drived by a high V_{dd} device. If it is low, both high and low V_{dd} device can drive this solution.

Cbl: When two solutions with different sink are merged at buffer location, Cbl states the extra load capacitance that the buffer needs to drive.

$bend$: The accumlated number of bendings in current solution. We use this to prune solutions.

$totLength$: The accumlated wire length.

According to the type of the node, one or some of the initial solutions are filled for each node. The initial solution at sink node is the starting point for the propagation, while the solutions at the other nodes are formed through the propagation toward the source node. The rules for filling the initial solution are described below. We assume that there are n_H high V_{dd} buffers, n_L low V_{dd} buffers, m voltage level converters in the buffer library.

- If node i is a sink node, fill one solution:

  $(cap_i, rat_i, 0, \{i\}, \{i\}, \phi, signalV_i, 0, 0, 0)$

  It states a buffered routing sub-tree with zero wire length and it contains only the sink node i. The cap and rat state the load capacitance and requirement arrival time of this sink i separately.

- If node i is a source node, fill one solution:

  $(0, \infty, 0, \{i\}, \phi, b_s, Low, 0, 0, 0)$ where b_s input capacitance=0, delay=0, output resistance=driving resistance of source node.

  It states a buffer with the driving resistamce of the device at source node is placed.

- If node i is not a buffer location, fill one solution:

  $(0, \infty, 0, \{i\}, \phi, \phi, Low, 0, 0, 0)$

  It states none of buffer is placed here.

- If node i is a buffer lcoation and within the voltage island (low V_{dd} region), fill $1 + n_L$ solutions:

  $(0, \infty, 0, \{i\}, \phi, \phi, Low, 0, 0, 0)$;

  $(0, \infty, 0, \{i\}, \phi, b_{1L}, Low, 0, 0, 0)$˜ $(0, \infty, 0, \{i\}, \phi, b_{nL}, Low, 0, 0, 0)$

  It states the case that none of buffer is placed or the cases for various kinds of low V_{dd} buffer placed.

- If node i is a buffer location and outside the voltage island, fill $1 + n_H + m$ solutions:

  $(0, \infty, 0, \{i\}, \phi, \phi, Low, 0, 0, 0)$;

  $(0, \infty, 0, \{i\}, \phi, b_{1H}, Low, 0, 0, 0)$˜ $(0, \infty, 0, \{i\}, \phi, b_{nH}, Low, 0, 0, 0)$;

  $(0, \infty, 0, \{i\}, \phi, b_1, Low, 0, 0, 0)$˜ $(0, \infty, 0, \{i\}, \phi, b_m, Low, 0, 0, 0)$

  It states none of buffer is placed or the cases for various kinds of high V_{dd} buffer being placed or the cases for various kinds of level converter being placed. Because some of the level converters needs both of the high and low V_{dd} voltage source and it easilier for us to transfer a high V_{dd} to a low V_{dd}, we only allowed the converters placed at a high V_{dd} region.

During filling initial solutions, solution at each sink node is pushed into a container called wave separately. The solutions in the waves will be propagated in next step.

3.3 Solution Propagation

Here we choose a wave w in each iteration, and then propagate every solution within w to neighboring nodes. While a solution s_A at node A propagates to solution s_B at the neighboring node B, there are three conditions regarding voltage islands to follow:

- If any sink voltage in s is high and source voltage is high, we cannot put any buffer within voltage island.
- If $signalV$ in s is high, the neighboring node cannot be placed low V_{dd} buffer.
- $rn_A \cap rn_B = \phi$

Due to these conditions, we generate a new solution s_{new} and store it in node B. We also store the solution in a wave with its rs. The new solution s_{new} is:

If $B_B = \phi$ (Node B does not place buffer):

- $cap_{new} = cap_B + cap_A + C_W$
- $rat_{new} = min(rat_B, rat_A - D_W)$
- $pow_{new} = pow_A + pow_B + P_w$
- $rn_{new} = rn_A \cup rn_B$
- $rs_{new} = rs_A \cup rs_B$
- $B_{new} = B_A \cup B_B$
- $signalV_{new} = signalV_A$
- $Cbl_{new} = 0$
- $bend_{new} = bend_A + bend_B + ((turn\ direction)?1 : 0)$
- $totLength_{new} = totLength_A + totLength_B + (distance\ between\ A\ and\ B)$

If $B_B \neq \phi$ (Assume buffer j places at node B with output resistance $R_\mathbf{j}$, delay $D_\mathbf{j}$, capacitance $C_\mathbf{j}$)

- $cap_{new} = C_j$
- $rat_{new} = min(D_1, D_2)$, $where\ D_1 = rat_B - R_j \cdot (C_w + cap_A)$; $D_2 = rat_A - (D_w + D_B + R_j \cdot Cbl_{new})$
- $pow_{new} = pow_A + P_w(Vdd\ based\ on\ driver) + P_B$
- $rn_{new} = rn_A \cup rn_B$
- $rs_{new} = rs_A \cup rs_B$
- $B_{new} = B_A \cup B_B$
- $SignalV_{new} = (B_B\ is\ a\ level\ converter)?low : (V_A || V_B)$
- $Cbl_{new} = cap_A + C_w + Cbl_B$

- $bend_{new} = bend_A + bend_B + ((turn\ direction)?1 : 0)$
- $totLength_{new} = totLength_A + totLength_B + (distance\ between\ A\ and\ B)$

3.4 Solution Pruning

In order to reduce runtime and the memory space usage, we have defined two prune conditions in modified RMP algorithm. Assume that there are two solutions s_A and s_B within the same node, we prefer to prune the solutions with more bends, and with power and capacitance dominance.

Prune with bends: If $bend_A > bend_B$, $totLength_A \geq totLength_B$, $rat_A \leq rat_B$, then s_A is dominated and can be pruned.

Prune with VG approach: If $signalV_A = signalV_B$, $pow_A < pow_B$, $cap_A \leq cap_B$, $rat_A \geq rat_B$, then s_B is dominated and can be pruned.

Chapter 4

Greedy Heuristic: Fast Voltage island Aware Buffered Tree Construction(FVABT)

Generally the modified RMP algorithm deals with a net that has n sink nodes, and the grid graph has size $N * M$ and K solutions with the same rs need propagation in each node. Because n sink node has 2^n kinds of combinations, the modified RMP algorithm has $O(2^n \cdot N \cdot M \cdot K)$ solutions during propagation. It is obvious that computation time will increase expontentially as number of sink increase. If modified RMP algorithm only process one sink at a time, the complexity can be reduced to $O(N \cdot M \cdot K)$.

Rather than handling 2^n kinds of sink combinations, FVABT performs modified RMP algorithm with one sink only at each iteration. In other words, because the propagation allows to merge solutions with different sink at each node in the grid graph, the modified RMP algorithm regards each node as a Steiner point. The solutions in the grid graph corresponding to various kinds of Steiner tree are generated. Instead of treating each node as a Steiner point, a desired solution is selected and then retrace the graph to erase the solutions that does not relate to the desired one. This approach can efficiently reduce Steiner points.

FVABT algorithm performs modified RMP algorithm for each sink, until all the sinks are handled. After one source-sink pair has finished the execution of the modified RMP algorithm, a desired lowest power solution is selected. Note that since the objective is low power, if we only keep the exact low power solution and prune others, in next iteration

modified RMP algorithm may not merge the solution with the stored low power solution due to timing violation problem. Therefore, one of the major issues in FVABT is to keep the useful solutions in the grid graph. Assume that after performing modified RMP to sink t_i and a desire solution sol_D has chosen. For each node in graph, the following solutions should be kept while erasing solutions:

- For every nodes in the graph, keep the initial solutions generated during filling initial solution step.

  Because the initial solutions are used such that a buffered routing tree could be generated at the source node, therefore, we need these initial solutions to generate the buffered routing tree for next sink.

- For nodes on the desired path that sol_D passed by, keeps solutions that $rs = \{t_i\}$ and solutions related with various kinds of buffer insertion on the desired path should be kept.

  The reason of keeping these two kinds of solution is that we want to keep some Steiner points for next sink's usage. Here, we use an example to explain. While performing FVABT for a net with 3 sink nodes, shown in Figure 4.1, t_1 is first processed with modified RMP algorithm because it is nearest to source node. A desire path between source node and t_1 is then select. The solutions of nodes on the desired path with $rs = \{t_1\}$ are kept. The reason is that the nodes on the desired path could be Steiner points while performing modified RMP for next sink t_2. While performing modified RMP for t_2, shown in Figure 4.2, t_2 could share with the former buffered routing tree by using node A as a Steiner point since solutions with $rs = \{t_1\}$ are kept at node A. After performing modified RMP on t_2, solutions of nodes on the desired path with $rs = \{t_2\}$ should be kept. Furthermore, solutions of nodes on the desired path with $rs = \{t_1, t_2\}$ should also be kept such that they can be used (nodes as Steiner points) when the path is possibly shared by handling next sink. Finally t3 is processed, if there exists another sink which needs to be processed, the useful solutions of the nodes on the desired path should be kept, shown in Figure 4.3.

The FVABT algorithm is shown in Figure 4.4. First, the waves in FVABT are different from the ones in the modified RMP. In order to pop the desired wave with target

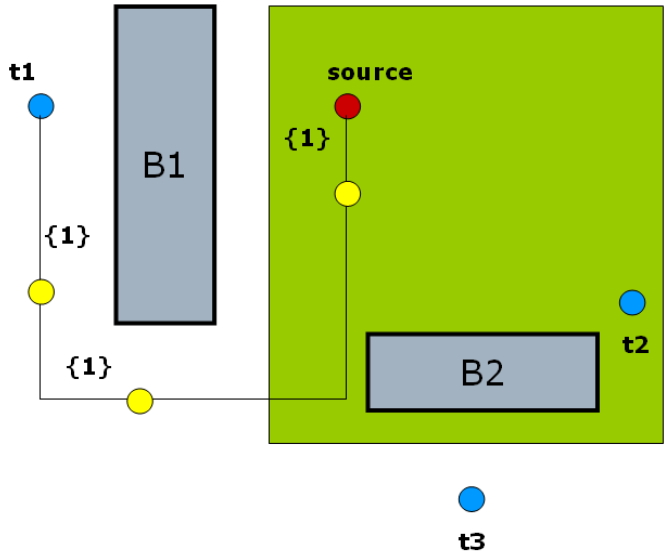

Figure 4.1: An example of performing FVABT to a net with 3 sink nodes, the yellow circles are buffer locations. After generating the buffered routing tree for t1, the solutions with $rs = \{t_1\}$ need to be kept for the node on the desired path could be Steiner points while performing modified RMP on next sink node.

sink nearest to source during each iteration, VABT only marks each wave with its target sink, while modified RMP marks each wave as the solution's reachable sinks of nodes within a wave. Therefore, solutions in a wave might have different reachable sinks in FVABT algorithm, but all these solutions are intended to bulid a tree for this target sink. After a wave with target sink nearest to source is fetched, solutions of nodes in this wave will be propagated to its neighbors with the solution propagation described in Section III.C (line 4-9). Once we have propagated one of its neighboring node, we prune the redundant new generated solutions and store the non-redundant solutions to the neighboring node. When one or more than one new generated solutions are stored, we store the neighbor to a temporary wave (line 10-14). Whenever we complete a wave propagation, we check whether source is visited. If visited, we choose a desired solution which has lowest power consumption and reach the target sink of this wave. If not visited, we again store the new generated wave (line 15-20). The program stops until all the sinks have been processed.

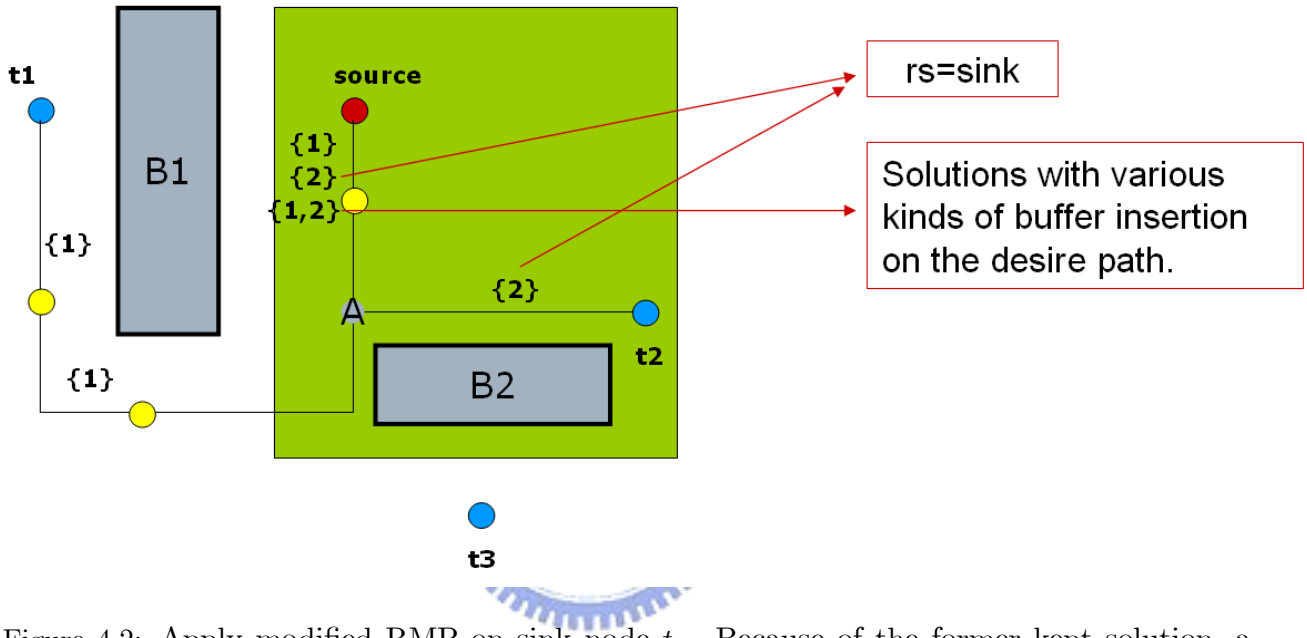

Figure 4.2: Apply modified RMP on sink node t_2. Because of the former kept solution, a buffered routing tree with a shared path between source and node A could be generated. After performing modified RMP to t_2, solutions of nodes on the desired path with $rs = \{t_2\}$ should be kept. Besides, solutions of nodes on the desired path with $rs = \{t_1, t_2\}$ should also be kept such that they can be used (nodes as Steiner points) when the path is possibly shared by handling next sink.

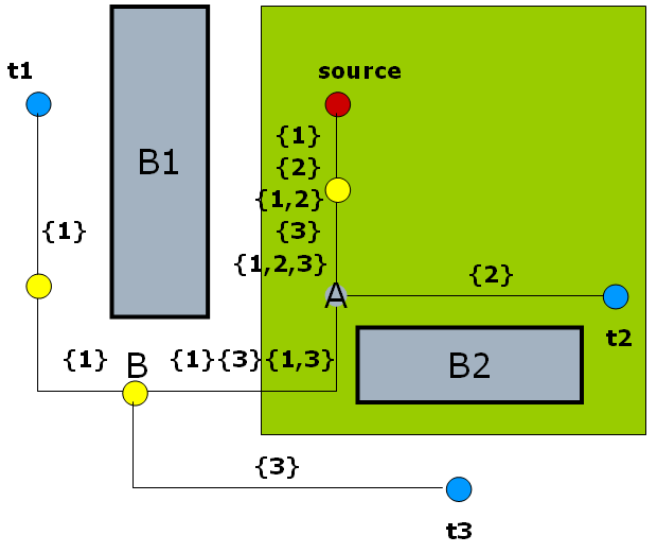

Figure 4.3: After performing modified RMP on sink node t3, if there exists another sink which needs to be processed, the solutions that we demonstrate in this figure should be kept.

Algorithm modified RMP algorithm with greedy heuristic

Input: 1. A grid graph builds by step 1~2
2. wavePool W: Classify solution at each sink as one wave, push all these waves into wavePool

Output: Solutions at source node. Each solution corresponds to a buffer tree

```
1. While (W is not empty )
2. {
3.      fetch a wave w with target sink nearest to source node.
4.      for each node n_i in w{
5.          for each solution s_i in n_i{
6.              for each node n_k which is n_i's neighbor{
7.                  propagate s_i with solution propagation manner to the
                    solutions n_k.
8.                  store the new generated solution in the temporary
                    container Q.
9.              }
10.             prune the redundant solutions in Q.
11.             if Q is not empty{
12.                 store the new generated solution from Q to n_k.
13.                 store n_k to a temporary wave w_temp
14.     }}}
15.     if w_temp contains the source node {
16.         pick up a desire solution with lowest power consumption.
17.         erase the useless solutions in the grid graph.
18.     }else{
19.         W=W∪{w_temp}
20.     }
21.}
```

Figure 4.4: FVABT (Modified RMP with greedy heuristic). Let modified RMP only applies on one sink. Once we process this sink, a desired solution is chosen, and the useless solutions are cleared in this grid graph. We keep on applying modified RMP algorithm until all the sinks are processed.

Chapter 5

Experimental Results

We have implemented modified RMP and our heuristic in C++ and the platform is on AMD Dempron 1.75GHz with 1GB memory. We randomly generate several test cases. All these cases have six obstacles, one voltage island, and ten buffer feasible locations, but the number of sinks, pin location assignment, and buffer location assignment are different. For each of these cases, the size of the grid graph is about 25*25 on a 17*17mm design, and the number in each file name shows the total number of source and sink pins.

The experimental results for modified RMP and our heuristic are shown in Table 5.1. Since our heuristic performs the tradeoff in finding solutions, the results present advantages in both algorithms for cases *net*4, *net*5, and *net*6. However our approach shows massive speedup over modified RMP algorithm. Moreover, when the number of sink is larger than six, modified RMP cannot find solutions in six hours.

The buffered routing trees of *net*5 and *net*6 are shown in Figure 5.1 and Figure 5.2. Figure 5.1 presents the case that the source node is within the voltage island. In order to avoid leakage current and keep signal integrity, level converter must be placed somewhere between the sink pin out of voltage island and source pin. Figure 5.2 presents the case that the source node is out of the voltage island. Since the voltage island might be turned off, none of buffer is placed inside the voltage island. If we examine those buffered routing tree in Figure 5.1 and Figure 5.2, we can find that those trees are not Minimum Reclinear Steiner Tree (MRST), which most routing algorithms intend to obtain. The reason is that MRST cannot fit on a design with voltage islands, not only for the timing violation problem but also for the signal integrity and the leakage current problems.

Table 5.1: Comparison between modified RMP and FVABT algorithm. The results present advantages in both algorithms for cases *net*4, *net*5, and *net*6. However our approach shows massive speedup over modified RMP algorithm. For larger cases, modified RMP cannot find solutions in six hours.

file	modified RMP			VABT		
name	delay (ps)	power (fJ)	CPU time (sec)	delay (ps)	power (fJ)	CPU time (sec)
net4	1205	9166	51	1276	8990	0.12 (425X)
net5	1299	5348	336	971	6148	0.04 (8400X)
net6	1513	7435	638	1462	7606	0.17 (3752X)
net10	-	-	>6hr	1306	13957	281
net13	-	-	>6hr	1956	15825	591
net15	-	-	>6hr	1631	16882	18.3

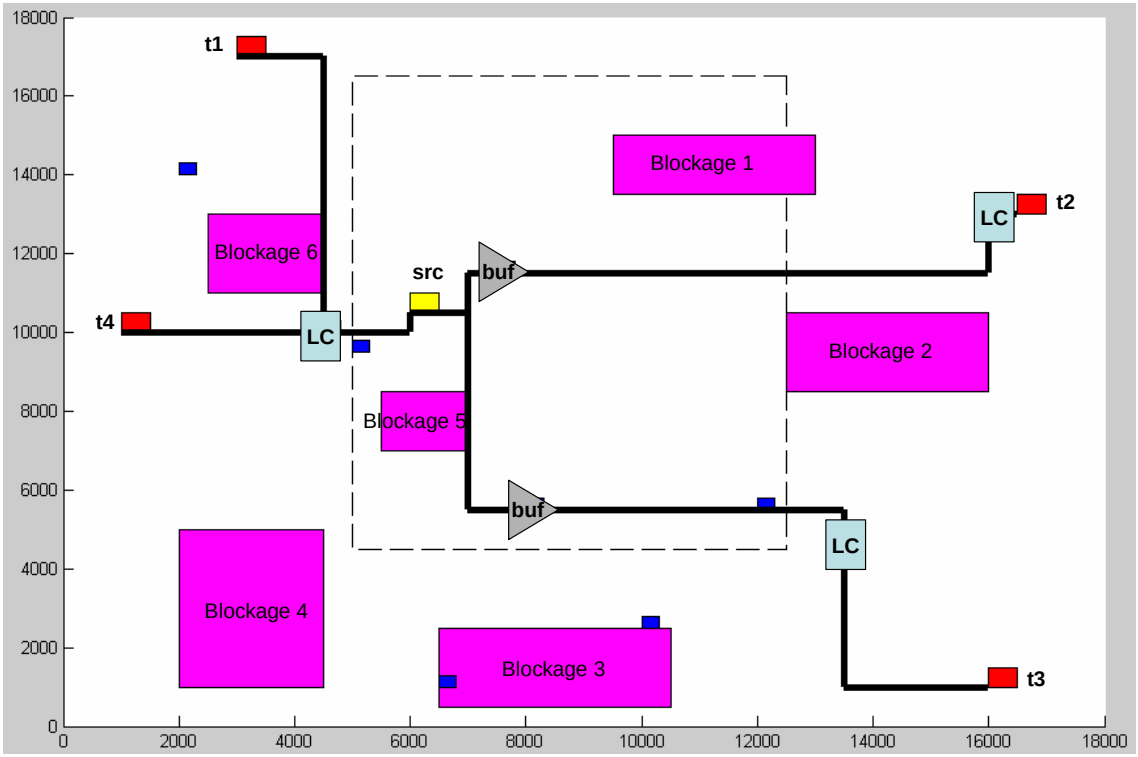

Figure 5.1: The resultant buffered routing tree of *net*5, which presents the case that a source node is within the voltage island. The little squares are feasible buffer lcoations where none of buffer places there. The rectangle circled with dotted line is the voltage island.

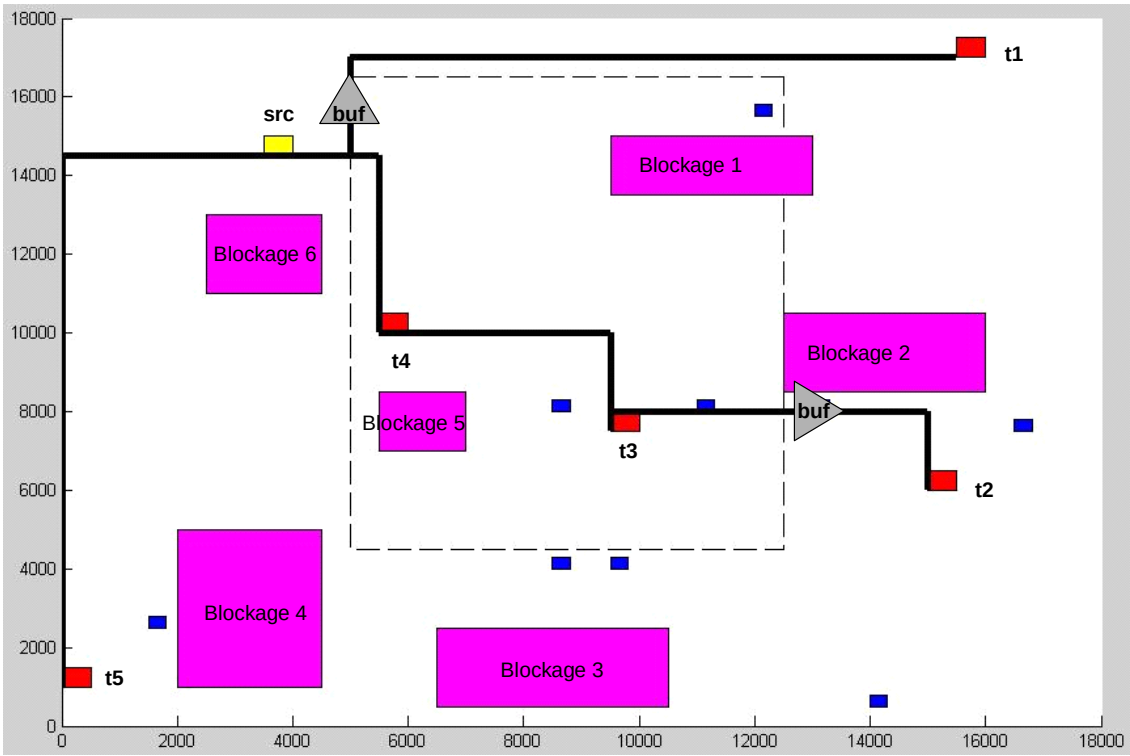

Figure 5.2: The resultant buffered routing tree of *net*6, which presents the case that a source node is out of the voltage island.

Chapter 6

Conclusion and Future Work

In this thesis, we have implemented modified RMP algorithm from [9] to deal with the designs in the presence of voltage islands. By using the prune techniques and a greedy heuristic, FVABT is much faster and can deal with multiple sinks net as number of sink increases. We plan to adopt the multilevel framework proposed in some routing algorithms to FVABT to make it even faster.

Bibliography

[1] Van Ginneken, "Buffer placement in distributed RC-tree networks for minimal Elmore delay" in Proc. of IEEE Int. Symp. Circuits Systems, May 1990, pp. 865–868.

[2] C. J. Alpert, A. Devgan, S.T. Quay, "Buffer insertion with accurate gate and interconnect delay computation" in Proc. of the Design Automation Conf., pp. 479–484, 1999.

[3] Y. Peng, X. Liu, "Freeze: Engineering a fast repeater insertion solver for power minimization using the Ellipsoid method" in Proc. of the Design Automation Conf., pp.813-818, 2005.

[4] Z. Li, C. N. Sze, C. J. Alpert, J. Hu, W. Shi, "Making fast buffer insertion even faster via approximation techniques" Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 13-18, Jan. 2005

[5] Z. Li, W. Shi, "A fast algorithm for optimal buffer insertion" IEEE Trans. on Computer-Aidede Design, vol. 24, no. 6, June 2005, pp. 879-891.

[6] J. Lillis, C. K. Cheng, T. Lin, "Optimal wire sizing and buffer insertion for low power and a generalized delay model" in IEEE International Conference on Computer Aided Design, pages 138–143, 1995.

[7] K. H. Tam, L. He, "Power optimal dual vdd buffered tree considering buffer stations and blockages" in Proc. of the Design Automation Conf., pp. 497–502, 2005.

[8] A. Youssef, M. Anis, M. Elmasry, "POMR: A power aware interconnect optimization methodology" IEEE Transaction on Very Large Scale Integration Systems, vol. 13, pp. 297-307, 2005

[9] J. Cong, X. Yuan, "Routing tree construction under fixed buffer location" in Proc. of the Design Automation Conference, pp. 379-384, 2000.

[10] W. Chen M. Pedram, P. Buch, "Buffered routing tree construction under buffer placement blockages" Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 381, 2002

[11] X. Tang, R. Tian, H. Xiang, D. F. Wong, "A new algorithm for routing tree construction with buffer insertion and wire sizing under obstacle constraints" in IEEE International Conference on Computer Aided Design, pp. 49–56, 2001.

[12] A. Dechu, C. Shen, C. Chu, "An efficient routing tree construction algorithm with buffer insertion, wire sizing, and obstacle considerations" IEEE Trans. on Computer-Aidede Design, vol. 24, no. 4, April 2005, pp. 600-608.

[13] M.Lai, D.F.Wong, "Maze routing with buffer insertion and wire sizing" in Proc. of the Design Automation Conf., pp. 374-378, 2000.

[14] J. Hu, Y. Shin, N.Dhanwada, and R. Marculescu, "Architecting voltage islands in core-based system-on-a-chip designs", IEEE International Symposium on Low Power Electronics and Design, pp. 180-185, 2004

[15] W. Hwang "New trends in low power SoC design technologies", IEEE International SOC Conference, pp. 422, 2003

[16] D.Lackey, P. Zuchowski, T. Bednar, D.Stout, S. Gould, and J. Cohn. "Managing power and performance for system-on-chip designs using voltage islands" in IEEE International Conference on Computer Aided Design, pages 195-202, 2002.

[17] H. Wu, I-Min. Liu, Martin D. F. Wong, and Y. Wang "Post-placement voltage island generation under performance requirement" in IEEE International Conference on Computer Aided Design, pages 309-316, 2005.

[18] H. Wu, Martin D. F. Wong, I-Min Liu "Timing-constrained and voltage island aware voltage assignment" in Proc. of the Design Automation Conf., 2006

[19] W.-P. Lee, H.-Y. Liu, and Y.-W. Chang "Voltage island aware floorplanning for power and timing optimization" in IEEE International Conference on Computer Aided Design, 2006.

作者簡歷

曾柏欽，民國六十六年九月出生於台中市。民國八十九年七月畢業於逢甲大學電機工程學系，並於民國九十三年九月進入國立交通大學電子研究所就讀，從事 VLSI 實體設計方面相關研究。民國九十五年七月取得碩士學位，碩士論文題目為『考慮存在電壓島的緩衝器繞線樹架構的一個有效率的演算法』。