INTERCONNECT TREE OPTIMIZATION ALGORITHM IN NANOMETER
VERY LARGE SCALE INTEGRATION DESIGNS

CHESSDA UTTRAPHAN EH KAN

A thesis submitted in fulfilment of the requirement for the award of the degree of
Doctor of Philosophy (Electrical Engineering)

Faculty of Electrical Engineering
Universiti Teknologi Malaysia

MARCH 2016
To Nukkul, Sukkith, Su Naas, Father and Mother
ACKNOWLEDGEMENT

I would like to express my deepest appreciation to all those who provided me the possibility to complete this thesis. A special gratitude I give to my supervisor, Dr. Nasir Shaikh Husin, whose contribution in stimulating suggestions and encouragement, helped me to coordinate my research especially in writing this thesis. I am also very thankful to Prof. Dr. Mohamed Khalil Mohd. Hani who gives countless ideas and supports in completing this thesis.

Furthermore, I would also like to acknowledge with much appreciation the crucial role of the staff member of VLSI research lab especially to En. Zulkifli Md. Yusof who contributed many ideas during the discussion of this work. Many thanks to Dr. Usman Ullah Sheikh for help me in debugging the codes. I am also indebted to Universiti Tun Hussein Onn (UTHM) and Ministry of Education, Malaysia for funding my Ph.D study.

My thanks, love and appreciations also go to my family. My wife, Su Naas who understands and always provide care and support throughout this challenging work. To my sons, Nukkul and Sukkrith who suffer from autism, I sincerely apologize if during this work I sometime spent less time with both of you.
ABSTRACT

This thesis proposes a graph-based maze routing and buffer insertion algorithm for nanometer Very Large Scale Integration (VLSI) layout designs. The algorithm is called Hybrid Routing Tree and Buffer insertion with Look-Ahead (HRTB-LA). In recent VLSI designs, interconnect delay becomes a dominant factor compared to gate delay. The well-known technique to minimize the interconnect delay is by inserting buffers along the interconnect wires. In conventional buffer insertion algorithms, the buffers are inserted on the fixed routing paths. However, in a modern design, there are macro blocks that prohibit any buffer insertion in their respective area. Most of the conventional buffer insertion algorithms do not consider these obstacles. In the presence of buffer obstacles, post routing algorithm may produce poor solution. On the other hand, simultaneous routing and buffer insertion algorithm offers a better solution, but it was proven to be NP-complete. Besides timing performance, power dissipation of the inserted buffers is another metric that needs to be optimized. Research has shown that power dissipation overhead due to buffer insertions is significantly high. In other words, interconnect delay and power dissipation move in opposite directions. Although many methodologies to optimize timing performance with power constraint have been proposed, no algorithm is based on grid graph technique. Hence, the main contribution of this thesis is an efficient algorithm using a hybrid approach for multi-constraint optimization in multi-terminal nets. The algorithm uses dynamic programming to compute the interconnect delay and power dissipation of the inserted buffers incrementally, while an effective runtime is achieved with the aid of novel look-ahead and graph pruning schemes. Experimental results prove that HRTB-LA is able to handle multi-constraint optimizations and produces up to 47% better solution compared to a post routing buffer insertion algorithm in comparable runtime.
ABSTRAK

# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECLARATION</td>
<td></td>
<td>ii</td>
</tr>
<tr>
<td>DEDICATION</td>
<td></td>
<td>iii</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td></td>
<td>iv</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td></td>
<td>v</td>
</tr>
<tr>
<td>ABSTRAK</td>
<td></td>
<td>vi</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td></td>
<td>vii</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td></td>
<td>xi</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td></td>
<td>xiv</td>
</tr>
<tr>
<td>LIST OF ABBREVIATIONS</td>
<td></td>
<td>xix</td>
</tr>
<tr>
<td>LIST OF APPENDICES</td>
<td></td>
<td>xxi</td>
</tr>
</tbody>
</table>

1 INTRODUCTION | 1
1.1 Overview | 1
1.2 Problem statement | 2
1.3 Research objectives | 6
1.4 Problem formulation | 7
1.5 Scope of works | 8
1.6 Research contributions | 9
1.7 Thesis outline | 9

2 LITERATURE REVIEW | 11
2.1 Interconnect routing | 11
2.1.1 Two-terminal nets routing | 12
2.1.2 Multi-terminal nets routing and topology |
3 FUNDAMENTAL THEORY AND MODELLING

3.1 Algorithm and complexity analysis
3.2 Dijkstra’s shortest path algorithm
   3.2.1 Implementation of Dijkstra’s algorithm using priority queue
   3.2.2 Time complexity of Dijkstra’s algorithm
3.3 Interconnect delay model
3.4 Power dissipation in buffered path interconnect
3.5 van Ginneken algorithm
3.6 Simultaneous routing and buffer insertion
   3.6.1 Modelling VLSI routing with buffer insertion as a shortest-path problem
   3.6.2 Simultaneous routing and buffer insertion for two-terminal nets
   3.6.3 Simultaneous routing and buffer insertion for multi-terminal nets
3.7 Delay and power model formulation for HRTB-LA
   3.7.1 Delay and power computation for upstream path expansion
   3.7.2 Delay and power computation for
downstream path expansion 57
3.8 Multi-constraint routing 59
3.9 Multi-constraint routing with look-ahead scheme 63
3.10 Summary 66

4 DESIGN AND DESCRIPTION OF HRTB-LA 67
ALGORITHM
4.1 HRTB-LA overview 67
4.2 Tree adjustment 68
4.3 Graph pruning 72
4.4 Path expansion and look-ahead scheme 78
  4.4.1 Path expansion in HRTB 78
  4.4.2 Path expansion in HRTB-LA 84
4.5 Time complexity of HRTB-LA 90
4.6 Numerical illustration of HRTB and HRTB-LA 91
  4.6.1 Numerical illustration of HRTB 92
  4.6.2 Numerical illustration of HRTB-LA 103
4.7 Summary 108

5 SOFTWARE DESIGN OF HRTB-LA 109
5.1 HRTB-LA data structure 109
  5.1.1 Data structure of the pre-processing data 110
  5.1.2 Data structure of the candidate solutions 110
5.2 Linked list functions 113
  5.2.1 Inserting data into a linked list 114
  5.2.2 Retrieving data from a linked list 114
5.3 Priority queue in HRTB-LA 115
5.4 Pseudo-code of HRTB-LA’s main function 115
5.5 Summary 118
6 VERIFICATION AND PERFORMANCE TEST OF HRTB AND HRTB-LA

6.1 Overview 119
6.2 Wire and buffer parameters 120
6.3 Verification of the proposed algorithm 122
  6.3.1 Verification for timing optimization 122
  6.3.2 Verification of the iterative power computation scheme 125
6.4 Performance test 1 – solution quality 130
6.5 Performance test 2 – solution quality, runtime and the number of candidate solutions 133
6.6 Performance test 3 – Delay-power optimization 140
6.7 Summary 146

7 CONCLUSIONS AND FUTURE WORKS 147

7.1 Conclusions 147
7.2 Future works 149

REFERENCES 152

Appendices A – C 160 - 189
LIST OF TABLES

<table>
<thead>
<tr>
<th>TABLE NO.</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Time complexity for heap operations for binary, binomial and Fibonacci heap (Cormen et al. 2009)</td>
<td>35</td>
</tr>
<tr>
<td>4.1</td>
<td>\textit{Rev}_i and \textit{Prev}_j for all vertices for the graph in Figure 4.9</td>
<td>81</td>
</tr>
<tr>
<td>4.2</td>
<td>List of the predicted end-to-end delay, end-to-end power and predicted end-to-end path length ( l_\text{P}(P) ) at vertex 8 in Figure 4.10</td>
<td>88</td>
</tr>
<tr>
<td>4.3</td>
<td>The values in the list and priority queue for path expansion from \textit{sink1} to \textit{Steiner} node in HRTB for graph in Figure 4.12. The key in the grey box indicates the lowest key value that will be extracted in the next \texttt{EXTRACT_MIN(Q)} (a) initial values of the list and priority queue (b) to (l) the values in the list and priority queue after 1\textsuperscript{st} extraction to 11\textsuperscript{th} extraction respectively (m) the values in the list and priority queue after the path expansions were completed (the text in red colour indicates that the candidate solution is dominated)</td>
<td>93</td>
</tr>
<tr>
<td>4.4</td>
<td>The values in the list and priority queue for path expansions from \textit{sink2} to \textit{Steiner} node in HRTB for graph in Figure 4.12. The key in the grey box indicates the lowest key value and it will be extracted in the next \texttt{EXTRACT_MIN(Q)} (a) initial values of the list and priority queue (b) and (c) the values after the 1\textsuperscript{st} and 2\textsuperscript{nd} extractions respectively (d) the final values after the queue is empty</td>
<td>100</td>
</tr>
</tbody>
</table>
4.5 The values in the list and priority queue for path expansions from the Steiner node to the source in HRTB for graph in Figure 4.12. The key in the grey box indicates the lowest key value and it will be extracted in the next EXTRACT_MIN(Q) (a) initial values of the list and priority queue (b) to (e) the values after 1st extraction to 4th extraction respectively (f) the final values after the queue is empty

4.6 The values in the list and priority queue for path expansions from the Steiner node to the source in HRTB for graph in Figure 4.12. The key in the grey box indicates the lowest key value and it will be extracted in the next EXTRACT_MIN(Q) (a) initial values of the list and priority queue (b) to (e) the values after 1st extraction to 4th extraction respectively (f) the final values after the queue is empty

4.7 The values in the list and priority queue for the path expansions from sink2 to Steiner node in HRTB-LA for graph in Figure 4.12. The key in the grey box indicates the lowest key value and it will be extracted in the next EXTRACT_MIN(Q) (a) initial values of the list and priority queue (b) and (c) the values after the 1st and 2nd extraction respectively

4.8 The values in the list and priority queue for path expansions from the Steiner node to the source in HRTB-LA for graph in Figure 4.12. The key in the grey box indicates the lowest key value and it will be extracted in the next EXTRACT_MIN(Q) (a) initial values of the list and priority queue (b) to (d) the values after the 1st extraction to 3rd extraction respectively

5.1 Attributes of a candidate solution

6.1 Wire dimension and parameters

6.2 Buffer library

6.3 Characteristics of the test nets and graphs
| 6.4 | Solution from HRTB and FBI running on test nets | 125 |
| 6.5 | Delay at source comparison between FBI, RIATA, HRTB and HRTB-LA | 132 |
| 6.6 | Solution quality, runtime and number of candidate solutions for net N5 | 135 |
| 6.7 | Solution quality, runtime and number of candidate solutions for net N10 | 135 |
| 6.8 | Solution quality, runtime and number of candidate solutions for net N25 | 136 |
| 6.9 | Performance comparison between HRTB and HRTB-LA for net N5 | 142 |
| 6.10 | Performance comparison between HRTB and HRTB-LA for net N25 | 142 |
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE NO.</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>(a) Buffer insertion on fixed routing tree that ignores buffer obstacles (b) buffer insertion on fixed routing tree that avoids obstacles (c) buffer insertion on the fixed routing tree with tree adjustment (RIATA) and (d) simultaneous routing tree and buffer insertion on the adjusted tree</td>
<td>5</td>
</tr>
<tr>
<td>1.2</td>
<td>A tree on uniform grid graph $G = (V, E)$</td>
<td>8</td>
</tr>
<tr>
<td>2.1</td>
<td>(a) Rectilinear minimum spanning tree (R-MST) (b) rectilinear Steiner minimal tree (R-SMT). Hollow dots indicate net terminals while solid dots are the Steiner nodes</td>
<td>13</td>
</tr>
<tr>
<td>2.2</td>
<td>(a) A wire of length $L$ and (b) Corresponding $\pi$-model RC circuit</td>
<td>17</td>
</tr>
<tr>
<td>2.3</td>
<td>A wire inserted with $N-I$ number of buffers</td>
<td>18</td>
</tr>
<tr>
<td>3.1</td>
<td>$O$-notation gives an upper bound for a function to within a constant factor (Cormen et al. 2009)</td>
<td>30</td>
</tr>
<tr>
<td>3.2</td>
<td>Runtime trend of algorithms</td>
<td>31</td>
</tr>
<tr>
<td>3.3</td>
<td>Relaxation on edge $(u, v)$ with weight $w(u, v) = 3$. (a) The relaxation step when $v.d &gt; u.d + w(u, v)$ and (b) The relaxation step when $v.d &lt; u.d + w(u, v)$</td>
<td>33</td>
</tr>
<tr>
<td>3.4</td>
<td>A $5 \times 4$ grid graph</td>
<td>34</td>
</tr>
</tbody>
</table>
3.5 Illustration of Dijkstra’s algorithm on general graph, with a as the source (a) initialization (b) path expansion from $a \rightarrow b$, $a \rightarrow c$ (c) path expansion from $c \rightarrow d$, $c \rightarrow e$ (d) path expansion from $b \rightarrow d$ (e) path expansion from $d \rightarrow e$, $d \rightarrow f$ (f) final expansion from $e \rightarrow f$ gives shortest path from $a \rightarrow c \rightarrow d \rightarrow e \rightarrow f$ with cost = 7

3.6 Illustration of Dijkstra’s algorithm on uniform grid graph (a) Initial graph $v,d = \infty$ (b) expansion $1 \rightarrow 2$ and $4$ (c) expansion $2 \rightarrow 3$ and $5$ (d) expansion $4 \rightarrow 5$ and $7$ (e) expansion $3 \rightarrow 6$ (f) expansion $5 \rightarrow 6$ and $8$ (g) expansion $7 \rightarrow 8$ (h) expansion $6 \rightarrow 9$ (i) expansion $8 \rightarrow 9$

3.7 A simple RC tree illustrating the process of calculating Elmore delay

3.8 Buffered path interconnect

3.9 Fixed routing tree connecting source node to the Steiner node and all sink nodes

3.10 Candidate solutions at each node. The red colour is the best solution for the given path

3.11 Path expansion in 2-terminal net simultaneous routing and buffer insertion (a) expansion from Sink node to vertices 5, 9 and 15 (b) expansion from vertex 5 to vertex 4

3.12 The 2D grid graph representing the interconnect tree in Figure 3.9

3.13 Simultaneous routing and buffer insertion in multi-terminal net. The arrows show the direction of the path expansions

3.14 Wire expansion from vertex $v$ to vertex $u$ for upstream path expansion

3.15 Wire expansion from vertex $v$ to vertex $u$ and insert buffer at $v$
<table>
<thead>
<tr>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.16</td>
<td>Wire expansion from vertex $u$ to vertex $v$ for downstream path expansion</td>
</tr>
<tr>
<td>3.17</td>
<td>Wire expansion from vertex $u$ to vertex $v$ and insert buffer at $v$</td>
</tr>
<tr>
<td>3.18</td>
<td>Path expansion in multi-constraint graph (a) Initialization (b) first path expansion (c) expansion for path $c \rightarrow d$, $c \rightarrow e$ (d) expansion for path $b \rightarrow d$ (e) expansion for path $e \rightarrow f$ (f) expansion for path $d \rightarrow e$, $d \rightarrow f$ extracted from $0.6$ (g) expansion for path $d \rightarrow e$, $d \rightarrow f$ extracted from $0.7$ (h) expansion for path $e \rightarrow f$</td>
</tr>
<tr>
<td>3.19</td>
<td>Path expansion in multi-constraint graph with look-ahead scheme (a) initialization (b) first path expansion (c) expansion for path $c \rightarrow d$ (d) expansion for path $d \rightarrow f$ (e) expansion from path $b \rightarrow d$</td>
</tr>
<tr>
<td>4.1</td>
<td>Main stages in HRTB-LA</td>
</tr>
<tr>
<td>4.2</td>
<td>Tree adjustment technique (a) a Steiner node $m$ is inside the obstacle (b) an alternative Steiner node $m'$ is generated (Hu et al. 2003)</td>
</tr>
<tr>
<td>4.3</td>
<td>Flowchart of the tree adjustment</td>
</tr>
<tr>
<td>4.4</td>
<td>Sample tree for illustrating tree adjustment</td>
</tr>
<tr>
<td>4.5</td>
<td>Flowchart of the graph pruning</td>
</tr>
<tr>
<td>4.6</td>
<td>Vertex $v$ is pruned as $L_{\text{ToEnd}}[v] + L_{\text{ToStart}}[v] &gt; L_{\text{StartEnd}}$</td>
</tr>
<tr>
<td>4.7</td>
<td>Graph pruning in HRTB-LA (a) initial graph (b) graph pruning for $\text{sink1} \rightarrow \text{Steiner}$ node expansions (c) graph pruning for $\text{sink2} \rightarrow \text{Steiner}$ node expansions (d) graph pruning for $\text{Steiner}$ node $\rightarrow \text{Source}$ expansions</td>
</tr>
<tr>
<td>4.8</td>
<td>Flowchart of the path expansion in the proposed algorithm</td>
</tr>
<tr>
<td>4.9</td>
<td>Example of candidate solutions at each vertex</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>4.10</td>
<td>Sample routing graph and path expansion of the proposed algorithm</td>
</tr>
<tr>
<td>4.11</td>
<td>The look-ahead weight vectors for the graph in Figure 4.10</td>
</tr>
<tr>
<td>4.12</td>
<td>Sample grid graph</td>
</tr>
<tr>
<td>4.13</td>
<td>Routing solution</td>
</tr>
<tr>
<td>4.14</td>
<td>Look-ahead weight vectors for (a) first set (b) second set</td>
</tr>
<tr>
<td>5.1</td>
<td>Node labelling for a net with two sinks in HRTB-LA</td>
</tr>
<tr>
<td>5.2</td>
<td>Illustration of Previ and Prevj attributes</td>
</tr>
<tr>
<td>5.3</td>
<td>Linked list for vertex v with three candidate solutions</td>
</tr>
<tr>
<td>6.1</td>
<td>Sample tree with 5 sinks</td>
</tr>
<tr>
<td>6.2</td>
<td>Solution from FBI algorithm</td>
</tr>
<tr>
<td>6.3</td>
<td>Solution from HRTB algorithm</td>
</tr>
<tr>
<td>6.4</td>
<td>Illustration of the iterative power computation (a) sample net</td>
</tr>
<tr>
<td></td>
<td>(b) upstream computation (c) downstream computation</td>
</tr>
<tr>
<td>6.5</td>
<td>Illustration of the iterative power computation for multi buffer types</td>
</tr>
<tr>
<td></td>
<td>(a) sample net (b) upstream computation (c) downstream computation</td>
</tr>
<tr>
<td>6.6</td>
<td>Plot of net N5 test results (a) slack at source (b) runtime (c) number of candidate solutions</td>
</tr>
<tr>
<td>6.7</td>
<td>Plot of net N10 test results (a) slack at source (b) runtime (c) number of candidate solutions</td>
</tr>
<tr>
<td>6.8</td>
<td>Plot of net N25 test results (a) slack at source (b) runtime (c) number of candidate solution</td>
</tr>
<tr>
<td>6.9</td>
<td>Plot of net N5 test results for delay-power constraint optimization (a) runtime (b) number of candidate solutions</td>
</tr>
<tr>
<td>6.10</td>
<td>Plot of net N25 test results for delay-power constraint optimization (a) runtime (b) number of candidate solutions</td>
</tr>
</tbody>
</table>
6.11 Routing solutions for 4-sink net for different power constraints
(a) routing solution for maximum slack with no power constraint (b) routing solution when power was constrained at 30 mW (c) routing solution when power was constrained at 20 mW

7.1 Sample net where a pair of nodes are on the same horizontal line (a) before pruning (b) effective search space for path expansions between sink1 and Steiner node

A1 Fibonacci heap structure

A2 Inserting a node into a Fibonacci heap (a) a Fibonacci heap $H$
(b) Fibonacci heap $H$ after inserting the node with key 21

A3 The process of EXTRACT_MIN($H$) (a) meld the child nodes into root list (b) label the rank (c) to (e) mark current node and updating rank list from left to right (f) link 23 into 17 (g) to (h) link 24 into 7 (i) to (k) link 41 into 18 (l) final heap

A4 DECREASE-KEY($H$, $x$, $k$), if the heap order not violated (a)
initial heap structure (b) the key of $x$ is decreased from 46 to 29

A5 DECREASE-KEY($H$, $x$, $k$), if the heap order is violated (a)
decrease the key (b) cut the tree rooted at $x$, meld into the root list and former parent node is marked
## LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-Tree</td>
<td>Arborescence-Tree</td>
</tr>
<tr>
<td>BPRIM</td>
<td>Bounded Prim</td>
</tr>
<tr>
<td>BRBC</td>
<td>Bounded Radius Bounded Cost</td>
</tr>
<tr>
<td>BR-MRT</td>
<td>Bounded Radius - Minimum Routing Tree</td>
</tr>
<tr>
<td>CMOS</td>
<td>Complementary Metal Oxide Semiconductor</td>
</tr>
<tr>
<td>C-Tree</td>
<td>Clustered-Tree</td>
</tr>
<tr>
<td>DP</td>
<td>Dynamic Programming</td>
</tr>
<tr>
<td>ED</td>
<td>Elmore Delay</td>
</tr>
<tr>
<td>FBI</td>
<td>Fast Buffer Insertion</td>
</tr>
<tr>
<td>HRTB</td>
<td>Hybrid Routing Tree and Buffer insertion</td>
</tr>
<tr>
<td>HRTB-LA</td>
<td>Hybrid Routing Tree and Buffer insertion with Look-Ahead</td>
</tr>
<tr>
<td>ITRS</td>
<td>International Technology Roadmap for Semiconductors</td>
</tr>
<tr>
<td>L-RST</td>
<td>L-shaped Rectilinear Steiner Tree</td>
</tr>
<tr>
<td>MCOP</td>
<td>Multi-Constraint Optimal Path</td>
</tr>
<tr>
<td>MCP</td>
<td>Multi-Constraint Path</td>
</tr>
<tr>
<td>MOS</td>
<td>Metal Oxide Semiconductor</td>
</tr>
<tr>
<td>MRSA</td>
<td>Minimum Rectilinear Steiner Arborescence</td>
</tr>
<tr>
<td>MST</td>
<td>Minimum Spanning Tree</td>
</tr>
<tr>
<td>NP</td>
<td>Non-deterministic Polynomial time</td>
</tr>
<tr>
<td>PDF</td>
<td>Probability Density Function</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality-of-Service</td>
</tr>
<tr>
<td>RAT</td>
<td>Required Arrival Time</td>
</tr>
<tr>
<td>RC</td>
<td>Resistor-Capacitor</td>
</tr>
<tr>
<td>RIATA</td>
<td>Repeater Insertion with Adaptive Tree Adjustment</td>
</tr>
<tr>
<td>RLC</td>
<td>Resistor-inductor-Capacitor</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>RMP</td>
<td>Recursive Merging and Pruning</td>
</tr>
<tr>
<td>R-MST</td>
<td>Rectilinear Minimal Spanning Tree</td>
</tr>
<tr>
<td>R-SMT</td>
<td>Rectilinear Steiner Minimal Tree</td>
</tr>
<tr>
<td>RTBW</td>
<td>Routing Tree with Buffer insertion and Wire sizing</td>
</tr>
<tr>
<td>SAMCRA</td>
<td>Self-Adaptive Multi-Constrained Routing Algorithm</td>
</tr>
<tr>
<td>SMT</td>
<td>Steiner Minimal Tree</td>
</tr>
<tr>
<td>S-RABILA</td>
<td>Simultaneous Routing and Buffer Insertion with Look-Ahead</td>
</tr>
<tr>
<td>VLSI</td>
<td>Very Large Scale Integration</td>
</tr>
</tbody>
</table>
# List of Appendices

<table>
<thead>
<tr>
<th>Appendix</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Fibonacci heap operations</td>
<td>160</td>
</tr>
<tr>
<td>B</td>
<td>C Code for HRTB-LA algorithm</td>
<td>165</td>
</tr>
<tr>
<td>C</td>
<td>Output sample</td>
<td>187</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

1.1 Overview

The demand for high speed and low power consumption for today’s applications has forced dramatic changes in the design and manufacturing methodologies for very large scale integration (VLSI) circuits (Celik et al. 2002; Ekeke 2010; ITRS 2012; 2013). To meet the demand, the number of devices (i.e. transistors) on a single chip must be increased and this requires decrease of the device size and also will need a larger layout area to support huge amounts of devices.

As the size of devices decreases and the device operates at a higher speed, the interconnect delay becomes much more significant compared to the device delay. Most of the delay in integrated circuits is due to the time it takes to charge and discharge the capacitance of the wires and the gates of the transistors. The resistance $R = rl$ of a wire increases linearly with its length $l$ and so does its capacitance $C = cl$. Where $c$ and $r$ are unit capacitance and unit resistance respectively. Hence, the RC delay of the wire is $D = \frac{1}{2}RC = \frac{1}{2}rcI^2$ (van Ginneken 1990). Clearly, the delay increases quadratically with the length of the wire (Saxena et al. 2004; ITRS 2012).

One of the effective techniques to reduce the interconnect delay is by inserting a buffer to restore the signal strength along the interconnect tree. As design dimensions continue to shrink, more and more buffers are needed to improve the performance. However, buffer itself consumes power and it has been shown that power dissipation overhead due to optimal buffer insertion is significantly high (Ekeke 2010).
According to (Saxena et al. 2004), the critical inter buffer length (the minimum wire segment length where the buffer is required) decreased at the rate of 68% when the VLSI technology migrates from 90 nm to 45 nm. This inter buffer length scaling significantly outpaces the VLSI technology scaling which is roughly 0.5 times for every two generations. The total block cell count made up of buffers will reach 35% in the 45-nm technology node and 70% in 32-nm technology.

The dramatic buffer scaling undoubtedly generates large and profound impact on VLSI circuit design. With millions of buffers required per chip, almost nobody can afford to neglect the importance of optimal buffer insertion as compared to a decade ago when only a few thousands of buffers are needed for a chip (Cong 1997). Because of this importance, buffer insertion algorithms and methodologies need to be deeply studied on various aspects. First, a buffer insertion algorithm should deliver solutions of high quality because interconnect and circuit performance largely depend on the way that buffers are placed. Second, a buffer insertion algorithm needs to be sufficiently fast so that millions of nets can be optimized in reasonable time. Third, accurate delay models are necessary to ensure that buffer insertion solutions are reliable. Fourth, buffer insertion techniques are expected to simultaneously handle multiple objectives, such as timing, power and signal integrity (Alpert et al. 2009).

1.2 Problem statement

Interconnect is a wiring system that distributes clock and other signals to the various functional blocks of a CMOS integrated circuit. When the VLSI technology is scaled down, gate delay and interconnect delay change in opposite directions. Smaller devices lead to less gate switching delay. In contrast, thinner wire leads to increased wire resistance and greater signal propagation delay along wires. As a result, interconnect delay has become a dominating factor for VLSI circuit performance (ITRS 2012; 2013).
Among the available techniques, buffer insertion has been proven to be one of the best techniques to reduce the interconnect delay for a long wire. The main challenge in interconnect buffer insertion is how to determine optimal number of buffers and their placement in the given interconnect tree. The most influential and systematic technique was proposed by (van Ginneken 1990). Given the possible buffer locations, this algorithm can find the optimum buffering solution for the fixed signal routing tree that will maximize timing slack at the source according to Elmore delay model (Elmore 1948). As the number of buffers inserted in the circuits increases dramatically, an algorithm that is fast and efficient is essential for the design automation tools. van Ginneken's algorithm utilized dynamic programming which tries to find an optimal solution to a problem by first finding optimal solutions to sub problems and then merging them to find an optimal solution to the larger problem.

Recently, many techniques to speedup van Ginneken's algorithm and its extensions were proposed such as in (Lillis et al. 1996), (Shi and Li 2003), (Shi and Li 2005), (Li and Shi 2006) and (Li et al. 2012). However, van Ginneken's algorithm and its extensions can only operate on fixed routing tree. They will give optimal solution when the best routing tree is given but produce a poor solution when a poor routing tree is provided especially when there are obstacles in the designs. In today's VLSI design, some regions may be occupied by predesigned libraries such as IP blocks and memory arrays. Some of these regions do not allow buffer or wire to pass through and some regions only allow wire to go through but are restricted for any buffer insertion. Therefore, buffer insertion has to be performed with consideration of this buffer and wire obstacles (Alpert et al. 2009; Khalil-Hani and Shaikh-Husin 2009). The best way to handle the obstacles is to perform the routing and buffer insertion simultaneously using a grid graph technique. However, research has shown that simultaneous routing and buffer insertion is NP-complete (Hu et al. 2009). The available known techniques today are either explore dynamic programming to compute optimal solution in the worst-case exponential time or design efficient heuristic without performance guarantee.

The dynamic programming algorithm such as RMP (recursive merging and pruning) algorithm can find an optimal buffering solution for multi-terminal nets
(Cong and Yuan 2000), but it is not efficient when the number of sinks and the number of possible buffer locations are big as the search space is very large. Indeed, (Hu et al. 2003) show that the searching in RMP is NP-complete, and they also proposed a heuristic algorithm to solve multi-pin nets buffer insertion problem by constructing a performance driven Steiner tree and create an alternative Steiner node if the original Steiner node is inside the obstacle. The algorithm is called RIATA for Repeater Insertion with Adaptive Tree Adjustment. RIATA is very fast because it operates on a fixed tree. However, the quality of the solution may not be good enough if many paths of the adjusted tree still overlap with the buffer obstacles as illustrated in Figure 1.1.

Figure 1.1 shows example of possible solutions for a net with a tree structure (multi-terminal) where the grey areas represent buffer obstacles. It has three sinks \( s_1, s_2 \) and \( s_3 \) with \( s_0 \) as the source. In this illustration, appropriate parameters for wires and buffers are applied (will discuss in detail in Chapter 2). Figure 1.1(a) shows the solution from van Ginneken’s algorithm where the slack at source is -899.74 ps (the slack is the required arrival time at sink minus the accumulated delay). This means that the timing is not met because most of the routing paths are inside the buffer obstacles where buffer insertion is not allowed. One can rerout the tree such that all the paths avoid the buffer obstacles as shown in Figure 1.1(b). The slack is improved to -44.39 ps but still violates the timing requirement due to increased wire length. The tree adjustment technique according to RIATA produces a solution as shown in Figure 1.1(c). Now the timing requirement is met, with slack at source of 11.64 ps. RIATA is efficient in terms of runtime but its solution quality still depends on its newly generated tree. If most of the paths are inside the buffer obstacles, the room for timing improvement is still limited.

Instead of fully constructing the routing path simultaneously with buffer insertion like in RMP algorithm, one can utilize the simultaneous approach on the adjusted tree. Figure 1.1(d) illustrates the routing path generated by this approach. The slack obtained at source is improved to 217.65 ps. Clearly, this hybrid technique produces the best result compared to the techniques that perform buffer insertion on the fixed routing path like van Ginneken’s algorithm (and its extensions) and RIATA. The runtime of this hybrid technique can be improved by adopting the technique called
look-ahead proposed by (Shaikh-Husin 2008; Khalil-Hani and Shaikh-Husin 2009) to solve the simultaneous routing and buffer insertion for two terminals (single-sink) net problems.

![Diagram](image_url)

**Figure 1.1** (a) Buffer insertion on fixed routing tree that ignores buffer obstacles (b) buffer insertion on fixed routing tree that avoids obstacles (c) buffer insertion on the fixed routing tree with tree adjustment (RIATA) and (d) simultaneous routing tree and buffer insertion on the adjusted tree

Another issue that the previous dynamic programming algorithms did not take into consideration is power consumed by the buffers inserted along the interconnect tree. It has been found that power dissipation overhead due to optimal buffer insertion is significantly high and can be as high as 20% of total chip power dissipation (Nalamalpu and Burleson 2001). Hence, in addition to timing performance, power dissipation constraint should also be integrated into buffer insertion algorithm (Nalamalpu and Burleson 2001; Ekekwe 2010). Many methodologies to optimize propagation delay with power constraint have been proposed, (Nalamalpu and
Burleson 2001; Banerjee and Mehrotra 2002; Wason and Banerjee 2005; Li et al. 2005; Narasimhan and Sridhar 2010) but none of them can be integrated into buffer insertion algorithm that is based on dynamic programming on grid graph. The grid graph technique is used because the simultaneous routing and buffer insertion utilizes the maze search algorithm that is best implemented using the graph search algorithm (Cormen et al. 2009). Furthermore, the uniform grid graph allows the buffers to be inserted anywhere (except in buffer obstacle areas), hence, improve the solution quality. Meanwhile, the advantage of dynamic programming is that it allows the use of multiple buffer types.

From the discussion above, the problem is now summarized as follows; buffering in a multi-terminal net is known to be NP-complete and the existing available algorithms that give an optimum solution is too slow while heuristic algorithms are fast but produce poor solutions. Even though buffer insertion is one of the most studied problems in VLSI physical design, finding an efficient algorithm with provably good performance still remains an active research area. Also, as design dimensions continuously shrink, more and more buffers are needed to improve the performance (i.e. speed and signal integrity) of the designs but the buffer itself consumes power. Therefore, we need a new algorithm that is capable to handle these constraints efficiently.

1.3 Research objectives

The objectives of this research are as follows:

(1) To propose an efficient graph-based maze routing and buffer insertion algorithm for nanometer VLSI layout designs. The algorithm is designed for multi-terminal nets and multi-constraint optimization. The constraints are as follows; routing obstacles, timing performance and power dissipation.

(2) To propose a power computation scheme for the proposed algorithm that can be computed iteratively based on dynamic programming framework.
1.4 Problem formulation

The simultaneous routing and buffer insertion problem in VLSI layout design is essentially a buffered routing path search problem. In this work, it is formulated as a shortest-path problem in a weighted graph specified as follows. Given a routing grid graph $G = (V, E)$ corresponding to VLSI layout where $v \in V$ and $e \in E$ is a set of internal vertices and a set of internal edges respectively, with a source vertex $S_0 \in V$, $n$ sink vertices $s_1, s_2, \ldots, s_n \in V$, $n - 1$ Steiner vertices $m_1, m_2, \ldots, m_{n-1} \in V$, required arrival time $\text{RAT}(s_1), \text{RAT}(s_2), \ldots, \text{RAT}(s_n)$, a power constraint $P_c$, a buffer library $B$, and a wire parameter $W$. The goal is to find a routing tree simultaneously with buffer insertion such that the slack at source and power dissipation of buffers satisfy the given constraints. A vertex $v_i \in V$ may belong to the set of buffer obstacle vertices, denoted $V_{OB}$ or a set of wire obstacle vertices, denoted as $V_{OW}$. A buffer library $B$ contains different types of buffer. For each edge $e = u \rightarrow v$, signal travels from $u$ to $v$, where $u$ is the upstream vertex and $v$ is the downstream vertex and $u, v \notin V_{OW}$. A uniform grid graph illustrating some of the parameters for the problem formulation is shown in Figure 1.2.

The proposed algorithm is called HRTB-LA which stands for Hybrid Routing Tree and Buffer insertion with Look-Ahead. Instead of a fixed routing tree as in van Ginneken’s algorithm and RIATA, we use maze routing to find the solution. However, HRTB-LA will not explore the entire 2D graph as in RMP because we use an initial tree as a reference for determining the Steiner nodes as in RIATA. We also incorporate the technique of graph pruning and look-ahead to speed up the runtime of the algorithm.
1.5 Scope of works

The scopes of this research are as follows:

(a) Elmore delay metric is used to calculate the interconnect delay due to its high fidelity and speed (Alpert et al. 2007; Li et al. 2012; ITRS 2012; 2013).

(b) Uniform grid graph is used to represent VLSI layout and maze routing (Zhou et al. 2000; Khalil-Hani and Shaikh-Husin 2009) is used for path search.

(c) There are many algorithms for tree construction in VLSI routing (i.e. Steiner minimal tree) and the Steiner tree construction itself is a hard problem. Therefore, we assume that the pre-processing tree is available.

(d) The performance of the proposed algorithm is benchmarked with available similar algorithms.
1.6 Research contributions

We propose a new algorithm for simultaneous tree construction and buffer insertion with multi-constraint optimization. The contributions of this research can be listed as follows:

(a) The concept of look-ahead scheme (Khalil-Hani and Shaikh-Husin 2009) which is proven to be efficient for two-terminal (single-sink) nets is adopted into this work such that it can handle multi-terminal nets.

(b) The algorithm is designed such that it can also optimize multiple constraints such as obstacles, timing and power dissipation of the buffered interconnect tree.

(c) The iterative computation of power dissipation in dynamic programming framework is proposed.

(d) In algorithm design, the time complexity of the algorithm is used to measure the efficiency of the algorithm. Therefore, the time complexity of HRTB-LA is analysed and presented.

1.7 Thesis outline

This thesis consists of seven chapters. Chapter 1 presents research background, problem formulation and objectives of the research. The literature review is presented in Chapter 2 which discusses the evolution of interconnect optimization techniques ranging from two-terminal to multi-terminal nets. Next, the post routing optimization (focusing on buffer insertion) is discussed. In this section, we reviewed the buffer insertion algorithms on fixed tree followed by the simultaneous routing and buffer insertion algorithms. Lastly, the buffer insertion algorithms with multi-constraint optimization and other delay models are discussed.
Chapter 3 presents research background and the theories associated with this research. First, the concept of algorithm and its complexity analysis is presented followed by Dijkstra’s shortest path algorithm. Next, the Elmore delay and power dissipation in buffered interconnect are discussed. The details of buffering algorithms are also discussed in this chapter. Lastly, the delay and power formulation for the proposed algorithm is presented followed by the fundamental concept of multi-constraint routing and the look-ahead scheme.

Chapter 4 presents the design description of the proposed algorithm, HRTB-LA. The main stages of HRTB-LA are discussed in detail. The path expansion process, which is the core of the algorithm, is presented with the aid of numerical examples. We present two types of path expansion which are (1) the normal path expansion without look-ahead scheme and (2) path expansion with look-ahead scheme. The numerical examples demonstrate the advantages of the novel look-ahead scheme in HRTB-LA.

Chapter 5 gives detail descriptions of the software design of HRTB-LA. It focuses on the data structures that are used by the algorithm which are; array data structure, linked list data structure and priority queue implemented using a heap data structure. The pseudo-code of HRTB-LA’s main functions are also presented in this chapter.

Chapter 6 presents the verification and performance test of the proposed algorithm. HRTB-LA is benchmarked with other similar algorithms and the results are presented. And finally, Chapter 7 concludes the research and recommendations for future works are given.
CHAPTER 2

LITERATURE REVIEW

This chapter surveys the evolution of interconnect optimization which includes: interconnect routing and post routing optimizations with emphasis on buffer insertion algorithms. The advance optimization techniques such as simultaneous routing and buffer insertion, and multi-constraint optimization techniques are also presented.

2.1 Interconnect routing

There are two major steps in physical design. First, the placement step where all the functional modules are mapped onto the surface of the chip. Second, the routing step interconnects the terminals for all the modules to allow the travel of signals between modules. The latter is our main focus in this research. The fundamental routing objectives are: (a) minimization of total wire length, (b) minimization of signal delay and (c) minimization of skew among signal arrival times. In general, minimizing total wire length will also reduce the delay because signal delay is proportional to wire capacitance and resistance, which increase with length. Skew minimization is not our interest in this research as it is especially important for clock routing.

VLSI design rules specify a minimum separation between wires and therefore the area occupied by the routing on a chip is roughly proportional to the total wire length of the routing. Prior to the nanometer era, a fundamental objective in interconnect routing is to minimize the total wire length required to connect a prescribed set of terminals in the VLSI layout.
There are two types of net routing: single-net routing (also known as Steiner routing) and multi-net routing. This research is focused on single-net routing only. The single-net routing consists of (1) two-terminal net routing and (2) multi-terminal net routing.

2.1.1 Two-terminal net routing

The earliest automated routing algorithm is a maze search algorithm where the objective is to find a shortest path between two terminals. It is basically an extension of Moore's (Alpert et al. 2009) shortest path algorithm for a uniform grid structure. The algorithm operates on a two-terminal net, and a uniform grid graph, which can have some of its nodes specified as obstacles. It is guaranteed to find a path between the terminals of the net, and this path is guaranteed to be the shortest possible if it exists. The algorithm consists of two main phases. In the first phase, a wave search is expanded from source node to its neighbour nodes, which are then marked with label 1. Then, at every step $i$ (where $i > 1$), the unmarked neighbours of the nodes that were marked with label $L$ at step $i - 1$ are marked with label $L + 1$. This process continues until the wave reaches the target node. Once the target node is found, the shortest path construction is performed by backtracking to the source in the second phase of the algorithm.

Another algorithm for two-terminal routing is called line-search algorithm (Mikami and Tabuchi 1968). In line search algorithm, the routing space is modelled as a set of line segments instead of grid nodes. This feature reduces memory and runtime requirement compared to the maze routing algorithm which typically needs to allocate memory for each grid node.
2.1.2 Multi-terminal net routing and topology optimization

The objective of the multi-terminal routing is to connect all the terminals in VLSI layout such that the wire length is minimized. When all wires are point-to-point, with no intermediate junction other than the terminals, the optimum solution is called *minimum spanning tree* (MST). A classic algorithm for MST is Prim's algorithm (Prim 1957). If there are intermediate junctions between terminals, called Steiner points, the problem now becomes *Steiner minimal tree* (SMT) problem (Kahng and Robins 1995). The SMT problem also arises in non-VLSI applications, such as network communication (Hwang et al. 1992; Ivanov and Tuzhilin 1994). In VLSI, the SMT problem is focused to the rectilinear Steiner minimal tree (R-SMT) formulation, which reflects the Manhattan geometry of VLSI layout. Figure 2.1 illustrates the difference between R-MST (rectilinear minimal spanning tree) and R-SMT.

![Figure 2.1](image)

Figure 2.1 (a) Rectilinear minimum spanning tree (R-MST) (b) rectilinear Steiner minimal tree (R-SMT). Hollow dots indicate net terminals while solid dots are the Steiner nodes.

The concept of maze routing and line search algorithm can be extended into the multi-terminal nets routing. However, the problem of finding the optimal route for multi-terminal nets is an NP-complete problem. There are several heuristic-based algorithms that are used frequently in practice, for example (Hentschke et al. 2007) proposed the algorithm to route a multi-terminal net \( N \) which consists of two main steps. First, generate a Steiner topology \( T \) for terminals of net \( N \) and second, perform point-to-point routing between the terminals and Steiner nodes of topology \( T \). In this
technique, the quality of the solution depends on the topology constructed at the first stage (this idea is used in the proposed algorithm).

Another way to solve the R-SMT problem is to construct the R-MST first before the tree is refined to R-SMT. This is due to the fact that the quality of any R-MST is not worse than 3/2 of the optimal R-SMT (Huang 1976). The first algorithm of this refined R-MST was proposed by (Ho et al. 1989; 1990) called L-shaped Rectilinear Steiner Tree (L-RST). Their algorithm refines an initial R-MST by replacing the Euclidean edges in the initial R-MST with either an upper or a lower rectilinear L-shape. The goal of the algorithm is to maximize the overlap between the L-shapes and the existing tree.

The solution quality of the R-MST refinement technique is improved by (Kahng and Robins 1992; 1995). They proposed an algorithm called 1-Steiner algorithm. In 1-Steiner algorithm, given a set of points $P$ along with its R-MST denoted $R$-MST$(P)$, the algorithm seeks one additional Steiner point $s$ such that the wire length of $R$-MST$(P \cup s)$ is shorter than that of $R$-MST$(P)$. The Steiner point $s$ is called 1-Steiner point. The objective is to improve the total wire length by inserting the 1-Steiner point iteratively until no more wire length improvement is possible. It has been shown in (Kahng and Robins 1992) that the 1-Steiner algorithms provides a better quality solution compared to L-RST algorithm but at the cost of longer runtime. The runtime of the 1-Steiner algorithm was improved by (Borah et al. 1994) where their heuristic 1-Steiner algorithm produces results that are comparable to the 1-Steiner by Kahng and Robins but with complexity improvement. In their algorithm, the initial R-MST is improved by finding the shortest edge between a node in $P$ and any point along any MST edge. If the edge is inserted, a cycle is formed. Then, the removal of the longest edge on this cycle will result in wire length reduction of the R-MST.

The refinement R-MST algorithms above are essentially focused on wire length minimization. As researches in this area advances, the works on delay-oriented optimization has increasingly become more prevalent. An early batch of delay-oriented optimization algorithm is the BR-MRT (Bounded Radius – Minimum Routing Tree), which was based on Prim’s MST algorithm. The BR-MRT algorithm
seeks a rectilinear spanning tree with minimum wire length under a given radius bound. In this type of algorithm, the delay is measured by a so-called radius. The radius is defined as the longest source to sink path length among all sinks and reducing the radius translates to source-sink delay reduction. There are two BR-MRT algorithms proposed by (Cong et al. 1992) which are Bound Prim or BPRIM and Bounded Radius Bounded Cost or BRBC.

In BPRIM, the tree is grown and is controlled by the radius bound. Every time a node $y$ is added to the growing tree via an edge $(x, y)$, the algorithm makes sure the radius bound is not violated. Otherwise, another node $x'$ is chosen so that the radius bound is satisfied after inserting $(x', y)$. The radius bound is violated if $\text{dist}(s, x) + \text{dist}(x, y) > (1 + \varepsilon)R$ where $s$, $\varepsilon$ and $R$ are the source of the tree, user-specified constant and radius while $\text{dist}(s, x)$ denotes the path length between $s$ and $x$, and $\text{dist}(x, y)$ is the rectilinear distance between $x$ and $y$. Meanwhile, the BRBC algorithm constructs a spanning tree that has bounded wire length and diameter value. Total wire length is at most $(2 + 2/\varepsilon)$ times that of a minimum spanning tree and the diameter is at most $(1 + 2\varepsilon)$ times the diameter of the node set. The $\varepsilon$ parameter is used to trade off wire length and diameter. The BRBC algorithm yields a routing solution with radius smaller than $(1 + \varepsilon)R$ but at the expense of longer wire length compared to the BPRIM algorithm.

Another algorithm that is similar to BR-MRT algorithm is A-tree algorithm proposed by (Cong et al. 1993). The A-tree algorithm constructs an MRSA (Minimum Rectilinear Steiner Arborescence) using a sequence of moves that minimizes the overall wire length while maintaining the shortest linear path lengths for all sinks. Although wire length and radius give a considerable good solution for performance oriented Steiner routing, they are still an indirect metric to represent actual delay because their units are not in time domain.

One of the first algorithms that directly minimizes the popular Elmore delay metric was proposed by (Boese et al. 1995). Their algorithm resembles Prim’s MST algorithm and iteratively adds one node at a time to construct the tree. Given a candidate node $v$ to be evaluated, $v$ is added to the current tree and the Elmore delay at all sinks are computed. Then, the maximum Elmore delay among the sinks is
recorded. After examining all possible candidates, the node that results in the minimum Elmore delay is chosen as a final solution.

(Alpert et al. 2002) studied factors such as sink criticalities, non-uniform sink distribution and different polarity requirements that affect the final routing solution. They proposed a C-tree (Clustered tree) algorithm that takes all these factors into consideration to construct a Steiner tree. The main idea of the C-tree is to construct a tree in two stages. Firstly, sinks are clustered based on a distance metric. The timing metrics comprise of timing criticality, polarity requirement and physical distance from the source. In the second stage, the lower level trees are constructed for each cluster. After determining tapping points for each cluster, the top-level timing-driven tree is constructed, connecting the driver with cluster tapping points. Merging the top-level tree with cluster trees yields a final tree for the entire net. The C-tree algorithm offers a good trade-off between runtime and the quality of solutions. However, this algorithm is not designed to handle routing obstacles.

2.2 Post routing optimization with buffer insertion

After the best routing between terminals is obtained using one of the techniques discussed in section 2.1, the interconnect delay of the routing tree can be further optimized by inserting buffers along the routing tree. Buffers can reduce wire delay by restoring signal strength, in particular for a long wire. Moreover, buffer can be used to shield capacitive load from timing-critical paths such that the interconnect delay along the critical paths are reduced.

As the ratio of wire delay to gate delay increases from one technology to the next, more and more buffers are required to achieve performance goals. The technology scaling is studied by Intel and the results are reported in (Saxena et al. 2004). The technology scaling impacts the critical inter buffer length, which is the minimum distance between two buffers and the size of buffers that will give an optimal solution. Therefore, an algorithm that is capable of determining this critical inter buffer
length and buffer size is essential. There are two techniques for buffer insertion algorithm which are (1) closed-form solution and (2) dynamic programming.

2.2.1 Closed-form solution

The fundamentals of the closed-form solution was presented in (Bakoglu 1990). An un-buffered wire with length $L$ is shown in Figure 2.2(a) and its $\pi$-model RC circuit is shown in Figure 2.2(b). In Bakoglu’s model, the wire has one source and one sink node. Given $R_s$ is the source resistance, $C_L$ is the load capacitance, $c_w$ and $r_w$ are wire capacitance and resistance per unit length respectively.

\[ ED = R_s \left( c_w L + C_L \right) + r_w L \left( \frac{c_w L}{2} + C_L \right) \]  \hspace{1cm} (2.1)

![Diagram of a wire and its corresponding $\pi$-model RC circuit](image)

Figure 2.2 (a) A wire of length $L$ and (b) Corresponding $\pi$-model RC circuit

The Elmore delay at sink is given by

It is clear that the delay is increasing quadratically with length $L$. In order to reduce the delay, a buffer can be used to reconstruct the signal as it is propagated along a wire. The wire can be divided into $N$ segments of length $l$ and $N-$identical buffers can be inserted between each segment as shown in Figure 2.3 (in this example, $N = 2$).
In Figure 2.3, $r_b$, $c_b$ and $d_b$ are buffer output resistance, buffer input capacitance and buffer intrinsic delay respectively. For simplicity, one can assume that $R_s = r_b$, $C_L = c_b$ and $d_b = 0$. With the assumptions, the Elmore delay along the wire can be written as

$$ED = N \left[ r_b \left( c_b + c_w l \right) + r_w \left( \frac{c_b}{2} + \frac{c_w}{2} l \right) \right]$$

$$= L \left[ \frac{r_w c_w l}{2} + \left( r_w c_b + r_b c_w \right) + \frac{1}{l} \left( r_b c_b \right) \right]$$

where $L = NL$ \hspace{1cm} (2.2)

The optimum length, $l_{opt}$ (the length that gives $ED$ minimum) for each segment is obtained by differentiating $dED/dl$ and setting it to 0, and is given as follows

$$l_{opt} = \sqrt{\frac{2r_b c_b}{r_w c_w}}$$ \hspace{1cm} (2.3)

Another closed-form solution to determine the optimum distance between buffers for the two-terminal interconnects was proposed by [Alpert and Devgan 1997]. If the delay of a two-terminal net is to be minimized by using a single buffer type $b$, one needs to decide the number of buffers $k$, the spacing between the buffers $l$, the source and the sink. By using Kuhn-Tucker condition [Sinha 2006], they also found that the buffer need to be equally spaced with the specific size of buffer in order to minimized the delay.
2.2.2 Dynamic programming

Dynamic programming (DP) (Cormen et al. 2009) is essentially a divide-and-conquer method where a complex problem is solved by combining the solutions of the sub-problems. This technique can be summarized as follows; (1) break the problem into smaller sub-problems, (2) solve the smaller sub-problems optimally and (3) combine the optimal solutions of the sub-problems to get a solution to the original problem. The advantage of dynamic programming technique to find optimal buffer insertions over the closed-form solution is that it can be used to optimize multi-terminal nets and can handle multiple buffer types.

(van Ginneken 1990) was the first to utilize the dynamic programming technique in buffer insertion algorithm. The goal of the algorithm is to determine the best location of buffers on a given interconnect tree in order to optimize the Elmore delay. In this algorithm, a net is given as a binary routing tree $T = (V, E)$, where $V = \{s_0\} \cup v_i \cup v_n$ and $e \in E$ is a set of edges. Node $s_0$ is the source node and also the root of $T$, $v_i$ is a set of sink nodes and $v_n$ is a set of internal nodes. Each sink node $s \in v_s$ is associated with sink capacitance (load capacitance) $C_L$ and required arrival time (RAT). Each edge $e \in E$ is associated with lump resistance $r_e$ and capacitance $c_e$. The source resistance has a resistance of $R_s$ while input capacitance and output resistance of the buffer are $c_b$ and $r_b$ respectively. The timing buffering problem is defined as follows; given a routing tree $T = (V, E)$, find the best place to insert buffers along the tree (at any internal node $v_i$) such that the slack at source is maximized. Slack at any node $v$ is defined as

$$q_v = \text{RAT}(s) - D(v, s)$$

(2.4)

where RAT($s$) is the required arrival time at sink $s$ and $D(v, s)$ is the downstream delay from node $v$ to sink $s$. Each node is labelled with capacitance-slack, $(c, q)$ pair. We call this $(c, q)$ pair a candidate solution. To find the best slack at source, the algorithm iteratively computes the candidate solutions using the following operation; edge expansion, buffer insertion and branch merging. The process is repeated until the candidate solutions reach the source where the influence of the driver is added to find the optimum solution. Because van Ginneken’s algorithm is fundamental to all
dynamic programming based buffer insertion algorithms, hence, we will further discuss this algorithm in detail in Chapter 3.

The time complexity of van Ginneken’s algorithm is $O(n^2)$ where $n$ is the buffer candidate locations (complexity analysis will be discussed in Chapter 3). The algorithm was extended by John Lillis to handle multiple buffer types which runs in $O(n^2b^3)$ time where $b$ is the number of buffer types (Lillis et al. 1996). As the number of buffers inserted in the circuits increases dramatically, an algorithm that is fast and efficient is essential for the design automation tools. Recently, many techniques to speedup van Ginneken’s algorithm and its extensions were proposed. (Shi and Li 2005) improved the time complexity of van Ginneken’s algorithm to $O(b^2n\log n)$ for two-terminal nets buffer insertion and $O(b^2n\log^2 n)$ for multi-terminal nets buffer insertion. They also report the latest finding of an $O(mn)$ time algorithm for $m$-sink nets (Li et al. 2012). The speedup is achieved by the property explored in (Li and Shi 2006), a convex pruning which is a clever bookkeeping method and innovative linked lists that allow $O(1)$ update for adding an edge or a candidate. Their algorithm is called Fast Buffer Insertion (FBI) algorithm.

2.3 Simultaneous routing and buffer insertion

The algorithms discussed in section 2.2 are based on the post-routing buffer insertion technique (also known as two-step technique). The drawback of this technique is that the algorithms depend on the input routing tree (as discussed in section 1.2). They will give an optimal buffer solution when they are given an optimal routing tree but they will produce a poor solution when a poor tree is given. In the presence of obstacles, a simultaneous routing and buffer insertion algorithm seems to offer a better solution compared to the post routing buffer insertion. Next, we will discuss this type of algorithm for two-terminal net routing and multi-terminal net routing.
2.3.1 Two-terminal net

For two-terminal (one source and one sink) net, (Zhou et al. 1999; 2000) introduced graph-based dynamic programming for the routing and at the same time finding an optimum buffer solution. This algorithm utilized the maze search algorithm to find the best buffered path on the VLSI layout with consideration for buffer and wire obstacles. In Zhou’s algorithm, the VLSI layout is represented by a uniform grid graph where the source and sink vertices are specified. The path expansion begins from the sink node spreading to the entire graph until the candidate solutions reach the source node where the driver influence is added.

Expansion to speed-up this algorithm was proposed by (Khalil-Hani and Shaikh-Husin 2008; 2009). The speedup is achieved by incorporating a novel look-ahead technique into earlier graph-based algorithms. The algorithm is called S-RABILA, which refers to Simultaneous Routing and Buffer Insertion with Look-Ahead. This look-ahead scheme significantly improves the run-time of previous graph-based algorithms by guiding the path expansion process using the look-ahead weight vectors calculated by the algorithm. The look-ahead scheme will be discussed in detail in Chapter 3.

2.3.2 Multi-terminal net

Although S-RABILA is proven to be fast, however, it can only handle two-terminal nets which are not suitable for today’s design where multi-terminal nets are prevalent. To solve simultaneous routing and buffer insertion for multi-terminal net problem, Cong proposed an algorithm called RMP (Recursive Merging and Pruning) algorithm (Cong and Yuan 2000).

In the classic van Ginneken’s algorithm, when the solutions from each sink reach a common Steiner node, the branch merging operation is performed by adding the capacitance from each sink, and selecting the worst slack of the merging
candidates. Also, in van Ginneken's algorithm, the merging nodes (Steiner nodes) are known. However, in simultaneous routing and buffer insertion for a multi-terminal problem, a merging node could be anywhere and to determine the merging nodes (location on the graph) that will give an optimum solution is a hard problem. In RMP algorithm, the path search begins from each sink spreading to entire graph until all candidate solutions reach the source. The expansion is the same as in (Zhou et al. 2000) where the algorithm will keep updating the non-dominated slack and capacitance \((q, c)\) pairs (i.e. the candidates) from the sinks towards the source. The difference is that RMP considers merging solutions to form sub-trees. When the candidate solutions meet at a particular node, the branch merging operation is performed. This will create a new solution set at the merging node and these solutions will be propagated towards the source. To achieve an optimum solution, the algorithm allows sub-trees to be merged to a new sub-tree and keep the original separate sub-trees at the same time. However, it only allows a merge between two labels that have no common sink nodes in their reachable sets. The reason for this is to avoid duplicate connections. The RMP algorithm can find the optimal solution in exponential time and this is clearly too slow for large circuits.

Another technique to optimize the multi-terminal nets was proposed by (Tang et al. 2001) called RTBW (Routing Tree with Buffer insertion and Wire sizing). Tang used graph-based approach where the routing graph \(G = (V, E)\) was first transformed into a buffer graph \(G_B = (V_B, E_B)\) and then obtained the minimum delay buffered path using the Dijkstra's shortest path algorithm. The buffered paths are pre-calculated and stored in a look-up table, and the algorithm will search iteratively a possible solution in the look-up table until the optimal solution is obtained. The RTBW is extended by (Dechu et al. 2005) to speed up the computation time (several hundred times for net with 8 sinks to 25 sinks).
2.4 Tree adjustment technique

Although RMP and RTBW can find optimal buffering solution for multi-sink nets, they are still not efficient when the number of possible buffer locations is big as the search space is very large (almost the entire graph in RMP). Indeed, (Hu et al. 2003) showed that the searching in RMP and RTBW is NP-complete and they proved that the runtime of the two-step approach is significantly faster than the simultaneous approach with only a small degradation in solution quality. Hu also proposed an algorithm to solve multi-terminal nets by constructing a performance driven Steiner tree and creates an alternative Steiner node if the original Steiner node is inside an obstacle area. This adjustment is also a candidate solution that is propagated towards the source. The tree adjustment is adopted only when its corresponding candidate solution is selected at the source. The algorithm chooses the path for rerouting if the path has a large overlap with obstacles. The algorithm is called RIATA for Repeater Insertion with Adaptive Tree Adjustment. RIATA is very fast because it operates on a fixed tree. However, the quality of the solution may not be good enough if many paths of the adjusted tree still overlap with the obstacles.

2.5 Multi-constraint optimization techniques

Although timing optimization is crucial and available algorithms are capable of providing an optimal solution, however, other performance metrics are also very important in determining the overall performance of the VLSI circuits. Therefore, a buffer insertion algorithm should be able to handle more than one constraint. For example, recent designs are more prone to crosstalk because design geometry is reduced resulting in a much smaller distance between wires. A wire close to another one may induce noise to its neighbour resulting in an incorrect functional response (Elgamel and Bayoumi 2003; Ekekwe 2010; Caignet et al. 2001; Saxena et al. 2004). Alpert et al. were the first to propose a buffer insertion with noise avoidance algorithm that can fix noise violations while optimizing the delay (Alpert et al. 1999). A similar algorithm was proposed by (Zhang and Sapatnekar 2007) with an additional feature
that utilizes power/ground wires as a shield between signal wires to reduce capacitive coupling. Both Alpert and Zhang utilized Devgan’s coupling noise metric (Devgan 1997) to calculate the downstream noise. However, this noise metric is not a good estimation to the real crosstalk noise because it assumes that the interconnect wires are always in parallel.

As mentioned in Chapter 1, it was reported in (Saxena et al. 2004) that the percentage of block level nets requiring buffers grow from 5.8% in 90-nm technology to 19.6% in 45-nm technology. This shows that the number of buffers needed increased significantly when we move from one technology to another technology. Because a buffer itself consumes power, therefore power constraint should also be integrated into buffer insertion algorithm (Ekekwe 2010). In this thesis, we choose to optimize power simultaneously with timing performance as power dissipation of a buffer is inversely proportional to the interconnect delay. In other words, when we insert more buffers, the delay is reduced, however, the total power dissipation of the interconnect will increase due to power dissipation contributed by the inserted buffers.

For the closed-form solution, Nalamalpu and Burleson introduced a technique to minimize power and area while satisfying delay constraint when inserting buffers into the interconnect (Nalamalpu’and Burleson 2001). They proposed a mathematical equation to find the number of buffers and their sizes for minimizing area and power overhead while meeting a given delay target. Another closed-form solution to solve power optimization in buffer insertion algorithm was proposed by (Banerjee and Mehrotra 2002). The objective is to calculate the repeater size and interconnect length which minimizes the total interconnect power dissipation for any given delay penalty. Instead of only dynamic power as in Nalamalpu and Burleson technique, Banerjee and Mehrotra also include leakage power and short circuit power in their calculation. They derive three nonlinear equations with three unknowns, which are segment length \( l \), buffer size \( s \) and \( dl/ds \). The solution is then obtained numerically using Newton-Raphson method.

The available techniques that calculate power dissipation iteratively for dynamic programming buffer insertion algorithm were proposed by (Lillis et al. 1996)
REFERENCES


