INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or “target” for pages apparently lacking from the document photographed is “Missing Page(s)”. If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.

2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of “sectioning” the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.
AUTOMATIC HARDWARE COMPILER
FOR
THE CMOS GATE ARRAY
by
Jhy-Fang Hu

----------
A Thesis Submitted to the Faculty of the
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
In Partial Fulfillment of the Requirements
For the Degree of
MASTER OF SCIENCE
WITH A MAJOR IN ELECTRICAL ENGINEERING
In the Graduate College
THE UNIVERSITY OF ARIZONA
1986
STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED:

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

Fredrick J. Hill
Professor of Electrical Engineering
ACKNOWLEDGMENTS

Most of the credit for the research goes to Professor Fredrick J. Hill, to whom I am most indebted, not only for technical advice, but also for his patient throughout this research. Without his help, this research could not have been completed. Also, I would like to thank Dr. Jerzy Rozenblit and Dr. Zainalabedini Navabi for their valuable suggestion and guidance.

My final thanks goes to my parents. Their continual encouragement has been the most important contribution to the completion of my research.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF ILLUSTRATIONS</th>
<th>vi</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>ix</td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Statement of Problem</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Design Automation</td>
<td>6</td>
</tr>
<tr>
<td>1.3 Concepts for a Silicon Compiler</td>
<td>7</td>
</tr>
<tr>
<td>1.4 VLSI Design Methodology</td>
<td>10</td>
</tr>
<tr>
<td>1.4.1 Gate Array Design</td>
<td>10</td>
</tr>
<tr>
<td>1.4.2 Standard Cell</td>
<td>11</td>
</tr>
<tr>
<td>1.4.3 Custom Design</td>
<td>12</td>
</tr>
<tr>
<td>2. MACRO CELL LIBRARY FOR CMOS GATE ARRAYS</td>
<td>14</td>
</tr>
<tr>
<td>2.1 Basic Array Cell</td>
<td>14</td>
</tr>
<tr>
<td>2.2 Transmission Gate</td>
<td>17</td>
</tr>
<tr>
<td>2.3 CMOS Inverter</td>
<td>18</td>
</tr>
<tr>
<td>2.4 CMOS NOR Gate</td>
<td>19</td>
</tr>
<tr>
<td>2.5 CMOS NAND Gate</td>
<td>20</td>
</tr>
<tr>
<td>2.6 CMOS XOR Gate</td>
<td>21</td>
</tr>
<tr>
<td>2.7 CMOS XNOR Gate</td>
<td>22</td>
</tr>
<tr>
<td>2.8 CMOS D-Type Flip-Flop</td>
<td>23</td>
</tr>
<tr>
<td>2.9 CMOS D-Latch</td>
<td>25</td>
</tr>
<tr>
<td>2.10 Complex Gates and Other Macro Cell</td>
<td>27</td>
</tr>
<tr>
<td>3. LOGIC RECONFIGURATION</td>
<td>30</td>
</tr>
<tr>
<td>3.1 Fanin Consideration</td>
<td>30</td>
</tr>
<tr>
<td>3.2 Fanout Consideration</td>
<td>32</td>
</tr>
<tr>
<td>3.3 Handling Critical Path</td>
<td>34</td>
</tr>
<tr>
<td>3.4 Logic Optimization</td>
<td>36</td>
</tr>
<tr>
<td>4. CHIP PLAN</td>
<td>37</td>
</tr>
<tr>
<td>4.1 Estimating the Chip Size</td>
<td>38</td>
</tr>
<tr>
<td>4.2 Architecture of a Gate Array Chip</td>
<td>40</td>
</tr>
<tr>
<td>4.3 Color Graphic Output</td>
<td>42</td>
</tr>
<tr>
<td>4.4 Cell Placement</td>
<td>44</td>
</tr>
<tr>
<td>4.4.1 Defining a SCZ</td>
<td>46</td>
</tr>
<tr>
<td>TABLE OF CONTENTS—Continued</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>4.4.2 Computation of Connection</td>
<td>52</td>
</tr>
<tr>
<td>Force</td>
<td></td>
</tr>
<tr>
<td>4.4.3 Cell Assignment Strategy</td>
<td>54</td>
</tr>
<tr>
<td>4.5 Algorithm</td>
<td>60</td>
</tr>
<tr>
<td>5. A HYBRID APPROACH TO OPTIMAL CHANNEL ROUTING</td>
<td>61</td>
</tr>
<tr>
<td>5.1 Background</td>
<td>62</td>
</tr>
<tr>
<td>5.2 Classifying Routing Segments</td>
<td>65</td>
</tr>
<tr>
<td>5.3 Merging Segments Sharing a Common</td>
<td></td>
</tr>
<tr>
<td>Branching Node</td>
<td>69</td>
</tr>
<tr>
<td>5.4 HE Algorithm for Track Assignment</td>
<td>72</td>
</tr>
<tr>
<td>5.5 Best-First Search with Backtracking</td>
<td>74</td>
</tr>
<tr>
<td>5.5.1 Data Structure</td>
<td>76</td>
</tr>
<tr>
<td>5.5.2 Constraint Expert</td>
<td>77</td>
</tr>
<tr>
<td>5.5.3 Weight Function</td>
<td>81</td>
</tr>
<tr>
<td>5.5.4 Algorithm</td>
<td>85</td>
</tr>
<tr>
<td>5.6 CAD Interactive Channel Routing</td>
<td>87</td>
</tr>
<tr>
<td>6. EVALUATION OF THE RESULTS</td>
<td>91</td>
</tr>
<tr>
<td>6.1 Evaluating a Layout System</td>
<td>91</td>
</tr>
<tr>
<td>6.2 Systematic Difference</td>
<td>93</td>
</tr>
<tr>
<td>6.3 Examples</td>
<td>96</td>
</tr>
<tr>
<td>6.3.1 4-bit shift register</td>
<td>97</td>
</tr>
<tr>
<td>6.3.2 Operation test</td>
<td>101</td>
</tr>
<tr>
<td>6.3.3 Detector</td>
<td>104</td>
</tr>
<tr>
<td>6.3.4 Sequential selector</td>
<td>107</td>
</tr>
<tr>
<td>6.4 Results</td>
<td>111</td>
</tr>
<tr>
<td>7. SUGGESTION AND CONCLUSIONS</td>
<td>112</td>
</tr>
<tr>
<td>APPENDIX A</td>
<td>118</td>
</tr>
<tr>
<td>APPENDIX B</td>
<td>120</td>
</tr>
<tr>
<td>APPENDIX C</td>
<td>121</td>
</tr>
<tr>
<td>APPENDIX D</td>
<td>122</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>123</td>
</tr>
</tbody>
</table>
LIST OF ILLUSTRATIONS

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>AHPL-Based VLSI design environment</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>Automatic CMOS gate array compiler</td>
<td>4</td>
</tr>
<tr>
<td>1.3</td>
<td>Organization of silicon compiler</td>
<td>9</td>
</tr>
<tr>
<td>1.4</td>
<td>Typical chip plan of semicustom design</td>
<td>13</td>
</tr>
<tr>
<td>2.1</td>
<td>Structure of basic array cell</td>
<td>16</td>
</tr>
<tr>
<td>2.2</td>
<td>CMOS transmission gate</td>
<td>17</td>
</tr>
<tr>
<td>2.3</td>
<td>CMOS inverter</td>
<td>18</td>
</tr>
<tr>
<td>2.4</td>
<td>CMOS NOR gate</td>
<td>19</td>
</tr>
<tr>
<td>2.5</td>
<td>CMOS NAND gate</td>
<td>20</td>
</tr>
<tr>
<td>2.6</td>
<td>CMOS XOR gate</td>
<td>21</td>
</tr>
<tr>
<td>2.7</td>
<td>CMOS XNOR gate</td>
<td>22</td>
</tr>
<tr>
<td>2.8</td>
<td>D flip-flop with external sets</td>
<td>23</td>
</tr>
<tr>
<td>2.9</td>
<td>Simple D flip-flop</td>
<td>24</td>
</tr>
<tr>
<td>2.10</td>
<td>Simple D latch</td>
<td>25</td>
</tr>
<tr>
<td>2.11</td>
<td>D latch with external sets</td>
<td>26</td>
</tr>
<tr>
<td>2.12</td>
<td>CMOS 3-input NOR</td>
<td>27</td>
</tr>
<tr>
<td>2.13</td>
<td>CMOS 4-input NOR</td>
<td>28</td>
</tr>
<tr>
<td>2.14</td>
<td>CMOS 3-input NAND</td>
<td>28</td>
</tr>
<tr>
<td>2.15</td>
<td>CMOS 4-input NANA</td>
<td>29</td>
</tr>
<tr>
<td>2.16</td>
<td>Parallel NOT gate</td>
<td>29</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>3.1</td>
<td>Conversion models for fan-in consideration</td>
<td>31</td>
</tr>
<tr>
<td>3.2</td>
<td>One level fan-out consideration</td>
<td>33</td>
</tr>
<tr>
<td>3.3</td>
<td>Two-level fan-out consideration</td>
<td>33</td>
</tr>
<tr>
<td>3.4</td>
<td>Parallel cells used in critical path</td>
<td>35</td>
</tr>
<tr>
<td>3.5</td>
<td>Optimization of XNOR gate</td>
<td>36</td>
</tr>
<tr>
<td>4.1</td>
<td>Architecture of a gate array chip</td>
<td>41</td>
</tr>
<tr>
<td>4.2</td>
<td>Zoom-out capacity for graphic layout</td>
<td>43</td>
</tr>
<tr>
<td>4.3</td>
<td>Possible chip plans for a 4-element circuit</td>
<td>45</td>
</tr>
<tr>
<td>4.4</td>
<td>Defining a &quot;Strongly Connected Zone&quot; (SCZ)</td>
<td>48</td>
</tr>
<tr>
<td>4.5</td>
<td>CPU time for SCZ method</td>
<td>51</td>
</tr>
<tr>
<td>4.6</td>
<td>Memory usage for SCZ method</td>
<td>51</td>
</tr>
<tr>
<td>4.7</td>
<td>Computation of connection force</td>
<td>53</td>
</tr>
<tr>
<td>4.8</td>
<td>Force-Directed chip plan</td>
<td>55</td>
</tr>
<tr>
<td>4.9</td>
<td>Pseudo-Continuous (PC) cell assignment</td>
<td>57</td>
</tr>
<tr>
<td>4.10</td>
<td>Comparison between PC and traditional method</td>
<td>57</td>
</tr>
<tr>
<td>4.11</td>
<td>Routing decentralization</td>
<td>59</td>
</tr>
<tr>
<td>5.1</td>
<td>Lee's maze router</td>
<td>63</td>
</tr>
<tr>
<td>5.2</td>
<td>First type routing segments</td>
<td>65</td>
</tr>
<tr>
<td>5.3</td>
<td>Second type routing segment</td>
<td>66</td>
</tr>
<tr>
<td>5.4</td>
<td>Splitting a second type segment</td>
<td>68</td>
</tr>
<tr>
<td>5.5</td>
<td>Merge of segments</td>
<td>69</td>
</tr>
<tr>
<td>5.6</td>
<td>Routing optimization by merging segments</td>
<td>71</td>
</tr>
<tr>
<td>5.7</td>
<td>HE (Horizontal End) algorithm</td>
<td>73</td>
</tr>
</tbody>
</table>
### LIST OF ILLUSTRATIONS — Continued

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.8</td>
<td>Best-First with back tracking (flow-chart)</td>
<td>75</td>
</tr>
<tr>
<td>5.9</td>
<td>Illegal trespass on cell rows</td>
<td>77</td>
</tr>
<tr>
<td>5.10</td>
<td>Routing layer constraint</td>
<td>78</td>
</tr>
<tr>
<td>5.11</td>
<td>Dead-End constraint</td>
<td>79</td>
</tr>
<tr>
<td>5.12</td>
<td>Chip dimension constraint</td>
<td>80</td>
</tr>
<tr>
<td>5.13</td>
<td>Best-first by shortest distance</td>
<td>82</td>
</tr>
<tr>
<td>5.14</td>
<td>Routings between two terminals</td>
<td>84</td>
</tr>
<tr>
<td>5.15</td>
<td>Best-First with back tracking (example)</td>
<td>86</td>
</tr>
<tr>
<td>5.16</td>
<td>Specifying X-Y coordinates on a gate array chip</td>
<td>89</td>
</tr>
<tr>
<td>5.17</td>
<td>CAD interactive routing (flow-chart)</td>
<td>90</td>
</tr>
<tr>
<td>6.1</td>
<td>SHIFTR by HU's compiler</td>
<td>99</td>
</tr>
<tr>
<td>6.2</td>
<td>SHIFTR by CHEN's compiler</td>
<td>100</td>
</tr>
<tr>
<td>6.3</td>
<td>OPERATION by HU's compiler</td>
<td>102</td>
</tr>
<tr>
<td>6.4</td>
<td>OPERATION by CHEN's compiler</td>
<td>103</td>
</tr>
<tr>
<td>6.5</td>
<td>DETECT by HU's compiler</td>
<td>105</td>
</tr>
<tr>
<td>6.6</td>
<td>DETECT by CHEN's compiler</td>
<td>106</td>
</tr>
<tr>
<td>6.7</td>
<td>SEQSEL by HU's compiler</td>
<td>109</td>
</tr>
<tr>
<td>6.8</td>
<td>SEQSEL by CHEN's compiler</td>
<td>110</td>
</tr>
<tr>
<td>7.1</td>
<td>Dynamic cell placement</td>
<td>114</td>
</tr>
<tr>
<td>7.2</td>
<td>Routing improvement</td>
<td>115</td>
</tr>
</tbody>
</table>
ABSTRACT

Due to the trend in complexity of Integrated Circuits (IC), the silicon compilation has become the inevitable solution to today's VLSI (Very Large Scale Integrated circuits) design.

This thesis introduces a new hardware compiler which automatically compiles the AHPL (A Hardware Programming Language) [1] into the CMOS gate array chip. The automation of the compiler includes the chip estimation, chip plan, and channel routing. The flexible design of the CMOS compiler allows the user to define his own chip dimension. Furthermore, the capability of interactive routing is also provided in this CMOS compiler. Inside the compiler, the MBPC (Module-Based Pseudo Continuous) cell placement method and the hybrid global router with AI (Artificial Intelligence) search techniques are installed to give satisfied solution.
CHAPTER ONE

INTRODUCTION

Silicon Compilation appears to be an inevitable solution to today's VLSI (Very Large Scale Integrated Circuit) design crisis, and the dream toward true silicon compilation will come true in the very near future.

1.1 Statement of Problem

The objective of this thesis is to implement an automatic hardware compiler to compile the AHPL hardware description language into a CMOS (Complementary Metal Oxide Silicon) gate array chip. AHPL was first developed by Dr. Fredrick J. Hill at the University of Arizona in 1970's [1]. The AHPL is now well-implemented for multi-purpose VLSI design and simulation [2-5]. Figure 1.1 describes part of the AHPL-based VLSI design environment.
Figure 1.1 AHPL-based VLSI design environment
The detailed implementation of this compiler is shown in Figure 1.2. As indicated in the Figure 1.2, the introduced hardware compiler consists of several main parts: Logic Conversion, Cell Library, Chip Plan (including Chip Size, Module Formation, and Cell Assignment), Channel Routing (including heuristic and interactive routing), and Graphic Chip Layout. These principal parts of the CMOS compiler will be discussed in the following chapters.

The input of the CMOS gate array compiler is the stage01 output of the AHPL 3-stage hardware compiler [6]. The output of the implemented compiler is a color graphic chip layout on the TEKTRONIX 4105 color graphic terminal. The double metal (Aluminum) routing model is used to improve the circuit switching characteristics with respect to the single metal family. All information including SCZ partition, chip plan and cell efficiency are saved in file CHECK.DAT for checking purpose. The channel routing information is saved in files FLAT.TXT (saving horizontal segments) and VERT.TXT (saving vertical segments).
Figure 1.2 Automatic CMOS gate array compiler
The strong points of this approach to the CMOS gate array compiler include:

1. Module based cell placement

   A cell assignment method named "Module-Based Pseudo-Continuous Placement" (MBPC) is introduced in this thesis. By using the MBPC method, drawbacks, such as low cell efficiency and difficult complex routing caused by inadequate conventional cell placement methods [2] (always one-way assignment of cell rows from left to right) can be reduced.

2. Strongly Connected Zone (SCZ)

   Since the "Force-Directed" method is applied to reach the best chip plan, the connection force between any two components has to be calculated very carefully. As we know, a gate array chip may include thousands of components. For this reason, the computation of connection force is complicated. The concept of "Strongly Connected Zone" (SCZ) is introduced to relieve the burden of computation. In addition, the SCZs are also used as the basis of circuit partition in this thesis.

3. Hybrid approach to optimal routing layout

   A hybrid router including algorithmic, interactive, and heuristic methods is implemented to most effectively use the routing capacity. Under this method, the dream of optimal compilation (including 100 percent routability,
minimum wire length, and minimum vias used) becomes possible.

1.2 Design Automation

Why do we need an automatic hardware compiler? The main reason is the trend in complexity of Integrated Circuits (IC). Since Small Scale Integrated (SSI) circuits were introduced in early 1960s, the effort to minimize circuit size on silicon chip has continued unabatedly. The total number of transistors on a single silicon chip has increased from less than 100 transistors in SSI to more than 30,000 transistors in today's VLSI circuit. The compaction of the integrated circuits on the silicon chip will continue to increase in the future. A simple comparison between different level of integration for MOS circuits is shown in Table 1.1.

Table 1.1 Trend in IC complexity

<table>
<thead>
<tr>
<th>Transistors</th>
<th>Integration Level</th>
<th>Introduction Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 100</td>
<td>SSI</td>
<td>1960 - 1970</td>
</tr>
<tr>
<td>&lt; 3000</td>
<td>MSI</td>
<td>1970 - 1975</td>
</tr>
<tr>
<td>&lt; 30,000</td>
<td>LSI</td>
<td>1975 - 1980</td>
</tr>
<tr>
<td>&gt; 30,000</td>
<td>VLSI</td>
<td>after 1980</td>
</tr>
</tbody>
</table>
Due to the complexity in the VLSI circuits, even a small VLSI chip may take several man-years and millions of dollars to complete by using conventional manual design (for example, checking massive plots by eye for design rule violations). On the other hand, by using the Computer-Aided Design (CAD) method, the design time needed to create the same chip is dramatically reduced. Furthermore, the physical size and reliability of the VLSI chip are also improved. In other words, CAD design tools have become immediate necessary for today's VLSI design. Many articles that [7-13] focus on CAD hardware design have been published in the last decade.

1.3 Concepts for a Silicon Compiler

A silicon compiler is an optimizing software system which automatically generates mask-level descriptions of VLSI chips from a high level input language description. According to the input language description, silicon compilation can be divided into two categories: structural and functional compilation. In the structural approach, a circuit is described by specifying its circuit components and interconnection in a hierarchical manner. One example of a structural language is VHDL [14]. In the functional approach, a circuit can be specified by its architecture or by circuit behavior. The behavior silicon compilers are similar to the automatic programming systems that have been
studied by Artificial Intelligence (AI) researchers [15-16]. In order to help understanding the basic idea of a silicon compiler, the general organization of silicon compilation is shown in Figure 1.3.
Figure 1.3 Organization of silicon compiler
1.4 VLSI Design Methodology

Current design approach to VLSI circuits can be divided into three main categories: gate array, standard cell, and custom design. The gate array and standard cell are sometimes defined as semicustom to distinguish from custom design.

1.4.1 Gate Array Design

Gate arrays are late-mask-programmable devices which contain a predefined number of uncommitted transistors for interconnection. Any gates and logic functions can be implemented by routing different number of uncommitted transistors. The gate array design is the only approach that allows designers to change or add logic at the last moment with little or no penalty.

In comparison to full custom design, gate array design offers many advantages. Some of these advantages are:

1. Fast design turnaround:

Because few masks (depending on how many wiring layers are used) for interconnection are needed to be generated, a gate array chip can be integrated quickly in comparison to a full custom design which may require 8 or more masks to be designed, debugged, and produced. The elimination of the need to produce a complete mask set reduces the development time substantially.
2. Low design cost:

Since no manual layout phase is needed and the fabrication of the interconnection layers is a low-risk and inexpensive operation, the design cost for a gate array chip is 10 percent (or more) lower than the full custom design.

3. High reliability and little space required:

Due to the regular structure of gate array chips, more CAD tools have been developed as tools to support gate array design. These CAD tools make modern gate array design straightforward, reliable, and efficient in space.

4. High design flexibility:

Unlike the full custom refinements involving full mask set, the redesign of a gate array chip involves only personalized interconnection layers. This characteristic allows the gate array products to be updated with very little production changes or loss of marketing continuity.

According to the DATAQUEST report [17], "In 1985, the worldwide IC market was down 18 percent from 1984, while the dollar value of worldwide gate array shipments rose an amazing 45.5 percent," the gate array design is enjoying its widespread popularity.

1.4.2 Standard cell

The major difference between gate arrays and standard cells is that gate arrays are prefabricated with uncommitted transistors, while standard cells are
fabricated anew for each design. In comparison to gate array chips, predefined cells used in the standard cell design can be dropped anywhere in chip area. With a well-designed cell library, the density of standard cell design can be highly increased. As shown in Figure 1.4, the chip plan of standard cells is more "personalized" than for gate arrays'.

1.4.3 Custom design

Custom design implies that little or no structure is used in easing the task of chip design. For VLSI circuits, custom design uses hierarchical design by interconnecting small custom-defined cell into larger cells, and so on until the entire chip is laid out. Cells are not constrained to lie in rows as designed in a gate array chip but can be located anywhere on the chip to minimize wasted space. Although the custom design is the most time-consuming design method, it produces the most area-efficient chips and usually the fastest circuits. The trade-off is between design time and production yield.
Figure 1.4  Typical chip plan of semicustom design
CHAPTER TWO

MACRO CELL LIBRARY FOR CMOS GATE ARRAYS

A well predefined cell library allows the gate array designer to design with basic logic functions and releases him from the burden of designing VLSI circuits at the transistor level. These predefined macrocells can be put together to construct function blocks or software blocks, such as counters or shift registers, to reduce the time required for logic conversion and logic simulation. The macrocells defined in this thesis are invented by the GOULD AMI Semiconductors [37].

2.1 Basic Array Cell

The basic array cell is the smallest unit forming a cell row in the gate array chip. It contains a predetermined number of uncommitted transistor pairs, which are used to build logic gates and flip-flops by selecting the wiring pattern for the required function.

Different technology or design methodology may affect the configuration of a basic array cell. A basic array cell can be formed by different pairs of transistors based on design convenience or technology available. For example, the basic cell used in Motorola's HCA 6000 series design [38] is shown in Figure 2.1(a) is formed by eight N-
channel and eight P-channel non dedicated transistors which shares common sources and drains. Running through the basic cell are the power buses VDD and VSS.

Another basic cell shown in Figure 2.1(b) is the one used in this thesis. It contains only 2 transistor pairs, and has been designed for high component usage as well as highly efficient automated placement and routing of the macros. As shown in the figure, a feedthrough path is provided for the channel-to-channel connection.
(A) Motorola HCA 6000 series

(B) Another basic cell

Figure 2.1 Structure of basic array cell
2.2 Transmission Gate

Unique to MOS technology, the transmission gate acts as a two-way switch. It is widely used in flip-flops, multiplexers and Exclusive OR/NOR configurations. As shown in Figure 2.2, two transistors are required to implement the function. The transmission gate may be controlled by a clock or enable signals.

Figure 2.2 CMOS transmission gate
2.3 CMOS Inverter

The CMOS inverter as shown in Figure 2.3 is the most basic element in CMOS technology. It consists of two MOS transistors (one P-channel and one N-channel) connected in series. The source of the PMOS transistor is tied to VDD (logic "1") and the source of the NMOS transistor is connected to VSS (logic "0").

![CMOS Inverter Diagram]

**Figure 2.3 CMOS Inverter**
2.4 CMOS NOR Gate

As shown in Figure 2.4, each input of CMOS NOR gate requires a pair of complementary transistors. The PMOS transistors are connected in series and the NMOS transistors are connected in parallel with common source tied to VSS. The output of a CMOS NOR gate is high "1" only when all the inputs are low.

![CMOS NOR gate diagram]

Figure 2.4 CMOS NOR gate
Figure 2.5 presents the logic symbol, circuit implementation, and truth table for a 2-input CMOS NAND gate. In general two complimentary transistors are required to build a CMOS NAND gate. A multi-input CMOS NAND gate can be expanded by adding more PMOS transistors in parallel and more NMOS transistors in series. As the fan-in of a gate increased, however, the propagation delay will be affected by the increase of the parasitic capacitance.
2.6 CMOS XOR Gates

The XOR is known as the non-coincidence gate since it detects inputs with complementary states. It is widely used in building adders, parity checkers and ALUs. There are different configurations for performing the XOR function. Figure 2.6 presents one of them.

![CMOS XOR gate diagram]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 2.6 CMOS XOR gate
2.7 CMOS XNOR Gates

The Exclusive NOR gate is sometimes referred to as a logic comparator since its output is high only when both inputs have the same logic level. Figure 2.7 illustrates the XNOR configuration we used in the project.

Figure 2.7 CMOS XNOR gate
2.8 CMOS D-Type Flip-Flop

The D flip-flop is the most commonly used storage element in CMOS technology. In fact, many other types of flip-flops, such as JK flip-flop and SR flip-flop, may be built with D flip-flop and external gates. Figure 2.8 represents the logic symbol, circuit schematic, and truth table for the D-type flip-flop with asynchronous set and reset. Figure 2.9 illustrates simple D-type flip-flop without external set and reset.

![Diagram of D flip-flop with external sets]
Figure 2.9 D flip-flop without external sets
2.9 CMOS D-Latch

Another type of storage element is the clocked latch. As long as the control signal G is "1" (logic high) and \(^G\) is "0" (logic low), the D-Latch will change state in response to any change in the input port. Figure 2.10 gives the circuit connection and truth table for the simple D-Latch without external set and reset input. Preset and clear function may be added to the simple D-Latch as shown in Figure 2.11.

![D latch without external sets](image)

<table>
<thead>
<tr>
<th>G</th>
<th>(^G)</th>
<th>D</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td>Q</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>

Figure 2.10 D latch without external sets
Figure 2.11 D latch with external sets
2.10 Complex Gates and Other Macro Cell

Complex gates are the ones which have more than one level of logic. Implementing complex gates reduces the number of gates and increases speed. In this section all macros used are introduced by their circuit connection and logic diagram.

Figure 2.12 CMOS 3-input NOR
Figure 2.13 CMOS 4-input NOR

Figure 2.14 CMOS 3-input NAND
Figure 2.15 CMOS 4-input NAND

Figure 2.16 Parallel NOT gate
CHAPTER THREE

LOGIC RECONFIGURATION

In gate array design, the most efficient silicon compiling is typically first achieved by dividing larger functional blocks into more primitive components, namely gates and flip-flops. In other words, the full circuit function has to be described by logic gates or flip-flops, so the user can easily pick up the best fit cell from the predefined cell library. The logic reconfiguration is regarded as the preprocessing of the gate array design, and a good logic reconfiguration will ease the future design processing.

In the following sections, some important facts are explained, which have to be carefully considered during the logic reconfiguration.

3.1 Fanin Consideration

In MOS technology, as the number of inputs increases, the number of transistors in series increases, and the "ON" resistance between the output and VDD or VSS is also increased. This fact forces the gate to require more time to charge or discharge the increased load capacitance. In other words, the performance of a CMOS gate is degraded
as its fan-in increases. Usually, logic gates with more than five inputs should be implemented in two or more levels of logic by using basic transformation techniques derived from the theorems of Boolean algebra. The conversion models used in this thesis are shown in Figure 3.1.

![Conversion models for fan-in consideration](image-url)

Figure 3.1 Conversion models for fan-in consideration
3.2 Fanout Consideration

As fanout increases in a CMOS gate, the load capacitance is increased. The rise and fall time increases almost linearly with increasing load capacitance. Under this situation, some restrictions have to be applied on the driving gate whose unit loads exceeds the maximum number of fanout capability.

In this thesis, if a CMOS gate with heavy fanout is detected, identical driving gate should be generated to minimize the propagation delay of the required signals. As shown in Figure 3.2, one possible treatment for the driving NAND gate is to duplicate another NAND gate to increase the driving capacity so that the propagation delay can be reduced. In Figure 3.3, two level duplication is needed. The important thing is that all duplicated gates must be the same type as the original one, and their inputs must originate at the same location to assure proper operation.
Figure 3.2 One level fan-out consideration

Figure 3.3 Two level fan-out consideration
3.3 Handling Critical Path

Handling critical path is one of the most critical step of the gate array design process. The propagation delay on a critical path may determine the first success of a gate array design.

In the current AHPL-based VLSI gate array design the propagation delay is not well-considered. In order to handle this problem, the AHPL user is suggested to run the circuit simulation program to address the critical paths before he starts the stage23 gate array compiler. Since the CAD interactive routing is possible in our compiler, the AHPL user is allowed to route the critical paths manually against the system's timing requirement.

Generally, in order to get the best result, a critical path is usually routed as below:

1. Routing the critical path on the same layer.
2. Minimizing the length of the critical path.
3. Using parallel cells [38] for the elements lying on the critical path. By using the parallel cells, the propagation time of the path can be highly improved. As shown in Figure 3.4, the propagation delay of path TW is reduced since the parallel NOT gates are used.
Figure 3.4 Parallel cells used in critical path
3.4 Logic Optimization

During development of the compiler, it was observed that AHPL description has no direct symbol to represent the XNOR gate. An alternative combining NOT (\(^{\neg}\)) and XOR (\(\oplus\)) has been used to obtain XNOR function in the AHPL based design. Let us examine the gate implementation shown in Figure 3.5 and compare the cell used for each case. Referring to chapter two, we know one basic cell is wasted in the combinational type.

Figure 3.5 Optimization of XNOR gate
CHAPTER FOUR

CHIP PLAN

The chip plan or floor plan is a plan of how the chip or die will be organized. It is very important to develop a good chip plan because an inferior chip plan may cause the chip area to increase undesirably and complicate the wire routing. Generally, an IC chip is organized based on the routing of critical lines, the number of wiring segments, the logic size, the power buses, and the bonding pads. In this chapter, three chip plan units are used:

1. Basic Cell:
   A basic cell is an array basic cell (see Figure 2.1) used to form a macrocell.

2. Macrocell:
   A macrocell consists of some basic cells. Each macrocell in this thesis represents a logic function gate, such as AND gate, OR gate, XOR gate, etc.

3. SCZ (Strongly-Connected Zone):
   An SCZ includes different number of macrocells. The author has used the SCZ to represent a circuit partition in this thesis.
4.1 Estimating the Chip Size

Prior to generating the floor plan, the system must know how big the chip will be. The number of logic gates and flip-flops in the circuit is the main factor which determines the size of the gate array chip. Since the logic is in the macrocell form, it is easy to determine how many cells will be required to implement the circuit.

The automatic hardware compiler designed in this thesis automatically computes the cell count for the AHPL designers. This system-computed cell count will give the designer the first approximation of what chip to select for his design.

Because the gate array chip is a cell structure, an area penalty results from assigning the uncommitted cells to logic gates. Furthermore, one blank cell is reserved between adjacent macros to ease the routing problem. In order to assure the sufficient cells have been provided by the designer, a predefined constant named "Area Penalty Coefficient" (APC) has been designed to handle this problem. Thus the system cell count can be explained by the following formula:

\[
\text{System Cell Count} = \left(\text{Actual Cell Used}\right) \times (\text{APC})
\]
The value of APC is preassumed as 1.18 (APC=1.18) and may be changed according to different technology used. In this thesis the chip dimension is determined by the following parameters:

1. Cell row:
   The number of cell rows is defined as total cell rows used in a gate array chip.
2. Cell column:
   The number of cell columns is defined as total basic cells included in a cell row.
3. Channel width:
   The channel width is defined as the number of routing channels between any two adjacent cell rows.
4. I/O pads:
   The I/O pads are defined as the number of bonding pads used to attach a wire from the chip to the package leads.

These four parameters are fully determined by the AHPL user. After the system cell count is prompted on the screen, the AHPL designer may use different combinations of above parameters to implement his circuit. Actually, the AHPL designer would have to consider the availability of chip sizes in the IC market while defining these parameters.

All user-provided parameters used to define the dimension of the gate array chip will be automatically examined by the system. Insufficient cells or bonding pads
will be detected by the system, and the AHPL designer will redefine these parameters in order to continue the processing.

4.2 Architecture of a Gate Array Chip

There are three basic elements that make up a gate array die: the cell array, the channel routing area, and the periphery. The cell array is constructed by predefined number of basic cells (see Chapter 2) for customizing any logic function needed for user's circuit. The channel routing area provides interconnection between cell arrays. The routing area and the cell array occupy the majority of the chip area, and provide most of the signal processing capabilities on the chip. Around a gate array chip, peripheral cells are filled to provide interface with external components and the power buses which distribute the supply voltages, VDD and VSS, through the circuit. The corners of a gate array chip are occupied by test devices, such as alignment keys. (The alignment key is a process control technique used in mask fabrication.) Figure 4.1 shows the typical organization of a 3-micro double metal gate array.
Figure 4.1 Architecture of a gate array chip
4.3 Color Graphic Output

A color graphic display of the chip layout including bonding pads, I/O cells, cell arrays, and channel routing between cellmacros serves as the final output of the gate array design. In the color graphic layout, bonding pads are filled around the user-defined chip and are indicated by blue color. The external I/O cells are assigned to the first row (external input) and the last row (external output) of the chip, which are indicated by green color with yellow boundary. All macrocells in the same SCZ are marked by identical blue or pink color. With the color aid and the checking file provided by the system, the AHPL user can identify any macrocell used in the circuit. The green squares in the cell array (or between cellmacros) represent an unused cell which is used to ease the routing task.

For the wire connection, the red color is used to indicate the horizontal segments routed on the first metal layer. The vertical wiring segments routed on the second metal layer are shown in green color. Programmable via are opened at each intersection of the vertical and horizontal wiring segments.

In order to check the wire connection, simple zoom out capability is provided in this compiler. By using this function, the AHPL user can amplify any part of the layout. As shown in Figure 4.2, the AHPL user is allowed to zoom out any one of eight regions for checking purpose.
The color graphic layout is implemented on the TEKTRONIX 4105 color graphic terminal. The terminal driver (PLOT.FOR) is implemented as part of this hardware compiler. Almost all TEK-4100 series commands are implemented in this compiler. Detailed information for programming on the TEK terminal is introduced in the TEK-4100 series manual [18].

Figure 4.2 Zoom-out capacity for graphic layout
4.4 Cell Placement

The cell placement algorithm performs the critical role in formation of the floor plan. A good cell placement will ease the difficulty of routing. One way to perform this is performed is to partition the circuit into small modules consisting of several gates each, and then arrange these modules on the gate array chip.

Let us consider the complexity of determining the optimal chip configuration. Suppose a circuit including four components is designed, as shown in Figure 4.3, twenty-four possible chip configurations are obtained for cell assignment. In other words, if an N component circuit is designed, the number of possible configurations is N!. In VLSI circuits, the value of N is always no less than 1000. Based on this fact, it is unwise for us to determine the optimal solution by comparing all possible chip configurations.

In this section, a heuristic placement based on the connection force and "Strongly Connected Zone" (SCZ) is introduced to obtain near-optimum solutions in a reasonable computer time. The whole placement task is divided into three parts:

1. Defining SCZs

2. Computation of connection force

3. Assignment of cell allocation
Figure 4.3 Possible chip plans of 4-element circuit
4.4.1 Defining a SCZ

The intent of this step is to partition the whole circuit into various modules to ease the computation of connection force between gates. As a matter of fact, the "Strong Connection Zone" (SCZ) is merely a circuit block including some number of function gates. The most important task in this subsection is to define the size of a SCZ.

How large should an SCZ be? Too large SCZs will not actually simplify the computation of connection force within a SCZ. On the other hand, too small SCZs make the force computation trivial, and do not relieve the burden of the whole cell placement task as expected. Currently, there is no algorithm which can be used to determine the optimal size of an SCZ.

Let us go back to examine the characteristics of AHPL based design. An AHPL program may be simply divided into control section and data section. The data section can be further divided into some individual groups of memory registers. Based on these observations, the best size for a SCZ will be individual n-bit registers including source control logic. In the current CMOS gate array compiler, an SCZ construction program has been implemented to build SCZs automatically. In the SCZ construction program, the control section of each AHPL module is always defined as an individual SCZ. Each memory block (n-bit register) with source control logic is defined as another SCZ. A simple
example is shown in Figure 4.4 to explain how the SCZ is determined in this thesis. The SCZ partition in the current system can be explained as following algorithm:

**ALGORITHM: SCZ PARTITION**

1. Collect all circuit components in the control section as the first SCZ;

2. Form a queue (Q) consisting of all memory blocks declared in the AHPL program;

3. **DO WHILE FAIL(EMPTY(M = FIRST(Q)))**
   Form an SCZ including M and its source control logic;
   Q = Q - M;
   **END DO**;

4. Form another SCZ including all logic gates left from above processing;

*** The memory block in this algorithm represents an n-bit register in the AHPL program.

By defining the SCZs, the wiring problem between gates on a gate array chip is now simplified to the wiring of gates included in each SCZ and the wiring of SCZs. To prove the advantage of SCZ method, let us use a real example to explain the resource (CPU/MEMORY) saved by using this method.
Figure 4.4 Defining a Strongly Connected Zone (SCZ)

SCZ 1: 66, 67, 60
SCZ 2: 54, 55, 56, 57, 58
SCZ 3: 88, 89
Example 4.1: If 300-component circuit is under design, please compare the CPU time and memory used for each following case: (1) without SCZ method (2) with SCZ method.

Solution:

(1) without SCZ method

CPU Time :: \( \frac{300!}{(212981)} \) = 44850 units
MEMORY :: 300 \times 300 = 90000 bytes

(2) with SCZ method

If the SCZ method was used and the circuit was partitioned into 30 SCZs and each SCZ contained 10 circuit components, then the CPU time and MEMORY used will be:

CPU Time :: \( \frac{10!}{(2181)} \times 30 + \frac{30!}{(21281)} \) = 1785 units
MEMORY :: 30 \times 30 = 1000 bytes

*** The computation of CPU time is based on how many times of searching needed to figure out the interconnection relation between components.

*** The computation of required MEMORY is based on the bytes needed to construct the connection table.

*** Mean values have been used in case (2) to ease analysis.
From the above example, there is no difficulty for us to figure out that a large amount of CPU computation time and memory units has been saved by using the SCZ method. In order to help understanding the argument mentioned, two experimental results are presented in schematic forms for easy reference. The relation between CPU computation time and SCZ partitions is shown in Figure 4.5; and the relation between MEMORY used and SCZ partitions is shown in Figure 4.6.

Since the SCZ formation is a stochastic processing, there is no way to constrain the number of components within a SCZ in order to reach the optimal point. The way we are doing this is to limit the SCZ partition into the acceptable region for certain level optimization. One thing we can conclude here is the complex computation for counting connection force between circuit components and the number of alternatives of chip plan (See Figure 4.3) is highly simplified by using the SCZ method.
Figure 4.5 CPU time for SCZ method

Figure 4.6 Memory usage for SCZ method
4.4.2 Computation of Connection Force

Currently some placement algorithms have been reported for better placement [19-20]. One of these placement techniques is named Force-Directed method. In this algorithm, the connection force between any two components is computed to determine the priority of placement. The components with strong connection force are put together to keep interconnects short. The connection force represents the connectivity between two components, and is computed by the number of wiring connections between these two components. Since the total number of circuit components in a VLSI design may increase to ten thousand or more, the computation of the connection force between components becomes exhaustive and prohibitive. That is why we partition the circuit into some SCZs before the connection force is computed. The computation of connection force is explained in Figure 4.7. By using connection table, the connection force between any two components is indicated. The Y-coordinate of a connection table represents the source component, and the X-coordinate represents the target component.
Figure 4.7 Computation of connection force
4.4.3 Cell Assignment strategy

Prior to the actual cell assignment, the connection force between any two SCZs is calculated to determine the allocating sequence. In order to obtain the best chip configuration for cell assignment, some placement rules are used in this thesis.

By applying the "Force-Directed" method, the SCZ configuration is determined first. After the best configuration of SCZs is obtained, the "Force-Directed method" is applied again to calculate the best configuration of gates within an SCZ. In Figure 4.8 an example is provided to explain the procedure for determining the best configuration. Obviously, more strongly connected elements will be allocated more closely. Once the final chip configuration is determined, we can start assigning uncommitted cells to each logic function according to the predefined gate array library.

As we know, incorrect cell placement may lengthen the interconnection wire and complicate the routing task. In order to ease the later channel routing, some strategies have to be considered during assigning uncommitted cells to logic function.
Figure 4.8 Force-directed chip plan
1. Pseudo-Continuous (PC) Assignment:

In this thesis the PC assignment method is suggested to improve the routing configuration. In this method, cell rows with odd row number (row 1, 3, 5, 7...) are assigned from left to right, and cell rows with even row number (row 2, 4, 6, 8...) are assigned from right to left. As shown in Figure 4.9, the first cell row is assigned from left to right, and the second row is assigned from right to left. The reason we adopt PC assignment is that our CMOS gate array compiler generates a sequenced SCZ list which is formed according to the connection force between SCZs. The SCZ list also determines the cell assignment sequence for SCZs. The use of conventional assignment methods will cause defect routing configuration. This argument can be proved by the comparison made in Figure 4.10. In Figure 4.10 the PC assignment method is compared with the conventional method (always-right or always-left assignment). From examining the resultant routing configuration in Figure 4.10, we understand profit is obtained from using the PC assignment method.
Figure 4.9  Pseudo-continuous cell assignment

Figure 4.10  Comparison between PC and Always-right method
2. Leaving space between macros.

Leaving unused cells between SCZs or marccocells is a method to decentralize the routing so that the all routing segments will not congest in a certain routing area and cause routing problem. In the current compiler one unused basic cell is placed between SCZs. Although this method (leaving space between macros) wastes some basic cells, it effectively eases the routing problem especially when the channel capacity (or channel width) provided by the designer is very critical. In Figure 4.11 a simple example is shown to demonstrate the alleviation of routing congestion by using this method.
Figure 4.11 Routing decentralization
4.5 Algorithm

The whole idea introduced in this chapter may be explained by the following algorithm.

ALGORITHM: MBPC CELL PLACEMENT

1. SCZ partition;

2. Computing connection force between SCZ to determine the optimal SCZ configuration;

3. Computing connection force between components within each SCZ to determine optimal cell configuration in each SCZ;

4. Assigning uncommitted cells to each logic function with exploiting PC (Pseudo Continuous) and LS (Leaving Space) techniques;

5. Repeat step 4 until all gates are assigned.
CHAPTER FIVE

A HYBRID APPROACH TO OPTIMAL CHANNEL ROUTING

A key problem in the design and implementation of VLSI auto layout system is the routing problem. Examining the current reported routers, the majority of these routers are able to satisfy user's requirement in many instances, but are often unable to complete the routing of all nets in most complicate circuits [25-26]. These unrouted segments eventually require manual intervention — a tedious and time consuming necessity. In order to minimize design time, an auto router with the capability of 100 percent wiring is required for this purpose.

Except from the 100 percent routing completion rate, other significant factors, such as, routing channels, the number of vias, and the length of wiring segments must be considered at the same time. Due to the complexity of the routing problem, most current routing algorithms focus on one or at most two of these factors. For example, minimum routing area and 100 percent completion rate are the most frequently considered.

In this thesis, a global channel router is presented for solving two-layer model in which wires on different
layers can cross each other or share a corner. The strong point of this global router is that all routing factors mentioned above are considered at the same time.

5.1 Background

There are numerous techniques for routing a rectangular area. The first and most general being is Lee's maze router [21]. The advantage of Lee's algorithm is that it assures a minimal cost path in a maze between two points if such a path exists. The disadvantage of the maze router is it uses a large amount of computer memory and CPU time to mark the available routing cells. As shown in Figure 5.1, Lee's algorithm starts labeling the starting point $S$ by 0, and then marks available cell around $S$ by 1, in the same way, until the object point $T$ is met.

Since the labeling process is done sequentially, the router using Lee's algorithm routes one net at a time; consequently, some of the pre-routed nets may block unrouted nets, requiring human work to complete the routing. The hierarchical router by Burstein [22] attempts to route one grid at a time and has the same limitations as the previous techniques. From above statement, we know that arbitrary decisions in the early stages of routing may affect or block the subsequent routing process.
Figure 5.1 Lee's maze router
Global routing is an attempt to route all or a number of the required wires simultaneously [23]. In other words, a global router should be capable of considering many or all wiring segments at the same time. In this thesis, both heuristic and algorithmic methods are combined to assure 100 percent completion rate, minimum routing area, minimum wire length, and minimum vias used. The whole channel routing task may be divided into the following main parts:

1. Classifying and Splitting the routing segments.
3. Applying HE algorithm for track assignment.
4. Choose heuristic search or CAD interactive method to meet 100% routability.

The implementation of each step will be discussed in more detail in the following sections. In order to help understanding the following explanation, some technical terms are explained below:

1. Terminal: the ends of a wiring segment.
2. Routing Block: the area between any two adjacent cell rows which includes predefined channels for routing.
3. Track: routing channel included in the routing block which is used for dropping wiring segments.
5. Feedthrough Path: The route that provides wiring between routing blocks.


7. Target Point: the stop end of a wiring segment.

8. Branching node: the points on a horizontal segment from which a vertical branch starts.

9. Via: the point used to connect wires on different routing layer.

5.2 Classifying Routing Segments

All wiring connections on a gate array chip can be divided into two categories:

1. Connections included in one routing block as shown in Figure 5.2.

2. Connections not included in one routing block as shown in Figure 5.3.
Figure 5.3 Second type routing segment
At the beginning of the global routing, the system identifies the second type segments, and splits them into 3 individual segments as explained below for better processing.

1st Segment:
From the source point (A) to the point which lies on the intersection (B) of adjacent cell row and the nearest available feedthrough path.

2nd Segment:
Travelling along the choosed feedthrough path from (B) to the cell row (C) which is just above the cell row including the target point.

3rd Segment:
From point (C) to the target point (D).

An example is given in Figure 5.4 to explain how a second type segment is split. After this spliting task, all wiring segments become the first type. The reason for doing this is we want to consider as many segments as possible in each routing cycle. By splitting segments, all wiring segments that pass the current routing block can be sorted and handled simultaneously on each routing cycle.
Figure 5.4 Splitting a second type segment
5.3 Merging Segments Sharing a Common Branching Node

In order to avoid undesirable segments, we merge all wiring segments sharing a common branching node into a larger segment. Referring to Figure 5.5, four segments AB, BC, CD, and DE may be merged into a larger segment AE. The whole merging task may be depicted as follows:

![Diagram showing merging of segments AB, BC, CD, DE into AE]

Figure 5.5 Merge of segments
STEP 1:
Selecting a segment AB from unmerged segments (W) and delete AB from W.

STEP 2:
Examining if any other segments connect to terminal A or B. If a segment BC were found, then segment AB and BC is merged into AC. Mark branching node B and delete BC from W.

STEP 3:
Examining if any segments in W connect to nodes (A, B, or C). If a segment CD were found, then segments AC and CD is merged into a new segment AD. Mark branching node and delete CD from W.

STEP 4:
Repeat step 3 until no more merges can be found. Save the final merged segment.

STEP 5:
Save the final merged segment and repeat step 1 to step 4 until all unmerged segments in W are checked.

This merging step performs an important role in optimizing the routing configuration. From Figure 5.6, we found this merging process guarantees the minimum vias used and the minimum wiring length.
Figure 5.6 Routing optimization by merging segments
5.4 HE Algorithm for Track Assignment

Track assignment can be done in numerous ways depending on the objective function. Our objective is to minimize the number of tracks in a routing block. In other words, to minimize the routing area. As we know, most of the area on a gate array chip is used for routing purpose. The minimum routing area will assure basic minimization on the chip area.

The HE (Horizontal Edge) algorithm is an algorithm to arrange all wiring segments in a routing block into the best routing configuration. It assures the minimum routing channels used. The basic idea of the HE algorithm comes from Hashimoto [24]. The only difference here is we merge all possible segments before applying the algorithm. In order to help understand this algorithm, the main points of the algorithm are explained below:

**HE ALGORITHM:**

1. Extract all segments \((Y)\) which might lie on the current routing block.

2. Start dropping nets on an available track by selecting the segment \((L_1)\) with the most left edge in \(Y\). Delete \(L_1\) from \(Y\).

3. Find the next segment \((L_2)\) whose left edge is most close to the right edge of \(L_1\). Delete \(L_2\) from \(Y\).

4. Repeat step 3 until no more segments can be found to drop in the current track.

5. Repeat step 2 to 4 until all segments in \(Y\) are assigned.
Figure 5.7 provides a simple example of applying this algorithm.

Figure 5.7 HE (Horizontal End) algorithm

According to the experiments, most of the time all segments will be assigned a track for wire connection at end of this algorithm. But, if the user-provided routing width is very critical (insufficient to drop all segments under this algorithm), we may have some segments left and unassigned at the end of this algorithm. Under this worse condition, current reported channel routers use two strategies to deal with:
1. Mark the unassigned segment and ask for manual intervention. — As mentioned at the beginning of this chapter, it is always tedious and time consuming. An undesirable increase in design time becomes inevitable.

2. Automatically increase channel width — somerouters [2,25,26] use this method to achieve 100 percent routability. Theoretically this might be an alternative, but actually, it may not be acceptable or feasible in real manufacturing.

From the above statement, these two impractical methods cannot be used in an intelligent router.

5.5 Best-First Search with back-tracking

Today's CAD design for VLSI circuits confronts a revolution caused by the booming technology -- Artificial Intelligence (AI). Along with the increasing complexity in the VLSI circuits, the AI technology has become the only way to reach the state-of-the-art design.

In this section an intelligent router using the AI search techniques is introduced as a post-processor for our global channel router. The main obligation of this heuristic router is to complete all unwired segments left from HE algorithm for track assignment. In Figure 5.8, a flow chart is given to represent the architecture of the post-processing router.
Figure 5.8 Best-First with back tracking
5.5.1 Data Structure

Examining searching algorithms used in the AI technology, most of them have time-bound or space-bound problem. For instance, the bread-first search requires too much space, $O(b^d)$, and the depth-first search uses too much time, $O(e^d)$, and does not always find the cheapest path (where "b" is node branching factor, "d" is depth, and "e" is edge branching factor).

Based on above observation, constraint experts have been used in our search algorithm to minimize the number of effective expanding nodes (the maximum value for node branching factor "b" is 3). Furthermore, we use a heuristic method to limit the storage used. Only the best expanding node is saved, the other effective nodes are temporary discarded. This method is similar to the Depth-First Iterative-Deepening (DFID) search first used in Slate's CHESS 4.5 [27]. The DFID is a search algorithm which suffers neither the drawbacks of breadth-first nor depth-first search on trees [28].

Although the theoretic lower and upper bounds for space and time is $O(d)$ and $O(b^d)$, we always get rid of the worst case since the heuristic method and the constraint experts used. More data structure analysis for AI searching algorithms may be found in Winston[29], Nilsson[30], and Barr[31].
5.5.2 Constraint Experts

During each node expanding cycle, we check "constraint expert" to help selecting effective nodes for expanding. Each subexpert in the constraint expert consists of simple rules for checking the availability of the current testing node. These experts are explained below:

1. Illegal Trespassing Constraint Expert:

A routing path is allowed to cross the cell row only on the source point, feedthrough path, and target point during the searching cycle. We have to check if the testing node has trespassed the other forbidden pins on the cell row. A simple example on this argument is shown in Figure 5.9.

![Figure 5.9 Illegal trespass on cell rows](image-url)
2. Routing Layer Constraint Expert:

Since any two wiring segments are allowed to cross each other only when their direction are different (one is horizontal and the other is vertical). No two horizontal or vertical segments are allowed to cross each other. During each expanding cycle, we have to check if the testing grid has been used as a wiring route on the same direction. This argument is shown in Figure 5.10.

\[ \text{Figure 5.10 Routing layer constraint} \]
3. Dead End Constraint Expert:

It is possible that the trying route meets a dead end (in which no more expanding nodes can be found). These dead-end nodes have to be recorded by the Dead End Constraint Expert to avoid wasteful and unnecessary duplicate searching. In Figure 5.11 the dead end is described.

Figure 5.11 Dead-end constraint
4. Dimension Constraint Expert:
This constraint expert checks if the expanding node has exceeded the dimension of the predefined chip area. Figure 5.12 gives explanation on this constraint.

Figure 5.12 Chip dimension constraint
Due to the use of the constraint experts, the space, time, and cost factors (usually used to evaluate a search algorithm) for our routing search is highly improved.

5.5.3 Weight Function

Since every wiring terminal has been assigned a coordinate \((x, y)\) during cell allocation, the shortest distance between any two nodes can be computed by:

\[
\text{DISTANCE} = \text{ABS}(x_2-x_1) + \text{ABS}(y_2-y_1)
\]

The distance between two nodes is used as one of determining factors for selecting the best expanding node. Referring to Figure 5.13, the expanding node with the shortest distance to the target node is selected as a member of the searching route.
Figure 5.13  Best-first by shortest distance
Another important factor to affect the selection of the best expanding node is the direction. In each expanding cycle, we may have two nodes with the same distance to the target node. Under this condition, we will prefer the cheapest node in the same expanding direction (keep original direction) rather than the node on the other expanding direction (from vertical to horizontal or from horizontal to vertical). As explained in Figure 5.14, if we are trying to route from point A to point B, then route (C) will be preferred since it uses minimum number of vias which is an important factor to evaluate the routers. In fact, minimizing the number of vias reduces the cost of manufacturing and propagation time of a routing wire.
Figure 5.14 Routing between two terminals
5.5.4 Algorithm

In the heuristic search, the source point (root) is passed to start the search. During each expanding cycle, routing constraints are checked to find the best expanding node (leaf). This expanding sequence will continue until the target node is found.

During a search, if a dead end was found, the heuristic router automatically goes back one level (to its parent node) instead of starting a new search from the source node. The whole searching tree structure of a simple example is shown on Figure 5.15. Only nodes in solid circle will become a tree node and be actually saved.

ALGORITHM: BEST-FIRST WITH BACK TRACKING

1. Passing starting node
2. Expanding children nodes
3. Check constraint expert
   IF (Deadend) THEN
      mask the deadend;
      back track to parent node;
   ELSE
      select the best expanding node;
   END
4. Repeat step 2 and 3 until the target node is found
Figure 5.15 Best-First with back tracking
5.6 CAD Interactive Channel Routing

The option of CAD interactive routing is implemented in the gate array compiler to provide an alternative for handling critical paths. By the interactive routing, the AHPL user is allowed to add/delete wire segments to/from the system-generated routing configuration. The reason for doing this is based on the following considerations:

1. Propagation delay

Since the propagation delay is not well-considered in the current AHPL-based gate array design, the AHPL user may want to meet the system's timing requirement by routing critical paths manually. The interactive routing capacity developed in this thesis made this idea possible.

2. Personal favorite

The AHPL user may not satisfied with the system-generated layout, and want to reroute one or more of these segments. The built-in interactive routing allows the AHPL user to make any changes on the final routing during the routing stage.

After the AHPL user specified the X-Y coordinates of the segment ends, the system automatically add/remove a segment to/from the routing configuration. Due to the implementation of the interactive routing, the hardware compiler developed in this thesis becomes more flexible and
powerful with comparison to published algorithmic-approach compilers.

The first step to use the CAD interactive routing is specifying the coordinates of the routing segment. The coordinate definition used in this thesis is explained in Figure 5.16. The X-coordinate is computed by the unit of pitches (the distance between adjacent pins), and the Y-coordinate is computed by the unit of row numbers. With the aid of the zoom-out capacity implemented in this compiler, it is not difficult for the AHPL user to specify the X-Y coordinates for a segment on the chip.

The basic idea for implementing the CAD interactive routing is shown in Figure 5.17. In each interactive transaction, the system saves interactive commands in the command file (ACTIVE.DAT) as if the backup file used in the computer system, and then re-executing all commands in the command file. Some improvements may be added to polish the current CAD interactive routing. We will discuss the topic in chapter seven.
Figure 5.16 Specifying X-Y coordinates on a gate array chip
Figure 5.17 CAD interactive routing
CHAPTER SIX

EVALUATION OF THE RESULTS

In this chapter an evaluation will be made on the implemented CMOS gate array compiler based on some important routing criterions. The identical evaluation will be applied to Dr. Chen's hardware compiler [2] for comparison purpose.

6.1 Evaluating a Layout System

The evaluation of a VLSI auto layout system is usually done by measuring the last wiring layout with the following matrices:

1. Wiring completion rate

Human intervention is costly and time-consuming. The human intervention may add several man months to the design time of a gate array chip. In today's VLSI design, long design time may cause a project to lose the market. Our dream toward the design automation is the 100 percent automation. Any necessary human intervention on the wiring completion will degrade the channel router.

2. Cell efficiency and chip area:

As we know, the objective of today's VLSI design especially in semicustom design is to improve the cell efficiency so
that the compaction of circuit designs can be highly increased. The chip area and cell efficiency are mainly dominated by the following factors:

1. Total number of transistors used.
2. Routing channel width between cell rows.

The total number of transistors used for implementing the designer's circuit is usually measured as cell efficiency. An efficient hardware compiler performs high cell efficiency to save the chip area. In comparison with other hardware compilers, it uses much fewer cells (or transistors) to implement the same circuit. In this chapter we will evaluate the cell efficiency by comparing the total transistors used in building the same circuit function.

The second factor determining the chip area is the channel width between cell rows. A good channel router uses as few routing channels as possible to complete interconnection. Obviously, if the total transistors and the channel width can be reduced, then the chip area will be highly reduced.

3. Number of vias
The number of vias used in the final routing configuration is one of the cost factors determining the chip development cost. The better the channel router is, the fewer vias it uses.
4. Wiring Configuration:
The wiring configuration may be evaluated by examining the following items:

A. the total length of wiring segments
B. the length of the critical path

The total length of wiring segments is a cost factor of chip processing, and may complicate the routing problem. To reduce the total length of wiring segments, a good floor plan and an intelligent channel router are needed to hit this target. By comparing the final wiring configurations, the basic quality of the hardware compiler can be judged.

The allowable length of the critical path is determined based on the timing consideration. Since the propagation delay of a wiring segment is proportional to the length of the segment, each routing segment has to be carefully limited in order to meet the timing requirement.

6.2 Systematic Difference

Before the AHPL examples are used to evaluate the two compilers, we would like to introduce the systematic differences between the two compilers, which determines the quality of the final layout.

The first systematic difference between Hu's and Chen's compilers is that Hu's compiler provides more interactive and flexible design than Chen's compiler. In
Hu's design, all parameters defining a gate array chip are determined by the AHPL user (see chapter four). This interactive characteristic allows the AHPL user to select the best-fit chip from the IC market without accessing the source codes of the hardware compiler. On the other hand, all parameters defining the chip size are predefined in the definition file. The AHPL user has to modify the parameters and recompile the whole source program in order to fit different chips selected. Furthermore, Hu's compiler provides the capacity of CAD interactive routing. Under this methodology, the AHPL user is allowed to modify the final routing configuration and route the critical paths manually. From experiments, the wires routed by the Best-First algorithm are not always optimal although the objective of 100% routing is met. This non-optimal wire is mainly caused by the trade-off selection between distance and direction factors. In the next chapter some improvements will be proposed for further investigation.

The second systematic difference between this two compilers is the approach to the 100 percent routing completion rate. Since hybrid optimization (including the global router, the Best-First with back-tracking search, and split-and-merge skill) is built in the CMOS compiler, 100 percent routing completion rate (if it is possible) is guaranteed by Hu's compiler. Although the searching task is
always time-consuming, the time penalty has been much improved in Hu's compiler due to the use of the constraint experts (see chapter five). Now let us examine Chen's compiler. Once the channel router needs more than 6 routing channels (default value) to drop all wiring segments, it automatically generates extra channels (instead of trying any other routing possibility) for utilization in order to reach the 100 percent completion rate. As mentioned in chapter five, this approach might not be accepted in the real processing technology.

The third systematic difference between the two compilers is the chip plan. In Chen's compiler, the control section is always processed first and assigned to the top part of the chip. One unused cell row between control section and data section is used for communication between the two parts. That is the reason why we add a cell row to Chen's final layout during evaluation. The cell assignment method is to assign cells in parallel (depending on the bits of the assigning macrocell) from left to right. On the other hand, Hu considers the control section and the data section simultaneously by partitioning the whole circuit into some "Strongly Connected Zones" (SCZ). The cell allocation for each macro is determined by the connection force between cell macros or SCZs (see chapter 4). In the following section, we will demonstrated the cell placement method used
in Chen's compiler wastes too many transistors.

Another systematic difference between the two compilers is the observability. In Chen's compiler there is no way to trace the external I/O lines and cell macros from the final routing configuration. The AHPL user is not allowed to check the accuracy of the final layout. On the other hand, in Hu's compiler a cell assignment table is provided for checking purposes. By referring to the cell assignment table and color marked on the cell array, the AHPL user can easily identify every macro cell on the gate array chip, in the same way, the interconnection between macros can be checked also.

6.3 Examples

In this section three AHPL programs are used to evaluate Hu's and Chen's compiler. During evaluation, we found some AHPL programs are not accepted by Chen's compiler. For example, programs with too many inputs or XNOR gates are not accepted by Chen's compiler. It seems that the macro retrieval subroutine and the fan-in/out consideration in Chen's compiler needs some modification for proper function. Based on this fact, only four AHPL programs are analyzed in this section.

Another fact in Chen's compiler is it does not include the wiring between the I/O cells and the circuit components (macros). All connections between external
inputs/outputs and cell arrays are left as open-ends in Chen's layout. Based on this reason, the comparison between Hu's and Chen's compiler will be made based on the same design level. That means in the following examples we neglect the vias, wires, or routing channels used in Hu's compiler, which are used to connect I/O cells and array macrocells.

6.3.1 4-bit Shift Register

The first example used in this chapter is a four-bit shift register. Two external inputs are used to control the "shift" and "load" operations. The AHPL source codes is shown below:

```plaintext
MODULE: SHIFTR.
MEMORY: M[4].
EXINPUTS: RESET; CLOCK; DATA[4]; SHIFT; LOAD.
EXOUTPUTS: OUT.

BODY SEQUENCE: CLOCK.
1 =>(^LOAD+SHIFT),LOAD,SHIFT)/(1,2,3).
2 M*LOAD <= DATA;
   =>(^LOAD,LOAD)/(1,2).
3 M*SHIFT <= (M[1:3],M[0]);
   =>(^SHIFT,SHIFT)/(1,3).
END SEQUENCE
   CONTROLRESET(RESET)/(1);
   OUT=M[0].
END.
```
The final layouts for this example by using Hu's and Chen's compiler are shown in Figure 6.1 and 6.2. From examining the two layouts, a simple comparison table is made below:

<table>
<thead>
<tr>
<th></th>
<th>Hu</th>
<th>Chen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chip Area</td>
<td>453,400</td>
<td>987,875</td>
</tr>
<tr>
<td>Cell Efficiency</td>
<td>0.9</td>
<td>0.41</td>
</tr>
</tbody>
</table>

According to the comparison table shown above, the Hu's compiler is better than Chen's. Referring to Figure 6.2, many cells are wasted in Chen's layout. Furthermore, we can easily point out some vias are wasted in Chen's compiler. For example vias are wasted in area A, B, C, and D. Furthermore, we figure out the cell assignment method (by starting assigning memory register vertically on each cell row [2]) used in Chen's program causes too much chip area waste. Obviously, in this example two cell rows (row 5 and row 6) are wasted by Chen's compiler. This two cell rows can be saved by assigning cell row 5 to the right end of the cell row 3 and cell row 6 to the right end of the cell row 4.
Figure 6.1 SHIFTR by HU's compiler
Figure 6.2 SHIFTR by CHEN's compiler
6.3.2 Operation Test

The second example used in this chapter is a simple operation program including OR (+), AND (&), XOR (@), CATENATE (,), and OR DEDUCTION (+/). The AHPL program is shown below:

```
MODULE: OPERATION.
MEMORY: M.
EXINPUTS: RESET; CLOCK.
EXINPUTS: IN1; IN2; IN3; IN4; DIN[5].
EXOUTPUTS: OUT; RESULT.

BODY SEQUENCE: CLOCK.
1 M <= IN1 + IN2 @ IN4 & IN3.
2 RESULT = +(DIN[3:4],IN2);
   =>(1).

END SEQUENCE
   CONTROLRESET(RESET)/(1);
   OUT=M.

END.
```

The final layouts for HU's and CHEN's compiler are shown in Figure 6.3 and 6.4. From examining Chen's layout in Figure 6.4, we know one routing channel is wasted by segments F and G. The segments C, D, F and H should be routed on the same channel, and the segment A and G should be assigned to the same routing channel in order to save a routing channel. Simple comparison table of this example is listed below for reference.
<table>
<thead>
<tr>
<th></th>
<th>HU</th>
<th>CHEN</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Chip Area</strong></td>
<td>144,160</td>
<td>192,225</td>
</tr>
<tr>
<td><strong>Cell Efficiency</strong></td>
<td>0.794</td>
<td>0.63636</td>
</tr>
</tbody>
</table>

Figure 6.3 OPERATION by HU's compiler
Figure 6.4 OPERATION by CHEN's compiler
6.3.3 Detector

The third example we used is a simple input detector. The AHPL program is shown below:

MODULE: DETECT.
MEMORY: TEMP[4].
EXINPUTS: RESET; CLOCK; INP[4]; READY.
EXOUTPUTS: OUT.

BODY SEQUENCE: CLOCK.

1 =>("READY,READY)/(1,2).
2 TEMP <= INP.
3 OUT = ^(&/TEMP);
   =>(1).

ENDSEQUENCE
   CONTROLRESET(RESET)/(1).
END.

The routing configurations for this example are shown in Figure 6.5 and Figure 6.6. From examining Chen's layout, we know at least two cell rows (row 4 and row 5) and one routing channel (see segments F and G) are wasted. The comparison table is listed below for reference.

<table>
<thead>
<tr>
<th></th>
<th>HU</th>
<th>CHEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chip Area</td>
<td>220,480</td>
<td>680,100</td>
</tr>
<tr>
<td>Cell Efficiency</td>
<td>0.92</td>
<td>0.3333</td>
</tr>
</tbody>
</table>
Figure 6.5 DETECT by HU's compiler
Figure 6.6 DETECT by CHEN's compiler
6.3.4 Sequential Selector

As introduced in chapter one, today's VLSI design may include more than 30,000 components. In order to prove the applicability of the CMOS compiler for VLSI design, a more complicated AHPL program is used to test this two compilers. The external input vector Y is used to select the data in register A, B, or C via a 2-to-4 decoder. The shift control signal P is used to input the external data vector X into A, B, or C. The complete AHPL description is shown below for reference:

MODULE: SEQSEL.
EXINPUTS: X[4]; Y[2]; P; Q; CLK; RESET.
MEMORY: A[4]; B[4]; C[4]; D[4]; F[4].
BUSES: BUS1[4].
EXOUTPUTS: Z[4].

BODY SEQUENCE: CLK.

1 => (P,Q, Q&~Q)/(2,4,1).
2 => (P)/(2).
3 A <= X; B <= A; C <= B; D <= C;
    => (1).
4 F <= BUS1;
    => (1).

ENDSEQUENCE
CONTROLRESET(RESET)/(1);
Z=F; BUS1 = (A!B!C!D) * DCD(Y).
END.
CLU : DCDER(IN) {I}.
INPUTS : IN[I].
OUTPUTS: OUT[2^I].
CTERMS : RESULT[2^I].

BODY
    FOR M=0 TO (2^I)-1 CONSTRUCT
        RESULT[M] = &/TERM(M;IN)
    ROF;
    OUT = RESULT.
END.

The routing configurations for this example are shown in Figure 6.7 and Figure 6.8. From examining the evaluation table, Hu's compiler is still superior than Chen's compiler.

<table>
<thead>
<tr>
<th></th>
<th>HU</th>
<th>CHEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chip Area</td>
<td>987,650</td>
<td>1,415,765</td>
</tr>
<tr>
<td>Cell Efficiency</td>
<td>0.905</td>
<td>0.65</td>
</tr>
</tbody>
</table>
Figure 6.7 SEQSEL by HU's compiler
Figure 6.8 SEQSEL by CHEN's compiler
6.4 Results

From above evaluation, we know the fact that Hu's compiler always generates more compacted layout than Chen's compiler and Chen's cell placement method wasted too many basic cells. The macro cell library used by Chen's compiler is suggested to be updated. From examples one and three, we proved Chen's compiler is unable to modify the chip size according to different circuit designs. Optimization for cell assignment and channel routing are also needed for Chen's compiler.

It is worthy to contribute more efforts on Hu's hardware compiler so that it can be used as a complete CAD tool for the AHPL-based design. Some valuable suggestions for improving Hu's compiler will be given in the next chapter.
CHAPTER SEVEN

SUGGESTIONS AND CONCLUSIONS

In this chapter some suggestions for further improvement and research are given to end this thesis. As evaluated in chapter six, the design idea implemented in this thesis is good and acceptable. The hardware compiler designed in this thesis may be further improved as a complete AHPL-based VLSI design tool. Concerning this purpose, some future improvements are suggested:

1. Including the wiring from I/O cells to bonding pads, and the power buses to cell components.

2. Allowing dynamic cell placement.

The space or gap (the number of unused basic cells) between adjacent macros (see chapter 4) is currently defaulted as 1 to ease the routing problem. As a matter of fact, we have wasted few cells in the current system since some adjacent macros do not need routing decentralization. In order to assure the optimal routing configuration can be obtained, the future designer is suggested to design the space or gap as a variable based on the number of cells provided by the AHPL user. In that way, the communication
between channel routing module and cell placement module must be provided to find the optimal gap between macros. In other words, once the channel router figured out the difficulty of routing for current cell placement, the cell placement module will be notified to modify the gap between macros for better routing processing. Basic idea on this modification is shown in Figure 7.1 for reference.

3. Improving the capability of interactive routing.

Although the current CAD interactive routing allows the AHPL user to modify the final routing configuration, the system does not check the routing violation for the user. For example, the user may route a horizontal segment over another horizontal segment. For future improvement, the system should assign the highest priority to the user-interactive segments. That means the user is allowed to arbitrarily route a segment on the chip, and the system is responsible to check the routing violation and rearrange all related wiring segments.
Figure 7.1 Dynamic cell placement
4. Improving the speed of the Best-First with back-tracking

As we know, searching is always time-consuming. From different experiments, we find out the back-tracking characteristic in the current search algorithm spends extra time to complete a routing. Although the time penalty is trivial with comparison to the human intervention explained in chapter five, a better searching algorithm may be built to improve the current speed. The Best-First search without back-tracking [29] implemented by queueing method would be one of the most potential candidates for future improvement.

5. Improvements on routing configuration

Since the HE algorithm is used to save the routing channel, some wiring segments are undesirably extended as shown in Figure 7.2. The future improvement may include a post routing optimization to polish the final routing layout.

![Figure 7.2 Routing improvement](image-url)
6. Designing post-processor to generate PG formats.

In the AHPL-based SLA (Storage Logic Array) design, we have successfully translated the AHPL language into the PG (Pattern Generator) [32] format by using intermediate transform language, such as CIFSYM [2,33]. These PG formats are later fed into the Pattern Generator and generated the real processing masks [34]. In the same way, a post-processor generator may be designed for the CMOS gate array to generate the PG formats. These formats can be fed into the Pattern Generator to generate the interconnection masks for processing.

Due to the complexity of the routing problem in the VLSI design, the rule-based expert system is suggested to improve the solution quality [35]. The term "expert system" refers to a computer program that is a collection of heuristic rules and domain facts that have proven efficient in solving special technical problems. The technology of Artificial Intelligence (AI) has been widely used in solving engineering problem [36-37]. The reason for doing this is the restrictions of traditional algorithmic approaches. Generally speaking, the algorithmic approaches are oversimplified and overconstrained. They can only be applied to solve one specific type of routing problem. For example, an algorithmic channel router may be only applied in single-layer model or two-layer model with assigning different
layers to different directions. Whenever the application changed, the routing program with built-in algorithms has to be renewed every time. Unlike the algorithmic approach, the knowledge-based systems (KBS) include thousands of rules (or more) to handle every possible condition and obtain the final optimal routing configuration without changing any algorithm in the routing program. Obviously, the KBS approach is more flexible than the algorithmic approach. We leave this exploration to the future designer.
APPENDIX A

Using the AHPL CMOS Gate-Array Compiler

In order to run the AHPL CMOS gate array compiler in HU's directory, the user must follow the system message and provide requested information. In this appendix an actual executing example is provided to serve as a reference for AHPL users.

$ RUN [HU.S23]GATE_TEK
   ---> activate the system <--
   ---> for laser printout GATE_LASER <--

Enter input filename (Stage01 output): shift.stl
   ---> input your stage01 output file <--

Restore stage01..
Logic Configuration..
Block Formation..
Floor Plan..
Cell Assignment..

Total Equivalent Gates Estimated: 108
   ---> total gates estimated by the system <--

Specify the Dimension of gate-array chip below:
   ---> enter your chip dimension below <--

1. Cell row: 5
2. Cell column: 20
3. Routing Channel Capacity: 7
4. Bonding pads: 32

Channel routing..
Channel routing...
   --> trademark shows up <--

Type numerical key to start the program: 5
   --> chip layout shows up <--
   --> follow system message to check routing <--
   --> To use interactive routing, read <--
   --> chapter 4 and 5 first <--

-----------------------------
Reset the TEK terminal
-----------------------------

Type SETUP key

*RESET

Type SETUP key

Type RETURN key
APPENDIX B

Organization of Subprograms

-------------
!!! modified stage23
-------------

GATEARRAY
RESTORE
BLDSYS
STAGE3

-------------
!!! cmos compiler
-------------

CONVERT
COLLECT
INTERBLK
OUTERBLK
CHIP
XYPOINT
FIND
ROUTE
SAVE
CLEARST
TRUST

-------------
!!! TEK graph routing
-------------

GRAPH TEXTO FIGURE0
CELLPLOT VIEW HUMAN

-------------
!!! TEK function subroutine
-------------

VERT1 VERT2 VERT3 LINE POLYGON
INIT4105 AREA WINDOW BAUD
DIASET LINES EE DIAEN DIACLR
DIADIS MOVE DRAW LCOLR LSTYLE
GETEXT PATH TEXSIZE TEXCOLR MRKTYPE
MARK COPY BPANEL EPANEL EXIT VISIBLE
INVISIBLE NOGRAPH FPATTERN SQUARE

120
APPENDIX C

Parameter Expansion for Future System

PARAMETER (MAXGATE=5000)
--> maximum logic gates in a AHPL design

PARAMETER (MAXIN=10)
--> maximum inputs of a gate

PARAMETER (USRCOL=200)
--> maximum # of basic cells on a cell row

PARAMETER (MAXPOINT=4000)
--> maximum # of pins involved routing

PARAMETER (MAXLINE=10000)
--> maximum # of routing segments

PARAMETER (MAXMEM=100)
--> maximum # of memory blocks declared in AHPL

PARAMETER (MAXBLK=300)
--> maximum # of SCZ partition

PARAMETER (MAXSEG=3000)
--> maximum # of segments within a routing block

Some other array definition in PARA.FOR may also need modification when the system is expanded for real VLSI design.
APPENDIX D

Data Files Generated by the System

CHECK.DAT
This file includes the information of layout, circuit components, and cell efficiency for checking purpose.

MAN.DAT
This file includes line segments for interactive routing if the heuristic method was not used.

FLAT.TXT
This file includes all horizontal segments of layout.

VERT.TXT
This file includes all vertical segments of layout.
Reference


37. GOULD AMI semiconductors, "1985 Gate Array Databook," Santa Clara, CA.