Module generators from topological descriptions and graph theoretic approach.

Shakil Kaiser. Siddiq

University of Windsor

Recommended Citation
https://scholar.uwindsor.ca/etd/2770

Follow this and additional works at: https://scholar.uwindsor.ca/etd

This online database contains the full-text of PhD dissertations and Masters’ theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000 ext. 3208.
NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer à l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.
Module Generators from Topological Descriptions and Graph Theoretic Approach

by

Shakil Kaiser Siddiq

A Thesis
Submitted to the Faculty of Graduate Studies through the Department of Electrical Engineering in Partial Fulfillment of the Requirements for the Degree of Master of Applied Science at the University of Windsor

Windsor, Ontario
THE AUTHOR HAS GRANTED AN IRREVOCABLE NON-EXCLUSIVE LICENCE ALLOWING THE NATIONAL LIBRARY OF CANADA TO REPRODUCE, LOAN, DISTRIBUTION OR SELL COPIES OF HIS/HER THESIS BY ANY MEANS AND IN ANY FORM OR FORMAT, MAKING THIS THESIS AVAILABLE TO INTERESTED PERSONS.

L'AUTEUR A ACCORDE UNE LICENCE IRREVOCABLE ET NON EXCLUSIVE PERMETTANT A LA BIBLIOTHEQUE NATIONALE DU CANADA DE REPRODUIRE, PRETER, DISTRIBUER OU VENDRE DES COPIES DE SA THESE DE QUELQUE MANIERE ET SOUS QUELQUE FORME QUE CE SOIT POUR METTRE DES EXEMPLAIRES DE CETTE THESE A LA DISPOSITION DES PERSONNE INTERESSEES.
Abstract

Because of the increasing complexity of the designs, there is a great necessity for automatic layout tools which produce results similar in optimality to those produced by manual design methods. The continuous progress in VLSI technology presents new challenges in developing efficient algorithms for the layout of logic cells.

The feasibility of developing an automatic mask layout generator for switching trees from its symbolic counterpart is explored in this thesis. The symbolic layout is generated from a Switching Tree Layout Synthesizer which is then translated to the mask shapes. Each element in the symbolic layout represents an instance in the mask layout. The mapping of symbolic to mask layout is not constrained to just one solution. Depending on the overall structure of the design, necessary steps such as changing the transistor orientation and contact position and layout compaction for a particular section, are performed to obtain optimized and efficient layout. Two switching tree layout styles, DMS and IMS, are presented here which are dependent on the height and the width of the logic cell. A one-dimensional functional cell generator is also proposed in this thesis. Two algorithms to minimize the height and the width of the functional cell are studied and implemented based on graph theory. It is shown that the switching tree layouts are much more area efficient than that of one dimensional functional cells. Other advantages of the switching tree over other layout methods are also mentioned. Simulations are carried out to justify the heavily pipelining capacity of the switching trees.
To my parents,
Acknowledgements

I would like to express my sincere gratitude and appreciation to Dr. G. A. Jullien, who has been a constant source of inspiration and encouragement. His tremendous support, guidance, and stimulating ideas were invaluable in the successful completion of this thesis. I am thankful to Dr. W. C. Miller for his insightful comments and constant support which provided important direction for this work. I would also like to thank Dr. N. M. Wigley for his invaluable role as my supervisory committee member. I was very fortunate to work closely with Dr. Dimitris Phoukas, who really had been a great boss. Thanks must go to my friends for their support and understanding. Last, but not least, my family has been a constant source of strength and encouragement which kept my hope and confidence alive throughout the course of this work.
# Table of Contents

Abstract ........................................................................................................ iv  
Dedication ..................................................................................................... v  
Acknowledgements ....................................................................................... vi  
Table of Contents ........................................................................................ vii  
List of Figures ............................................................................................... x  
List of Tables ................................................................................................ xii

Chapter 1  Introduction .................................................................................. 1  
1.1  Introduction .......................................................................................... 1  
1.2  Thesis Organization ........................................................................... 3

Chapter 2  Background .................................................................................. 5  
2.1  Introduction .......................................................................................... 5  
2.2  Topological Layout vs. Mask Layout .................................................. 6  
2.3  SKILL Language ................................................................................ 7  
2.3.1  Interactive Language ..................................................................... 7  
2.3.2  User Interface Features ................................................................ 8  
2.3.3  Computational Capabilities .......................................................... 8  
2.3.4  Tight Coupling with the CADENCE System ................................ 9  
2.3.5  Design Database Access ............................................................... 9  
2.4  Technology .......................................................................................... 9  
2.4.1  CMOS ......................................................................................... 9  
2.4.2  BiCMOS ..................................................................................... 10  
2.5  CMOS Dynamic Logic Families ....................................................... 10  
2.6  Layout Styles ..................................................................................... 11  
2.6.1  Semi-Custom Layouts .................................................................. 11  
2.6.2  Full-Custom Layouts ................................................................... 11  
2.7  Layout Design Rules .......................................................................... 13  
2.8  Summary ............................................................................................ 14

Chapter 3  The Switching Tree .................................................................. 15  
3.1  Introduction ........................................................................................ 15  
3.2  Previous work .................................................................................... 16  
3.3  Pipelined Switching Trees .................................................................. 17  
3.4  Graph Based Reduction ..................................................................... 17  
3.5  Construction of the Switching Tree .................................................. 21  
3.6  Switching Tree Layout Styles ............................................................. 24  
3.7  Symbolic Layout (Topological Matrix) to Mask Layout .................. 26  
3.8  Problems with DMS and a solution .................................................... 33  
3.9  Area Calculation ................................................................................ 38  
3.9.1  Direct Method ............................................................................ 38  
3.9.2  Indirect Method .......................................................................... 38  
3.9.3  Which method should be used .................................................... 39  
3.10  Technology File ............................................................................... 40  
3.11  Concluding Remarks ......................................................................... 42  
3.12  Summary ........................................................................................... 42
### Chapter 4
- The Functional Cells .................................................. 44
  - 4.1 Introduction ....................................................... 44
  - 4.2 Graph Theoretic Background .................................. 45
  - 4.3 The Layout Problems ............................................ 46
    - 4.3.1 Width Minimization .......................................... 46
    - 4.3.2 Height minimization ......................................... 47
  - 4.4 The Layout Style and Procedure ............................... 48
  - 4.5 The Algorithm to Find the Euler Path .......................... 52
  - 4.6 The Track Assignment Problem ................................. 53
    - 4.6.1 The algorithm ............................................... 53
    - 4.6.2 An example .................................................. 54
  - 4.7 Comparison with the Switching Tree ......................... 57
  - 4.8 Summary .......................................................... 58

### Chapter 5
- Simulation Results .................................................... 60
  - 5.1 Introduction ...................................................... 60
  - 5.2 Pipelining .......................................................... 61
    - 5.2.1 Pipelined CMOS gates ...................................... 61
    - 5.2.2 Clock skew .................................................. 63
  - 5.3 Latch Structures .................................................. 64
    - 5.3.1 CMOS current latch ........................................ 64
    - 5.3.2 The voltage sensing master-slave latch ................. 65
  - 5.4 Simulation Results ............................................... 66
    - 5.4.1 CMOS 1.2 micron technology .............................. 66
    - 5.4.2 Mitel 1.5 micron technology ............................. 72
  - 5.5 Summary .......................................................... 74

### Chapter 6
- Conclusions and Future Work ....................................... 76
  - 6.1 Conclusion ....................................................... 76
  - 6.2 Future Work .................................................... 77
    - 6.2.1 Switching tree ............................................. 78
    - 6.2.2 One-dimensional functional cell ......................... 78

### Appendix A
- How to use the package ............................................... 82
  - A.1 Switching Tree .................................................. 82
    - A.1.1 Which program should be used ............................ 82
    - A.1.2 Direct mapping style ...................................... 82
    - A.1.3 Indirect mapping style ................................. 83
  - A.2 The Euler Path Approach ...................................... 85

### Appendix B
- Layouts and examples ................................................ 87
  - B.1 Switching Tree .................................................. 87
    - B.1.1 The leaf cells ............................................. 87
    - B.1.2 Direct Mapping Style ...................................... 89
    - B.1.3 Indirect Mapping Style ................................... 92
  - B.2 The Functional Cell ............................................ 92
    - B.2.1 An example ............................................... 92

### Appendix C
- The SKILL Codes .................................................... 95
  - C.1 Switching Tree .................................................. 95
    - C.1.1 Direct mapping style ...................................... 95
    - C.1.2 Indirect mapping style ................................... 106
    - C.1.3 A Sample Technology File (Mitel 1.5 micron) ......... 138
C.1.4  Adding a New Menu in the CTW................................................................. 143
C.1.5  A sample .cdsinit file for Mitel 1.5 micron Technology............................. 145
C.2  Euler Path and the Metal Tracks................................................................. 145
     C.2.1  Euler path......................................................................................... 145
     C.2.1  The metal track(s).............................................................................. 155
     C.2.2  One Dimensional Functional Cell Layout.......................................... 159

Vita Auctoris......................................................................................................... 177
# List of Figures

<p>| Figure 2.1 | The Y chart | 6 |
| Figure 2.2 | A transistor and corresponding layout | 13 |
| Figure 3.1 | Pipelined switching tree | 17 |
| Figure 3.2 | A binary tree | 18 |
| Figure 3.3 | Merging of shared sub-trees | 19 |
| Figure 3.4 | Deletion of common edges | 20 |
| Figure 3.5 | Original and alternative form of siblings | 21 |
| Figure 3.6 | The truth table of the sum bit of a full adder | 21 |
| Figure 3.7 | Switching graph of the sum bit of a full adder | 22 |
| Figure 3.8 | The graph and the corresponding full adder circuit (only the sum bit) | 23 |
| Figure 3.9 | A simple switching graph and the mask layout | 24 |
| Figure 3.10 | Floor plan for the Direct Mapping Style | 25 |
| Figure 3.11 | The floorplan for the Indirect Mapping Style | 26 |
| Figure 3.12 | Flow chart to implement the core section of the tree | 27 |
| Figure 3.13 | The topological matrix | 27 |
| Figure 3.14 | The Matrix after modification | 28 |
| Figure 3.15 | The polysilicon with metal2 | 29 |
| Figure 3.16 | Horizontal metals(metal1) | 30 |
| Figure 3.17 | Transistors in different rows | 30 |
| Figure 3.18 | The core of the sum bit of the full adder | 31 |
| Figure 3.19 | Complete layout for the sum bit of a ripple carry full adder with the latch and the inverters | 32 |
| Figure 3.20 | Layout of the mod17 multiplier (one output) | 33 |
| Figure 3.21 | Upper section of the layout | 34 |
| Figure 3.22 | Lower section of the layout | 34 |
| Figure 3.23 | Modified upper section | 35 |
| Figure 3.24 | Corner cell | 35 |
| Figure 3.25 | Modified layout (with latches and inverters) | 37 |
| Figure 3.26 | The entire switching tree process | 41 |
| Figure 4.1 | A simple graph | 46 |
| Figure 4.2 | A graph and the corresponding layout | 47 |
| Figure 4.3 | The graph without and with the pseudo-edge | 50 |
| Figure 4.4 | The layout for Figure 4.3 (b) | 52 |
| Figure 4.5 | Node conflicts | 55 |
| Figure 4.6 | A layout with the length of the metals | 57 |
| Figure 5.1 | Pipelining | 61 |
| Figure 5.2 | Switching tree embedded in a latch | 62 |
| Figure 5.3 | Timing diagram | 62 |
| Figure 5.4 | Current steering Latch | 65 |</p>
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.5</td>
<td>Master-Slave latch embedded in an N-FET block</td>
<td>66</td>
</tr>
<tr>
<td>5.6</td>
<td>Simulation of MOD 7 @ 142 Mhz</td>
<td>67</td>
</tr>
<tr>
<td>5.7</td>
<td>Test structure 1 (10 high)</td>
<td>68</td>
</tr>
<tr>
<td>5.8</td>
<td>Test structure 2 (10 high)</td>
<td>68</td>
</tr>
<tr>
<td>5.9</td>
<td>Simulation of test structure 1 @ 100 Mhz</td>
<td>68</td>
</tr>
<tr>
<td>5.10</td>
<td>Simulation of test structure 2 @ 100 Mhz</td>
<td>69</td>
</tr>
<tr>
<td>5.11</td>
<td>Test structure 3 (Indirect)</td>
<td>69</td>
</tr>
<tr>
<td>5.12</td>
<td>Simulation of 4 cascaded 10 high trees (test structure 3) @ 70 Mhz</td>
<td>70</td>
</tr>
<tr>
<td>5.13</td>
<td>Simulation of 4 cascaded 10 high trees (test structure 3) @ 83 Mhz</td>
<td>70</td>
</tr>
<tr>
<td>5.14</td>
<td>Simulation of 3 cascaded test structures (4) @ 70 Mhz</td>
<td>71</td>
</tr>
<tr>
<td>5.15</td>
<td>Test structure 4 (12 high)</td>
<td>71</td>
</tr>
<tr>
<td>5.16</td>
<td>Simulation results of 4 cascaded 1m test structures @ 100 Mhz</td>
<td>72</td>
</tr>
<tr>
<td>5.17</td>
<td>Test structure 1m (10 high)</td>
<td>72</td>
</tr>
<tr>
<td>5.18</td>
<td>Test structure 3m (12 high)</td>
<td>73</td>
</tr>
<tr>
<td>5.19</td>
<td>Simulation results of test structure 3m @ 125 Mhz</td>
<td>73</td>
</tr>
<tr>
<td>5.20</td>
<td>Simulation results of test structure 3m @ 142 MHz</td>
<td>74</td>
</tr>
<tr>
<td>A.1</td>
<td>Menu in CIW</td>
<td>85</td>
</tr>
<tr>
<td>B.1</td>
<td>The half_p and half_n inverters (direct mapping style)</td>
<td>87</td>
</tr>
<tr>
<td>B.2</td>
<td>Inverters for the indirect mapping style</td>
<td>88</td>
</tr>
<tr>
<td>B.3</td>
<td>Current steering master-slave latch</td>
<td>88</td>
</tr>
<tr>
<td>B.4</td>
<td>Voltage sensing master-slave latch</td>
<td>89</td>
</tr>
<tr>
<td>B.5</td>
<td>The topological matrix of the sum bit of a full adder</td>
<td>89</td>
</tr>
<tr>
<td>B.6</td>
<td>Layout of the sum bit of a full adder</td>
<td>90</td>
</tr>
<tr>
<td>B.7</td>
<td>Topological matrix for Mod 7 multiplier</td>
<td>91</td>
</tr>
<tr>
<td>B.8</td>
<td>Layout of Mod 7 multiplier</td>
<td>91</td>
</tr>
<tr>
<td>B.9</td>
<td>Layout of Mod 17 multiplier (direct method)</td>
<td>92</td>
</tr>
<tr>
<td>B.10</td>
<td>Layout of Mod 17 multiplier (indirect method)</td>
<td>92</td>
</tr>
<tr>
<td>B.11</td>
<td>The precharged CMOS implementation of the full adder (sum bit)</td>
<td>93</td>
</tr>
<tr>
<td>B.12</td>
<td>Layout of the sum bit of a full adder (Euler path approach)</td>
<td>94</td>
</tr>
<tr>
<td>B.13</td>
<td>The layout of Mod 7 multiplier (the core section)</td>
<td>94</td>
</tr>
<tr>
<td>C.1</td>
<td>Transistor specifications</td>
<td>142</td>
</tr>
<tr>
<td>C.2</td>
<td>The column and row spacing</td>
<td>143</td>
</tr>
</tbody>
</table>
List of Tables

Table 2.1. Design rules ........................................................................................................ 14
Table 3.1. The number of metal wires in left set .............................................................. 36
Table 3.2. Dimensions for different technologies.............................................................. 40
Table 4.1. Comparison of Euler Path and Switching Tree approach .............................. 57
Table 5.1. Simulation results ............................................................................................ 74
Chapter 1

Introduction

1.1 Introduction

Since the introduction of the first commercial integrated circuit, the density of digital integrated circuits has been increasing at a rapid pace. Commercial integrated circuit (IC) chips in the 1960's had about 100 transistors. Through dynamic improvements in the fabrication technology, VLSI (very large scale integration) circuits with millions of transistors are being fabricated at present, and the number of transistors per chip increases every year. The design of chips with such large numbers of transistors presents major problems in design effort and cost. CAD (computer-aided design) tools are developed to help designers cope with these problems. Such tools now play a key role in managing design complexity and improving designer productivity.

The layout of an IC is the process of assigning geometric shape, size and position to the components (transistors and connections) used in its fabrication. It is a complicated, error prone, and time consuming work due to the large volume of details that must be handled [11]. Because of this complexity, the required turnaround time cannot be met by manual design methods. In addition to the long design time, the chances of making mistakes using manual
layout techniques are quite high. Manual (custom) layout techniques lend themselves to the bottom up approach [12], with the result that the designer can become confused about the overall picture of the design. Hence there is a great necessity for automatic layout tools which produce results close to those produced by manual design methods in terms of silicon area and performance. Obviously there is a trade off involved in automating the design process. Using automatic layout techniques means sacrificing some amount of silicon area which can be saved by cleverly designing, dense custom layout. But as the complexity of the circuits increase, automating the layout becomes an increasingly attractive alternative.

The automatic generation of a cell is usually divided into two steps. First, a symbolic layout is generated, and then it is translated into mask shapes [7]. The symbolic layout is divided further into placement and routing, which are carried out separately, whereas translation to mask shapes is performed either by employing a compactor or by direct shape embedding. Module generators are a class of CAD tools which build some specific functional module of a circuit [17]. The user of the generator can customize the attributes of the module by specifying values for a set of parameters pertaining to that particular function. But the degree of control the user has over the output of the generator depends absolutely on what features the designer of the module generator has anticipated and provided.

The first published work suggesting a regular layout structure for ease in the automatic layout of random logic was that of Weinberger [25]. The layout style presented in [25] became known as the Weinberger array. Many additional layout styles have been suggested; most notable are the programmable logic array (PLA) [20], gate array, functional cell [24], and gate matrix [18]. The functional cell, gate array, and gate matrix layout styles receive the majority of attention from current researchers. In this thesis, we give most of our attention to a new layout style. Any truth table for a logic function can be mapped into a binary decision tree, which is effectively an N dimensional ROM. Recently Jullien et. al. have suggested a graph based reduction technique for pipelined logic, which allows efficient implementation of these trees with reduced transistor count [14]. The
combined technique of dynamically pipelining a reduced tree is referred to as a switching trees. We offer this switching tree approach as a design methodology, which can be applied to reduce local routing and silicon area for high throughput bit-level systolic arrays. This technique provides a minimized structure which lends itself to automatic logic function synthesis [6] and layout.

1.2 Thesis Organization

This work is organized into six chapters. The first chapter comprises this introduction. In chapter 2 the relevant background material for this thesis is reviewed. We discuss briefly the functional programming language SKILL, supported by the Cadence design environment, and discuss the concept of silicon compilation and the reasons for its growing importance in the domain of VLSI design. Since layout is the keyword in this thesis, various layout styles are also discussed in chapter 2 together with dynamic CMOS circuits and the significance of layout design rules for different technologies.

Chapter 3 is the heart of this thesis. We briefly look at the rules and definitions of Switching Trees introduced by Jullien et.al. [14]. We describe the VLSI realization of a switching graph, and a brief consideration has been given to the heuristic synthesis technique which is the prior work to this thesis. We introduce two layout styles for the switching trees; direct and indirect mapping. For both styles we explain, in detail, the rules and the procedure to implement a particular pipelined logic function from a topological description.

In chapter 4 we discuss the one-dimensional functional cell as a layout style, and we introduce an exact algorithm to find the Euler paths of a graph, which is the driving point of the layout style. An alternate algorithm to find the minimum number of metal tracks, which is the height of the layout, is also proposed and implemented. The Euler paths and track lists are used to implement a one dimensional layout style which is discussed in detail in this chapter.
Chapter 5 investigates the direct use of switching trees in pipelined arithmetic structures. A review of pipelining strategies is briefly given, followed by the descriptions of two recently proposed pseudo-single phase clocked latches. Some simulation results of various height trees in CMOS 1.2 and Mitel 1.5 micron technologies are presented.

The final chapter summarizes the overall research work accomplished, and also offers suggestions for future work.

A manual for the Switching Tree module generator is given in Appendix A. Appendix B shows the leaf cells and some automatically generated layouts. The SKILL codes are shown in Appendix C.
2.1 Introduction

The design of a VLSI chip is a complicated process requiring many intermediate steps, and several hierarchical levels of representation are used in the design process. One of the levels is the Register Transfer Level (RTL), where the variables and the data operators represent the hardware registers and functional blocks of the data-processing section of a chip. Next comes the logic level, at which the functional blocks are basic logic units called gates, such as AND, OR, XOR etc. At the transistor level, the logic gates are decomposed into transistor circuits, where a transistor is a four-terminal device with drain, source, gate and substrate. The lowest design level is the mask layout level, where transistors are represented as geometric figures with dimensions such as length, width and position. The geometric representation, or layout, of a basic logic unit is called a cell.

Each of the above design levels has behavioral, structural and geometric aspects. The Y chart of Gajski [8] illustrates these aspects in Figure 2.1. Multiple levels of abstraction are represented along each of the three axes. The levels of abstraction increase as one moves away from the vertex.
At higher levels, the behavioral aspects are more prominent, and the structural and geometric aspects are less important and less well defined. At the transistor level, all structures are explicit; the parts are low-level design components, such as transistors and wires, and the logical behaviour of each part is quite simple. At the layout level, structure and geometry are most important. At this level the geometric figures comprising each transistor and wire are assigned size and shape; each figure represents one of various materials used in fabricating an IC, such as diffusion, polysilicon, metal etc. Each of these materials occupies a different physical layer of the IC.

2.2 Topological Layout vs. Mask Layout

The structure to geometric mapping is a two-step process where the first step (usually called the topological layout) determines relative positions for all the structural elements and the second step (mask layout) determines the absolute positions of the elements.

Topological layout is an important part of the silicon compilation process since the need for more effective data description technique is becoming critical with the growing
complexity of VL-I circuits [15]. The most significant advantage of a topological layout is that it is technology-independent. Therefore, when design rules change, more compact mask layouts can be generated automatically using previously existing topological descriptions after the design specification file has been updated.

After the relative positions of the structural elements have been determined in the topological layout step, the next stage is the generation of the mask layout. The mask layout generator takes the topological layout as input and generates the absolute coordinates for the polygons realizing the devices and the interconnections. This generation is carried out using different placement and routing heuristics. The placement heuristics take care of the positioning of the devices and the routing heuristics are used for the interconnections between the devices. Unlike the topological layout, this phase of geometric representation is technology dependent; i.e. the area and the cost of the mask layout of any circuit will vary with any change of the technology.

2.3 SKILL Language

SKILL is a procedural design language. Its significance, however, lies not in its features for describing circuit designs (which it does fully support), but in the way it allows designers to access all the facilities that already are available on the host CAD workstation environment, with which it is tightly coupled. The use of a procedural language in conjunction with a graphics-based CAD system yields a combination that is more powerful and effective than the sum of all its components [17].

2.3.1 Interactive Language

SKILL is an interactive language. This is to say, that the user can enter a single SKILL statement and observe the result immediately. If the effect of the SKILL statement is to add a shape to a cell design, then that shape is immediately reflected on the pictorial representation of the design shown on the user's graphic screen. This fact in itself is of
significant benefit in allowing designers to develop SKILL procedures incrementally. For example the SKILL (in OPUS) instructions

\texttt{leHiCreateRect(), 0:0, 20:20}

have the effect of adding a primitive shape to the design currently being edited (with lower-left and the upper-right coordinates (0:0) and (20:20) respectively). If the three instructions are entered one at a time in the CIW (command interpreter window) interactively, the rectangle is immediately displayed on the graphics screen.

2.3.2 User Interface Features

SKILL provides a variety of powerful operations supporting interaction between a procedure and the user. For example \texttt{getpoint} (in EDGE SKILL) prompts the user to indicate a location on the screen by moving the cursor to the spot and clicking the mouse buttons. \texttt{hiCreateForm()} is used to solicit values of parameters from the user. Its effect is to bring up a form on the screen for the user to fill. SKILL also provides the capability for the procedure writer to present the user with a popup menu containing either pictorial symbols or text, or a mixture of both.

2.3.3 Computational Capabilities

SKILL is implemented as an extension of IL, a fully functionality, general-purpose programming language, embedded in LISP, but with more C-like syntax. This means that a full complement of control constructs and data types are available in SKILL. As well as the normal conditional and loop control functions found in all languages, IL supports powerful set-oriented control functions, such as \texttt{foreach}, which apply a function to each element in a list; or \texttt{setof}, which selects from a list of all elements of the list which satisfy a given condition.
2.3.4 Tight Coupling with the CADENCE system

SKILL extends IL by providing access to all the functionality of the host CAD system, so that any operation that can be executed interactively on the host may also be invoked procedurally from SKILL. For example, the instructions

```plaintext
ivHiDRC()

ivDRCOptionsForm->switches->value="do_drc"

hiFormDone(ivDRCOptionsForm)
```

will invoke the host system's design rule checker tool.

2.3.5 Design Database Access

Design database can also be accessed by SKILL. Any information in the database about objects and the relationship between objects can be obtained using SKILL. For example

```plaintext
dbCreateInst(cv inv nil list(0 0) "RO" 1)
```

will create an instance called `inv` at the reference point of 0:0 in the database `cv` with 0 degree rotation.

2.4 Technology

2.4.1 CMOS

CMOS (Complimentary Metal-Oxide Semiconductor) technology is recognized as a leading contender for existing and future VLSI systems. CMOS provides an inherently low power static circuit technology that has the capability of providing a lower power-delay product than comparable design-rule nMOS or pMOS technologies. A CMOS gate
consumes no DC power when the output is at '1' or '0' level, and 2N devices are required for an N-input gate for a fully complementary gate [21].

2.4.2 BiCMOS

BiCMOS technology is a combination of CMOS and bipolar technology. Both MOSFETs and Bipolar Junction Transistor devices are available on the same wafer, which means the advantages of both can be utilized at the expense of extra process complexity. Bipolar junction transistors offer high switching speed, high current density per unit area, and flexible I/O levels. Bipolar technologies have previously been used for small but fast system subcircuits, manufactured on separate substrates. In a BiCMOS process, both CMOS and bipolar devices are available on the same chip, thus affording the option of using both devices in an integrated design.

2.5 CMOS Dynamic Logic Families

When circuit delay is important, and where silicon area is at a premium, CMOS dynamic gates are used. Dynamic circuits are faster and need less area than the static CMOS circuits [22]. An important property of the dynamic CMOS gate is its ability to store charge in the parasitic capacitances for relatively large periods of time. In a dynamic circuit the pMOS block of the corresponding static CMOS circuit is replaced by a single pMOS precharging transistor. The advantages of this type of logic family are: smaller area, reduced load capacitance, higher speeds and easy pipelining [14]. Some of the disadvantages are that it only operates in a clocked system and it is more difficult to test than static CMOS. Also, clock timing is critical and for cascaded stages there is a chance of race conditions.

1. Node capacitances start to discharge before the correct inputs are established
2.6 Layout Styles

Digital integrated circuits are often classified according to their layout styles, which are either semi-custom, such as gate arrays and standard cells, or full-custom [19]. In this section both semi-custom and full-custom (unstructured methods, programmable logic arrays, gate matrix methods) styles are briefly discussed.

2.6.1 Semi-Custom Layouts

Gate arrays constitute a popular approach to the design and layout of digital circuits. A conventional gate array consists of rows and columns of cells separated by routing areas for wires interconnecting the gates. These cells are groups of unconnected transistors that can be wired together to form various types of gates, such as AND, OR, NOT, NAND, or NOR. Placement of the logic circuits onto the array and their interconnection can be performed automatically by CAD programs. Metal layers are added to the gate array to form circuitry within the cells and to interconnect these gates according to a particular design.

Gate arrays can be designed more quickly and at lower cost than standard cells. However, gate arrays are not as dense or as fast as standard cells or full-custom designs. They consume more power due to the larger capacitance associated with longer interconnections between gates, and the cost per chip is higher than that of other methods. Standard cells in the cell library have fixed height but they can vary in width depending on their complexity. Layouts designed with standard cells have greater density than those designed with gate arrays. This greater density is due to the increased compactness of the standard cells and the variability of the routing channel width.

2.6.2 Full-Custom Layouts

Unstructured (Hand Crafted) Method:
Unstructured layouts allow the placement of transistors in any location within a cell. In addition, maximum use of area can be achieved using this method as it allows the wires that interconnect the transistors to be routed in a random pattern. On the other hand the method is time consuming and subject to human error.

**Programmable Logic Arrays (PLA's):**

PLA is a widely used structured layout style. Its logical representation takes the form of a two-level, sum-of-products, multiple output circuit. Physically it consists of separate AND and OR arrays, with transistors placed at the grid points of the array. Each row of the AND array represents a logical product of various input signals, which are the columns of the AND array. These rows extend into the OR array, where the corresponding products are summed. The columns of the OR array then form the logical outputs of the PLA.

**Gate Matrix Layout:**

A gate-matrix consists of equally spaced vertical polysilicon columns that serve the dual role of transistor gates and general interconnections, and rows of diffusion and metal, with the intersection of horizontal diffusion and vertical polysilicon forming transistors. All layout information is supplied by the designer using a simple intermediate format. This information is passed to a gate-matrix compiler, which produces the final mask layout.

**Functional Cells:**

A cell is a layout of basic logic unit, such as a single gate. A functional cell can implement multilevel logic of arbitrary depth, although this depth is usually limited by performance considerations. The styles for one dimensional functional cells\(^1\) have been discussed in greater detail in chapter 4. Figure 2.2 shows the correspondence between the circuit symbol of a transistor and its corresponding layout in a cell.

---

1. A one dimensional functional cell is an array of transistors in which all drain and source terminals of transistors of a given type, either pMOS or nMOS, lie along a single row of diffusion.
**Switching Tree Layout Style:**

The technique for mask layout of switching trees is similar to the gate-matrix layout technique. The mask layout of the switching tree is the same basic design as the Gate-Matrix layout, only rotated by 90 degrees. The gate-matrix layout has a continuous polysilicon path connecting a vertical path of transistors, while the switching tree layout has a metal2 overlapped with polysilicon path connecting a horizontal path of transistors. Depending on the height and width of the binary tree two different approaches will be followed to layout a particular design. The entire process and the style of the switching tree will be discussed in Chapter 3.

### 2.7 Layout Design Rules

Layout rules, referred to as design rules, can be considered as a prescription for preparing the photomasks that are used in the fabrication of integrated circuits [21]. The rules provide a necessary communication link between circuit designer and process engineer during the manufacturing phase. The main objective associated with the layout rules is to obtain the circuit with optimum yield in as small a geometry as possible without compromising reliability of the circuit. These rules represent a tolerance that ensures very high probability of correct fabrication and subsequent operation. The more aggressive the rules are, the greater the probability of improvements in circuit performance.
There are several approaches that can be taken in describing the design rules. These include 'micron' rules stated at some micron resolution, alpha and beta rules, and lambda rules. Micron design rules are usually given as a list of minimum feature sizes and spacings for all the masks required in a given process. Table 2.1 compares some of the micron rules for CMOS4S 1.2, MITEL 1.5 and BiCMOS 0.8 technologies.

<table>
<thead>
<tr>
<th>minimum spacings</th>
<th>CMOS 1.2&lt;sup&gt;b&lt;/sup&gt;</th>
<th>Mitel 1.5</th>
<th>BiCMOS 0.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>minimum gate (poly) length</td>
<td>1.5</td>
<td>1.5</td>
<td>0.8</td>
</tr>
<tr>
<td>minimum metal1 width</td>
<td>2.5</td>
<td>1.5</td>
<td>0.8</td>
</tr>
<tr>
<td>minimum metal2 width</td>
<td>3</td>
<td>2</td>
<td>0.8</td>
</tr>
<tr>
<td>contact (min. length x width)</td>
<td>1.5 x 1.5</td>
<td>1.5 x 1.5</td>
<td>0.8 x 0.8</td>
</tr>
<tr>
<td>minimum poly spacing</td>
<td>2</td>
<td>1.5</td>
<td>1</td>
</tr>
<tr>
<td>minimum metal1 spacing</td>
<td>2</td>
<td>1.5</td>
<td>1.2</td>
</tr>
<tr>
<td>minimum metal2 spacing</td>
<td>2.2</td>
<td>1.8</td>
<td>1.2</td>
</tr>
<tr>
<td>via (minimum length*width)</td>
<td>2 x 2</td>
<td>1.8 x 1.8</td>
<td>0.8 x 0.8</td>
</tr>
</tbody>
</table>

<sup>a</sup> All dimensions are in micron
<sup>b</sup> 1 design micron = 0.8 actual micron

2.8 Summary

We have mentioned the advantage of using SKILL to provide a powerful and a supportive environment for the development and use of a module generator. This language has been used to implement the switching tree and functional cell generators, discussed in the next two chapters. An overview of various layout styles including the switching tree has been given and the relation between a topological layout and the mask layout has been illustrated. Finally the layout design rules and the importance of them in producing an integrated circuit layout have been discussed.
Chapter 3

The Switching Tree

3.1 Introduction

The design of a VLSI chip is a complicated process requiring many intermediate steps. The lowest level in the design process is the layout, where transistors are represented as geometric figures with attributes such as length, width, and position. This chapter is concerned with the translation of a design from the transistor circuit level to the layout level. Figure 2.2 shows the correspondence between the circuit symbol of a transistor and its corresponding layout. The conducting path between the drain and source terminals is open or closed depending upon the voltage level on the transistor's gate terminal; thus a transistor acts as an on-off switch. A transistor can be formed by a vertical polysilicon over a horizontal diffusion run. For each technology, specific design rules have to be satisfied.

In pipelined systems for limited throughput rate DSP applications, it is sometimes useful to consider trading off speed for greater functionality within the pipeline stages, and reaping the benefits of reduced area and power consumption [13]. Recently the VLSI group at the University of Windsor dealt with the implementation of pipelined logic blocks, suitable for high throughput arithmetic
arrays, using complex, merged trees operating within a true single phase dynamic latch. The technique has been referred to as implementation by switching trees, which are dense multiple output NFET blocks based on minimized binary trees. A step by step procedure to implement the switching trees in silicon is discussed in detail in this chapter. The output from a minimization algorithm [6], which is called the topological matrix, is used to drive the module generator for an arbitrary switching tree design. The network in this case is a direct mapping from the switching graph. The module generator has been developed using the SKILL language in the Cadence design environment.

3.2 Previous work

In 1992, Jullien, et.al. [14] introduced a building block for realizing switching functions called switching-trees\(^1\). Later he developed a topological layout matrix for describing the symbolic layout of the switching graph in a technology independent fashion. A Switching Graph Synthesizer, an abstract set of heuristics for converting a switching function to a switching-graph was implemented in 1993 [6]. The synthesizer takes input from the user in terms of essential minterms and don't care minterms. The output of the synthesizer is a switching-graph specified using a hypergraph model and a list of nodes representing all possible relative position of nodes in the VLSI layout. An algorithm was also developed to convert the abstract switching-graph into a topological layout matrix. This matrix is an array of ASCII characters representing transistors and wires connecting them. The symbols are placed in the matrix in such a way that the relative position of each symbol in the matrix is exactly the same as the relative position of the corresponding transistors or wires in the mask layout.

\(^{1}\) Also referred to as Switching Graphs

---

The Switching Tree  Previous work  16
3.3 Pipelined Switching Trees

A pipelined switching function may be realized by the pipelined switching graph. A path from the evaluation node to the ground is created when the Boolean variables represent a minterm of the switching function.

The switching tree, which is a complex NFET logic block is embedded in a single-phase CMOS master-slave latch as shown in Figure 3.1. The bottom of the tree is a common connection to the ground switch. The tree is designed as an N dimensional ROM (binary tree) where N is the number of input variables as shown in Figure 3.2.

![Figure 3.1 Pipelined switching tree](image)

3.4 Graph Based Reduction

A graph $G(X,V)$ is a structure which consists of a set of vertices $V=\{v_1,v_2,...\}$ and set of edges $X=\{x_1,x_2,...\}$. A tree can be represented by a graph where $X$ is a set of edges ($n$ channel transistors) $\{x_{i,j}\}$ and $V$ is the vertex set of nodes $\{v_{i,j}\}$. An edge $\{x_{i,j}\}$ is
defined by connection type $ct \in \{g,r,W\}$ and position $(i,j)$, where $i$ is the height of the edges in the tree, $i \in \{0,1,\ldots,N-1\}$, and $j$ is the position of the edge across the tree, $j \in \{0,1,\ldots,2^{i+1}-1\}$; $N$ is the tree height. The inputs to the tree are logic levels $g_i \in \{0,1\}$, $i \in \{0,1,\ldots,N-1\}$. If $g_i = 1$, then the path takes edge $g$ if it is present, and if $g_i = 0$, then the path takes edge $r$ if it is present. $W$ represents an edge which corresponds to a wire, or link, connection, and is only present following the successful application of the reduction rules [13].

A path $P_{(i,j)(k,l)}$ is the connection from node $v_{(i,j)}$ to node $v_{(k,l)}$, constructed by edges. A full path connects node $v_{(0,0)}$ to node $v_{(N,1)}$. A switching tree is the reduction of a unique set of full paths that describe a logical function. The tree is characterized by two sets of full paths, a true set in which an edge $g$ or $r$ is present at the $(N-1)$th level, and a complement set in which the opposite $g$ or $r$ is removed at the $(N-1)$th level. A truth table is mapped onto a full tree by removing a subset of edges $\epsilon \{x_{N-i,j}\}$, $j \in \{0,2^N-1\}$, from a full tree based on the set of 0's or 1's in the truth table to obtain the true or complement trees. There are two rules which are used in the graph reduction technique [13]. These two rules to reduce the graph are valid for both single and multiple output functions.

Figure 3.2 A binary tree

![Binary Tree Diagram](image)
Rule: 1 *Merging of shared sub-trees*

If paths $P_{(i, j) (N, l)}$ and $P_{(i, k) (N, m)}$ contain an identical set of edges, starting at a node at level $p$, for two sequences of nodes where the matching occurs in both sequences, the sequences can be merged at the $p$th level. Furthermore, if $k=j$ and $i-p=1$, then the edges from node $v_{i, j}$ to node $v_{p, l}$ can be replaced by a link edge $W$. Figure 3.3 shows the application of this rule.

![Figure 3.3 Merging of shared sub-trees](image)

A merge is beneficial if:

1) Two nodes are adjacent to each other i.e if the merge is a local merge.
2) The merge does not involve additional routing area.
3) Expected saving of silicon area is more than the expected cost in terms of additional routing area [6].

Rule: 2 *Deletion of Common Edges*

Consider a set of paths, $\{P1\}$, connecting a node $v_{i, j}$ at level $N$, and a set of paths $\{P2\}$ also connecting the node $v_{i, j}$ at the same level. If $\{P2\}$ covers $\{P1\}$, then the edge from
node $v_{i,j}$ in $\{P1\}$ is changed to a link edge $W$. If $\{P1\}$ covers $\{P2\}$, then the edge from the node $v_{i,j}$ in $\{P2\}$ is changed to a link edge $W$. In Figure 3.4(a), $P2$ is a subset of $P1$, and so, $P2$ has been replaced by a metal wire, as shown in Figure 3.4(b).

**Figure 3.4 Deletion of common edges**

![Diagram](image)

Rule 2 will reduce the number of transistors; however, this action does not necessarily minimize the layout area. Each node at any level may generate at most 2 nodes at the next level, each connected by an edge labeled by $x$ or $\bar{x}$. For example, in Figure 3.5(a), two nodes at level $N$ are produced from the parent node $v_{i,j}$. After applying rule 2 there will be an alternative form for the structure given in Figure 3.5(b). The alternative form is possible due to the fact that in Figure 3.5(a), the set of minterms corresponding to node 2, is a subset of the set of possible minterms corresponding to node 1. Some don't cares are implicitly generated here. For example in Figure 3.5(b), node 1 has minterm 5 as an internally generated don't care. The advantage is that it allows more possibilities for sharing of subnetworks for nodes representing similar Boolean functions.
3.5 Construction of the Switching Tree

In order to explain the basic construction process, we use, as an example, a circuit that computes the SUM bit of an adder.

In Figure 3.6 the on set (set of minterms) is \([1,2,4,7]\) and the don't care set is the null set.

Figure 3.6 The truth table of the sum bit of a full adder

<table>
<thead>
<tr>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>in1</td>
<td>in2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Figure 3.7 Switching graph of the sum bit of a full adder

If the number of inputs in the design is $N$, at any level $m$, any true value of the child from the parent node $[a_1, a_2, a_3 \ldots a_n]$ will be $[a_i] - 2^{N-m}$; $i = 1, 2, \ldots n$ and the complementary value of the child will be any $[a_i] < 2^{N-m}$; $i = 1, 2, \ldots n$. Any element of the true or the complimentary node is either zero or any positive value.

**Level 1**

$x'_1$ is the transistor with the complementary input.

$x_1$ is the transistor with true input.

Referring to Figure 3.7 on page 22, the root's complementary value is $[1, 2]$ (the elements of the set $[1, 2, 4, 7]$ which are less that $2^{3-1}$, where 3 is the number of inputs) from node $[1, 2, 4, 7]$.

The true value is $[0, 3]$ (the difference between $2^{3-1}$ and the elements of the set $[1, 2, 4, 7]$)

**Level 2**
$x_2'$ is the transistor with complementary input.

$x_2$ is the transistor with true input.

The complementary value from node [1,2] is [1]

The true value from node [1,2] is [0]

The complementary value from node [0,3] is [0]

The true value from node [0,3] is [1]

Notice that at Level 2, transistors have been flipped and merged.

Figure 3.8 The graph and the corresponding full adder circuit (only the sum bit)

(a) The graph of Figure 3.7 and the corresponding topological matrix

(b) The corresponding circuit

Level 3

$x_3'$ is the transistor with complementary input.

$x_3$ is the transistor with true input.

The complementary value from any node [0] is [0]
The true value from any node [1] is [0].

Figure 3.8(a) shows the graph of Figure 3.7 in Manhattan\textsuperscript{1} style with the corresponding topological matrix. The edge with true value is represented as 'G' and the edge with complementary value is represented as 'R'. 'W' indicates the connection between the transistors. 'b' is any blank element. Figure 3.8(b) shows the corresponding schematic of the circuit.

3.6 Switching Tree Layout Styles

As mentioned in chapter 2, the technique for mask layout of switching trees is similar to the gate-matrix layout technique. A switching tree along with its mask layout has been presented in Figure 3.9. Several similarities become apparent while comparing this layout. One similarity is that the mask layout of the switching tree is the same basic design as the gate-matrix layout, but rotated by 90 degrees.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{switching_tree_layout}
\caption{A simple switching graph and the mask layout}
\end{figure}

1. The edges are aligned vertically or horizontally
The gate-matrix layout has a continuous polysilicon path connecting a vertical path of transistors, while the switching tree layout has polysilicon path connecting a horizontal path of transistors. Metal2 is placed over every horizontal polysilicon and proper connections are made between them. The reason that metal2 is used is that the length of the common connection path in typical switching tree layouts, is much greater than the path in the gate-matrix layout.

We propose two different layout styles for the switching trees depending on the height and the width of the tree. They are the Direct Mapping Style (DMS) and the Indirect Mapping Style (IMS).

Figure 3.10 Floor plan for the Direct Mapping Style

```
   L
 P  CS  N
```

CS=Core of the design  
P=Half_p_inverter  
N=Half_n_inverter  
L=Latch(es)

Figure 3.10 shows the floorplan for the DMS while Figure 3.11 shows the floorplan for the IMS. A floorplan is a block of rectangular tiles, representing the circuit, where each tile represents a cell. For DMS the floorplan has only four rectangular tiles. The inverters at the input stage are represented as the $n$, $p$ cells while the core of the design and the latches at the output stage are the CS and L cells. For the IMS technique, it is a little more complicated; the core is divided into two different sections and there is a routing channel between them (see section 3.8). Two corner cells are placed at the upper two ends of the
design, while the latches and the inverters (shown together in one cell in Figure 3.11) are placed between the corner cells and the upper section of the core.

Figure 3.11 The floorplan for the Indirect Mapping Style

<table>
<thead>
<tr>
<th>C</th>
<th>I &amp; L</th>
<th>U</th>
<th>I &amp; L</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

L=Lower section of the core
U=Upper section of the core
C=Corner cell
R=Intermediate routing channel
I & L=Inverters and Latches

3.7 Symbolic Layout (Topological Matrix) to Mask Layout

The following algorithm generates a mask layout from the topological description. Figure 3.12 diagrams the procedure.

Step 1: Load the topological matrix. This matrix gives the relative placement of the transistors and also the connectivity between them (Figure 3.13). The row with 'G's or 'R's is the transistor row. This is the nth row of the matrix, where n is any even number. Any 'W' present in this row is the vertical connection between the source and the drain of two different transistors. The (n-1)th row is the horizontal connection row (HC row). 'W's in HC rows denote the connections between the drains or the sources of two different transistors. 'b' in any row indicates a nonexisting (blank) element.
Figure 3.12 Flow chart to implement the core section of the tree

- **start**
- Load the matrix
- Divide the T set in TR and TG rows
- Run the poly and metal/2
- Scan the matrix.
  - Place horizontal metal for W in HC row
  - Place vertical metal for W in T rows
  - Place transistor for R or G in TR or TG row
- Place the output pin for W followed by b in the first row. Place the inverters and latch(s).
- **stop**

TR row -> row of transistors with complementary input
TG row -> row of transistors with true logic input
T set -> TR row + TG row
W -> wire
G -> transistor with true input
R -> transistor with complementary input
b -> blank

Figure 3.13 The topological matrix

\[(WWbb \quad RbGb \quad WbWb \quad GRGR \quad bWbb \quad RbGR \quad WWWb \quad bbbR)\]

Step 2: Each pair of rows from the top of the matrix provide the transistor information, with true or complementary inputs, and also the connectivity between them. The first step is to provide a blank space between any two elements in each row of the matrix. The next step is to divide each transistor row (T row) into two individual rows of transistors, where one row consists of transistors with only true input (TG...
row) and the other row will have only complementary inputs (TR row). A T set comprises a TR row and a TG row. In the TR row the transistors will be of type 'R' and in the TG row the transistor type is 'G'. The relative positions of the 'G's and 'R's in the TR or TG row will be identical to the original positions of the 'G's or 'R's in the T row. Some additional elements are added in each row to provide short circuits. Short circuits are the connections between horizontal polysilicon and metal2, which will be explained later. These extra elements are denoted as 'X's. These short circuits are added at the beginning, at the end and after every eight elements in each row to provide proper connections between the polysilicon and the metal2. This will prevent the possible delay of the signal to the gates of the transistors in the same row. Since there are only four elements in each row in our example, there will not be any 'X's in the row other than the first and last. After this step the topological matrix becomes as shown in Figure 3.14. The length of each row in the matrix remains the same as in the original row. If there are N number of TR rows in the original matrix, after dividing each row into two, the number of transistor rows in the modified matrix becomes 2 × N. So, the total number of rows in the matrix is 2 × N + H, where H is the number of HC rows in the matrix. A few things are worth mentioning here: the sequence of the transistor rows (TG and TR) in each T set is opposite to that of the following T set; the last T row of the matrix is always a TR row of ground switches.

Figure 3.14 The Matrix after modification

(X W W b b X
X b b G b X
X R b b b X
X W b W b X
X b R b R X
X G b G b X
X b W b b X
X b b G b X
X R b b R X
X W W W b X
X b b b R X)
Step 3: A polygon of a polysilicon, overlapped with metal2, is placed for each TR or TG row, as shown in Figure 3.15. The length of this double layer polygon is the product of the number of elements in a row and a constant. This constant depends on the technology and the process used for generating the modules. A short circuit, for each X in the TR or TG row, is placed to make connections between the polysilicon and the metal2. The number of the polysilicon or metal2 layers is identical to the number of transistor rows in the matrix (7 in this example).

Figure 3.15 The polysilicon with metal2

Step 4: The matrix is read from the first element of the first row. The first row is always a HC row and a horizontal metal (metal1) is placed for any 'W' in the row as shown in Figure 3.16. If a 'W' is followed by an 'X' an extended horizontal metal is placed. The same rule is applied for 'W' in any HC row in the matrix. The metal is always extended towards right from the starting point. This phase is referred to as the horizontal routing phase.
Step 5: While scanning the TG row of a T set, an NFET transistor is placed for any 'G' in the row. If this row is the first row of the T set, then a vertical metal polygon (metal1) is abutted with the source of the transistor to make a connection with the drain of the transistor below. For any 'R' in the TR row of this T set, a vertical metal polygon is connected with the drain of the transistor to connect the source of the transistor above (Figure 3.17).

Figure 3.17 Transistors in different rows

(a) transistor in the lower row
(b) transistor in the upper row
The opposite happens if the TR row is above the TG row in the T set. Figure 3.18 shows the layout of the full adder (only the sum bit) with the corresponding topological matrix. The source of the transistor in the upper row is abutted to the drain of the transistor in the lower row, where necessary, and vice versa.

**Figure 3.18 The core of the sum bit of the full adder**

![Diagram of the core of the sum bit of the full adder]

Step 6: If a there is a 'W' followed by a 'b' in the very first row of the matrix or if there is a transistor (G or R) in the second row and the drain of the transistor is floating, a via with metal1 and metal2 is placed, which is one output of the design. This output is connected to the input of the latch for pipelining purpose. The number of outputs is equal to the number of output bits and is usually more than one. Ground switches are located at the bottom of the tree. When the sources of the transistors on the very last row are merged, a ground switch is added to connect to the VSS (ground) line. The number of ground switches is more than one, for anything other than a very simple single output design. Each input is fed through the inverters to...
obtain the compliment of each input signal; the number of inverters is clearly the same as the total number of inputs. Each inverter is designed as a half-p-invert (p transistor of the inverter) and a half-n-invert (n transistor of the inverter) which exist at two different ends of the tree (Figure 3.19).

Figure 3.19 Complete layout for the sum bit of a ripple carry full adder with the latch and the inverters

All the gates of the transistors in the circuit are fed by primary inputs. As the T sets in the original matrix are split into TR and TG rows, the inputs are also split into true and complement values. As mentioned earlier, the sequence of the rows in a T set will be opposite to that in the previous T set. For example, if the sequence is TR and TG in one T set the sequence for the next T set will be TG and TR. So the adjacent inverters at the input stage will also have opposite sequence of input and output pins.

The half-p-invert, half-n-invert and the master-slave latch (these are the leaf cells) are hand-crafted cells, placed simultaneously with the core of the circuit by the module generator. One has to be careful while laying out the inverters because the polysilicon of any inverter should match exactly with the horizontal polysilicon of the tree. The internal routing for the latch has to be carried out entirely with metall1, the second metal layer.
(metal2) can be used for the routing the clock line. This metal2 routing should be placed as closely as possible to the VDD line, to avoid any extraneous connection while performing the global routing for the entire design.

The above procedure is referred to as direct mapping. An operating manual for this module generator tool is available in Appendix A; Appendix C details the SKILL code, and some layouts, along with the topological matrices, are demonstrated in Appendix B.

3.8 Problems with DMS and a solution

In a switching tree, most of the minimization (merging of common nodes and deletion of common edges) takes place at the bottom of the tree with little change to the upper portion. For this reason, to maintain regularity throughout the tree, a large portion of the layout area remains unused. In this section we propose a solution for this problem and will try to utilize the wasted area in the upper section of the tree. Our goal is to make use of the wasted area to fit both the output stage latches and the input inverters. Also, using this approach we expect to get a more square aspect ratio. We refer to this approach as Indirect Mapping Style (IMS).

Figure 3.20 Layout of the mod17 multiplier (one output)

Four extra steps need to be performed to modify the topological matrix in order to get a different placement of the transistors in the upper portion of the tree. Figure 3.20 shows
the layout of one of the five outputs of a mod17 multiplier. We will use this layout as an example to describe the IMS algorithm.

Step 1: The topological matrix is divided into two different sections. The matrix is split in such a way so that there are 6 rows in the upper section and N-6 rows in the lower section where N is the total number of rows in the original matrix. Figure 3.21 shows the upper section of the layout and Figure 3.22 shows the lower section.

Figure 3.21 Upper section of the layout

Figure 3.22 Lower section of the layout

Step 2: The lower section remains the same as before.

Step 3: Any two adjacent transistors in the upper section are placed in such a way so that they maintain a constant distance (5 elements) in a specific row. For example, if there are N transistors in one row the distance between the first and the last transistor will be (N-1) × 5 units, where one unit is the distance between two adjacent elements in the original matrix. This way the number of W's and b's between the transistors in the T rows is reduced. The HR row above a T row is shrunk in the same manner. After modification of the matrix, the transistors in T rows should always have the same relative positions as in the original matrix; i.e. they should
share the same column before and after the modification. Figure 3.23 shows the corresponding layout of the matrix after modifications.

**Figure 3.23 Modified upper section**

Step 4: Necessary connections are made between the lower and upper sections.

A vertical metal2 line is passed from the end of each T row in the lower section to connect the inverters. This metal2 is placed to avoid short circuits between the horizontal metal2 lines. There are two corner cells at the two ends of the upper sections (Figure 3.24).

**Figure 3.24 Corner cell**

For the indirect layout style, the inverters and the latches are placed in such a way so that the use of area can be maximized. Three inverters are placed at the left of the upper section to drive the gates of the transistors in the upper section, as the number of the T sets in the upper section is three (six TR rows). The rest of the inverters are split in two sets. One set is placed beside the left corner cell and the other set is placed beside the right corner cell.
The latches are also divided in two sets. One set is placed between the two sets of inverters at the left of the upper section and the second set is placed on the other side (right side) of the upper section.

The number of vertical metal2 wires on both sides of the structure depends on the number of TR and TG rows in the lower section. Let \( x \) be the number of TR and TG rows. If \( x-1 \) is even and divisible by 4, the number of metal2s on the left side will be \( \frac{(x-1)}{2}+1 \). If \( x-1 \) is even and not divisible by 4, the number of metal2s on the left side will be \( \frac{(x-1)}{2}+2 \). So the number of vertical metal2s on the left side will always be odd and the number on the right side will always be even (odd including the line for the clock).

The connections between the two sections are also divided into two sets. One set will connect the left portion of the upper and lower sections, called the left set. The other set connects the right portion, called the right set. The number of metal wires in each section depends on the number of metal wires going up from the lower section (or the number of metal wires coming down from the upper section). Always the first two metal wires in the left set and the last two metal wires in the right set are metal1 to avoid possible short circuits between the metal wires in the corner cell. The rest of the metal wires are in pairs with each pair consisting of coincident wires of metal2 and metal1. Table 3.1. gives the number of metal wires in the left set. The number of metal wires in the right set is equal to the difference between the total number of metal wires and the number of metal wires in the left set.

<table>
<thead>
<tr>
<th>( y^a )</th>
<th>( y ) divisible by 4</th>
<th>( y-1 ) divisible by 4</th>
<th>number of metals on left set</th>
</tr>
</thead>
<tbody>
<tr>
<td>even</td>
<td>yes</td>
<td>N/A</td>
<td>( \frac{(y-2)}{2}+2 )</td>
</tr>
<tr>
<td>odd</td>
<td>N/A</td>
<td>yes</td>
<td>( \frac{(y-1)}{2}+1 )</td>
</tr>
<tr>
<td>even</td>
<td>no</td>
<td>N/A</td>
<td>( y/2 )</td>
</tr>
<tr>
<td>odd</td>
<td>N/A</td>
<td>no</td>
<td>( \frac{(y-1)}{2} )</td>
</tr>
</tbody>
</table>

a. \( y \) = total number of metal wires
These corner cells act as links between the vertical metal2s and the horizontal metal2s, which connect the inputs and the outputs of the inverters. For intermediate connections between the two sections there are overlapping metals (metal1 and metal2) running horizontally between the two regions. Since there are no vias, there will not be any short circuit between them. At the tip of each metal2 there is a via to connect the vertical wires within the region which are metal1s. Although some additional space is needed for the intermediate connections, the area will be saved by 10-15% comparing to that of DMS for higher switching trees. Figure 3.25 shows the modified layout (inverters and the latch are shown in the black box). For each layout (direct or indirect) short circuits (connection between horizontal polysilicon and metal2) are placed in each row at certain intervals.

Figure 3.25 Modified layout (with latches and inverters)

So far the layout style has been described with the assumption that the latch that is used at the output stage is a voltage sensing master-slave. There will some change in the style if the latch uses current steering. The ground switches for pull-down is not required, because the output is produced by sensing the current from the tree. This output is a voltage of either VDD, representing logic high, or VSS (zero), representing logic low. The line which was VSS in the case where the master-slave latches were used, will now be the VDD line and the inverters for driving the transistors are placed in an inverted position.
3.9 Area Calculation

3.9.1 Direct Method

Width of the cell = total number of rows in the original matrix × row spacing

Length\textsubscript{1} of the cell = total number of columns (total number of elements in one row) ×
\hspace{1cm} \text{column spacing} + \text{the length of an inverter}

Length\textsubscript{2} of the cell = total number of latches (number of outputs) × the length of a latch

Area of the cell with inverters and latches = Width × Length\textsubscript{1} (if Length\textsubscript{1} > Length\textsubscript{2})
\hspace{1cm} = Width × Length\textsubscript{2} (if Length\textsubscript{2} > Length\textsubscript{1})

3.9.2 Indirect Method

Width of the cell = total number of rows in the original matrix × row spacing +
\hspace{1cm} \text{Interm\_met\_width}

Length of the cell = total number of columns (total number of elements in one row) ×
\hspace{1cm} \text{column spacing}

Area of the cell with the inverters and the latches = Width × Length

\textbf{Interm\_met\_width}

Interm\_met\_width is the width of the routing channel between the upper and the lower
section. We present here the pseudo code to find the width of the routing channel ( "" indicates comment in the code).

```
count=0 ; initialize
if( 'W' at the nth column followed by 'b' at the 7th row, then
  count=count+1
  go to end )
else if ( 'G' or 'R' at the nth column followed by 'b' at the 8th
  row && no 'W' at the (n-1)th column of the 7th row, then
```
count=count+1) ; n= 1,2...total number of elements in a row
Intern_met_width = (count * (minimum metal2 width + minimum metal2
spacing))/4

3.9.3 Which method should be used

Since there are two layout styles to implement the logic, the user has to decide which of
the methods to use. An obvious solution is to run both methods and then keep the layout
with the lower area. Rather than running both programs, the following algorithm can be
used.

**Decision Algorithm:**

lat_length = total number of latches the width of each latch.

inv_length = total number of inverters x the width of each inverter.

upper = the width of the upper section.

col_spa = spacing between two columns.

row_no = no of rows in the matrix/2.

col_no = total number of columns.

pseudo code:

```plaintext
if( lat_length + inv_length + row_no*col_spa > col_no*col_spa-upper,
    then
        'USE DIRECT METHOD'
    else
        'USE INDIRECT METHOD'
); end if
```
3.10 Technology File

Unlike the topological layout, the mask layout is dependent on the technology [18]. The cost and the area for the mask layout of any circuit is dependent on the technology being used to realize the circuit. A mask layout generator always takes a technology file as its auxiliary input. This technology file is a database containing information on the minimum allowed distance of separation between the different mask rectangles, minimum overlapping distance allowed between them, names of the layers which realize the mask rectangles and various technology-related rules that govern the error free layout generation. Three technology files, CMOS 1.2, Mitel 1.5 and BiCMOS 0.8 micron, have been introduced for the automatic generation of layouts using switching tree style. A sample technology file, Mitel 1.5 micron, included in Appendix.C. Table 3.2. shows the dimensions used for the above three technologies.

**Table 3.2. Dimensions for different technologies**

<table>
<thead>
<tr>
<th></th>
<th>CMOS 1.2</th>
<th>Mitel 1.5</th>
<th>BiCMOS 0.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>row (poly) spacing</td>
<td>10.7</td>
<td>8.7</td>
<td>5.8</td>
</tr>
<tr>
<td>transistors (l x w)</td>
<td>1.5 x 4.1</td>
<td>1.5 x 4.1</td>
<td>0.8 x 2</td>
</tr>
<tr>
<td>horizontal metal (l x w)</td>
<td>10.1 x 2.5</td>
<td>8.1 x 2</td>
<td>4.5 x 1</td>
</tr>
<tr>
<td>extended horizontal metal</td>
<td>17.7 x 2.5</td>
<td>14.2 x 2</td>
<td>8.2 x 1</td>
</tr>
<tr>
<td>(l x w)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>column spacing</td>
<td>7.6</td>
<td>6.9</td>
<td>3.5</td>
</tr>
<tr>
<td>HC and T row spacing</td>
<td>5.8</td>
<td>4.6</td>
<td>3</td>
</tr>
<tr>
<td>vertical metal spacing</td>
<td>38</td>
<td>30.5</td>
<td>17.5</td>
</tr>
<tr>
<td>between upper and lower section</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>metal spacing in the corner cell</td>
<td>5.5 x 6.9</td>
<td>5.1 x 4</td>
<td>2.6 x 2.6</td>
</tr>
<tr>
<td>(vertical x horizontal)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>metal (&quot;W&quot;) length in the T row</td>
<td>12.5</td>
<td>11.6</td>
<td>7.4</td>
</tr>
</tbody>
</table>

a. l x w --> length x width

1. Dimensions are in microns
For the switching tree layout approach, inverters are used to feed the inputs and latches are placed at the output stage for pipelining purposes. These cells are laid out manually for different technologies. To get the leaf cells (inverters and the latches) along with the core of the design in the final mask layout, the height and width of these cells should also be included in the technology file and the relative positions of the input/output pins in the cells should be indicated. For the IMS, the height of the inverters and the latches should be equal to the height of the upper section and the input to the voltage sensitive latch (specially for DMS) should be placed as close as possible to the VDD (VSS for the current steering latch). This will ensure proper connectivity between the core of the design and the latches; especially when the number of latches in the design is high. Figure 3.26 shows the flow diagram of the entire process for the switching tree layout.

Figure 3.26 The entire switching tree process

- Truth Table
- Switching Graph Synthesizer
- Topological Matrix
- Technology Independent SKILL Program
- Automatic Placement and Routing
  - Manually laid out Latches and Inverters
  - Mask Layout
  - Technology File
3.11 Concluding Remarks

The switching tree layout style has some advantages over other layout styles. Any logic can be implemented using this approach. Since it consists of NFET logic blocks only, the overall area is much smaller than other methods because of the absence of its PFET counterpart. For gate arrays, longer interconnections will cause larger capacitances, and the cost per chip is higher. For the standard cell method, the cost of designing and maintaining a cell library is high. Although the hand crafted full custom layout will give the most compact and probably efficient layout, it is very time consuming and subject to error. The standard PLA structure also has some limitations. It does not handle logic functions in non sum-of-products form [19], and the outputs of the PLA must be routed back to the input to implement such function, which is wasteful in area and speed. Such problems do not exist in switching trees. Except for the intermediate connections between the two sections in IMS, other node connections are made by shortest possible metals and the diffusions of the transistors are abutted where necessary. The total number of transistors in the cell is reduced with the switching-graph synthesizer [6]. Since multiple layers of metal are used in the layout, the parasitic resistance and capacitance are low.

The switching trees are based on dynamic logic, so a uniform clock distribution throughout the cell is very important. A true single phase or a pseudo single phase (which can be made a true single phase by a local inverter) clocked latch should be used at the output stage of the cell to reduce the clock skew. For trees of larger heights, charge sharing becomes a serious problem where a large number of transistors are connected in series. Usually, as the tree becomes higher, the module generator tends to produce layouts although efficient in terms of area but the width/length ratio of the cell becomes fairly large, which means a poorer aspect ratio.

3.12 Summary

The programs written both in Edge SKILL and OPUS SKILL, are capable of producing layouts, from any topological matrices of any dimension, in 1.2 micron and 3 micron NT
CMOS, 0.8 micron BiCMOS and 1.5 micron MITEL CMOS technology. The program has been written in technology independent form so that new technologies can be incorporated at a future date. It has been observed that there is a linear relationship between the run time and number of transistors in the network i.e in the topological matrix. The topological matrix (symbolic layout) of a switching tree can be obtained from the synthesizer [6], and this matrix can be used directly as the input to this module generator to get the final layout. For high trees we have introduced a new approach to obtain a smaller layout area. Although the module generator is user-friendly, the end-user does not have much control over the generator's design technique. This is a drawback of the program. We believe that an integration of language and the graphics-based tools of the host CAD system can provide the most effective use of the particular expertise of a skilled circuit designer.
Chapter 4
The Functional Cells

4.1 Introduction

In chapter 3, we used the switching tree layout style to build a module generator to automatically produce a layout from a topological matrix. In this chapter, we discuss another layout style and propose an algorithm based on graph theory to generate the layout automatically. An important layout task is to place and orient the transistors on the chip in such a way as to minimize the layout area of a circuit. Connections between transistor nodes are usually performed using metal as routing material, and in most technologies there exists more than one metal layer. The greater the number of metal layers, the simpler the routing problem.

Our goal is to produce an automatic layout generator for arbitrary MOS transistor networks. Recently, researchers have given much attention to this topic for the series-parallel and dual circuits, i.e. when the P and the N subcircuits are geometric duals of each other. In our case we do not assume such constraints. We propose a method which is capable of producing layouts of any arbitrary dynamic CMOS circuit. Static circuits can also be laid out using this approach provided some conditions are fulfilled; as will be discussed later in this chapter.
4.2 Graph Theoretic Background

Given an undirected multigraph \( G(v, e) \) with a set \( v \) of vertices and a set \( e \) of edges, a path \( p \) in \( G \) is a sequence of alternating vertices and edges in \( G \) which begins and ends at a vertex and has no edge which appears in the sequence more than once. A set of paths \( P=\{p_1, p_2, ..., p_k\} \) is a covering path set of \( G \) if for every edge \( e \) in \( G \), there exists a \( p_i \in P \), such that \( e \) is in \( p_i \), and for every pair \( p_i, p_j \in P \) (\( i \neq j \)), \( p_i \) and \( p_j \) do not share any common edge (i.e. \( p_i \) and \( p_j \) are edge disjoint). We define any permutation of \( p_1, p_2, ..., p_k \) as a covering path sequence of \( G \). A sequence of edges is called an Euler path if and only if it traverses every edge exactly once in a given multigraph. If \( |P|=1 \), then the covering path is an Euler path and the circuit can be constructed with no diffusion path [2]. If an Euler path starts and ends at the same vertex, then it is also an Euler circuit. An Euler circuit exists in a nonempty graph if and only if it has no vertices of odd degree. The sum of the degree of the vertices in a graph \( G \) is equal to twice the number of edges. It follows from this that the number of vertices of odd degree is even. If \( k \) is the number of odd-degree vertices in \( G \), then we need \( k/2 \) (1 if \( k=0 \)) paths to cover all edges in \( e \) such that each edge occurs once in a path. Such a set of paths can be found in linear time by a simple algorithm. If \( k \leq 2 \) then the simple path in the set is an Euler path (note that \( k \) is always even). If \( k>2 \) i.e. if the number of odd-degree vertices in \( G \) is more than 2, \( k/2-1 \) pseudo-edges are added to find the paths to cover the graph. A pseudo-edge acts as any true\(^1\) edge while traversing the graph, but unlike the true edge it represents a diffusion gap (see Section 4.4.) in the layout of a circuit. If \( x \) is the number of pseudo edges in \( G \), the number of paths to cover \( G \) will be \( x+1 \). Figure 4.1 shows a simple graph with 4 nodes and 6 edges. The number of odd vertices (if the number of edges connected to a node is odd) is two and they are 2 and 1. So an Euler path can be found starting from node 1 and finishing in node 2 or vice versa. For example, \( (2 \ a \ 1 \ c \ 2 \ d \ 3 \ e \ 4 \ f \ 3 \ b \ 1) \) is an Euler path because if we follow this path we can traverse the entire graph without traversing any edges more than once. The optimization of arrays of functional cells uses the concept of an Euler path to minimize

---

1. The edge which is actually present in the graph
layout area [3][19][24]. A different approach to find all the possible paths to cover an arbitrary graph is presented in Section 4.5.

**Figure 4.1 A simple graph**

![A simple graph]

### 4.3 The Layout Problems

In this section we introduce the layout problems for one-dimensional CMOS functional cells.

#### 4.3.1 Width Minimization

The primary goal in the functional cell design is to minimize the cell width $W$, which is given by the following formula,

$$W = T + G + 1$$

where $T$ is the number of transistors, $G$ is the number of diffusion gaps. $W$ is measured in units of the basic grid size, which is the minimum horizontal spacing between adjacent polysilicon gate columns, as illustrated in Figure 4.2(b). Figure 4.2(a) shows the
corresponding graph. A diffusion gap is needed to isolate transistor terminals that are physically adjacent in the cell but not connected in the transistor circuit. In Figure 4.2(b) two transistors f and e are separated by a diffusion gap because they are not connected in the circuit. Diffusion abutment, which merges two neighboring diffusion regions, can be used if the physically adjacent terminals of the transistors in the cell are connected together in the corresponding circuit. In Figure 4.2(b) transistor b, a, c, d, f are connected by diffusion abutment. Therefore, in the interests of reducing cell width, the preferred method of connecting physically adjacent transistors is diffusion abutment [19].

Figure 4.2 A graph and the corresponding layout

(a) A graph
(b) Corresponding layout

4.3.2 Height minimization

The second design goal is to minimize the cell height, which is determined by the number of horizontal routing rows needed for interconnecting transistor terminals that are not physically adjacent in the cell; such interconnection is done with metal wires as shown in Figure 4.2(b). We call these routing rows metal tracks. The transistors c and e are connected in the circuit but are not adjacent to each other in the cell; therefore, their terminals are connected by a horizontal metal wire. In Figure 4.2(b), the number of metal tracks is 2, which is the cell height. We have developed a track assignment algorithm
which gives us the maximum number of metals in the minimum number of tracks; this reduces the height of the layout (see Section 4.6.)

Minimizing cell width and height is typically called area minimization, although there may be cases where a minimum area cell has neither minimum width nor height.

4.4 The Layout Style and Procedure

The layout style we are proposing here is for arbitrary circuits. The only constraint is that each transistor gate should be driven by a primary input, i.e. the gate of a transistor cannot be connected to the source or the drain of another transistor. For dynamic circuits, since we deal with n-transistors only (except for the pull-up circuitry) this approach is very efficient for producing the layout. For static circuits another condition has to be fulfilled. The sequence of the gates of pfets in p-block and nfets in an n-block has to be the same. This does not necessarily mean that the number of pfets and nfets have to be the same. For example, let us consider the case where there are five transistors in an n-block with the sequence of a, b, c, d, e and there are three transistors a, b, c in the p-block. In this case the sequence of the transistors in the p-block should be a, b, c.

To generate the layout we need the Euler path with the information of the tracks of metals. The gates are aligned vertically and the diffusion is run horizontally. If there is a pseudo-edge in the Euler path, then there will be a diffusion gap. Figure 4.2(a) shows a symbolic layout for the graph in Figure 4.2(b). In this particular case there is one diffusion gap between node 5 and 4 since one pseudo edge (not shown in the figure) was added while traversing the graph. If a node in an Euler path exists more than once, a metal with contact is placed to connect the diffusion between the gates. All the vertical connections are made by metal1 and the horizontal connections are made by metal2. In the first track both horizontal and vertical connections are made by metal1 since there is no chance of unwanted short circuits. If two vertical metal wires do not conflict with each other they can be placed in the same track. We get this information from the metal track list which is obtained along with the Euler path list.
The steps for the proposed layout procedure are as follows:

1. Obtain the input from the (SPICE) netlist.
2. Add pseudo-edges where necessary.
3. Find the path (or paths) that covers the graph (Euler path).
4. Find the connections which can be at the same track level (track list).
5. Feed the Euler path and track list to the layout generating program.

In loading the netlist from a SPICE deck, there are four terminals for each transistor. Discarding the substrate information, the net numbers for the drain, gate and the source are provided. After obtaining the original SPICE netlist, a modified netlist is generated where all the transistors, starting from the same node, are in one list. The final list, which is to be used to find the Euler path, is a list of lists where the elements of each sublist start with the same nodal information. For example, if 1 is the source of a transistor 2 is the drain and G1 is the gate, the element of a sublist will be (1 G1 2). Also there will be another element in a different sublist which will have the same information but in reverse order (2 G1 1). Let us consider a SPICE netlist of 12 nfets:

m1 9 4 7
m2 5 3 9
m3 7 8 5
m4 8 4 5
m5 5 3 8
m6 5 7 6
m7 1 1 4 0
m8 8 3 1 1
m9 7 4 1 0
m10 1 0 3 7
m11 0 8 1 0
m12 6 7 0
The substrate information is discarded with the assumption that it is always connected to ground for nfets and to the power rail for pfets. The source and the drain of a transistor are equivalent to the two nodes of the edge which is equivalent to the gate. Now our first goal is to modify the SPICE netlist so that we can represent the modified netlist in terms of a simple graph. Starting from the very first transistor, m1, the drains and the sources in all the transistors are mapped to an incremental order. For example, 9 is the first number for the first transistor, so 9 is replace by 1. All the 9s in the netlist will be replaced by 1s. Same way 7 is replaced by 2, 5 is replaced by 3 and so on. The gate of the transistor is represented as G[gate]. For example 4 (the gate for the first, fourth, seventh and ninth transistor) will be replaced by G4, 3 will be replaced by G3 and so on. The modified netlist will now be: (((1 G4 2) (1 G3 3)) ((2 G4 8) (2 G8 3) (2 G3 8) (2 G4 1)) ((3 G7 5) (3 G3 4) (3 G3 1) (3 G4 4) (3 G8 2)) ((4 G3 3) (4 G4 3) (4 G3 6)) ((5 G7 7) (5 G7 3)) ((6 G4 7) (6 G3 4)) ((7 G8 8) (7 G7 5) (7 G4 6)) ((8 G3 2) (8 G8 7) (8 G4 2))). The total number of elements in the modified netlist is twice that of the original netlist (for each transistor there are both the actual and reverse order of the nodes). The netlist can be represented as a graph where the elements of the first sublist start with node 1, the second sublist with node 2 and so on. The gates, which are denoted as Gs, are the edges of the graph. Figure 4.3(a) shows the graph.

Figure 4.3 The graph without and with the pseudo-edge

(a) without a pseudo-edge

(b) with a pseudo-edge
As discussed in the background theory of Section 4.2., if there exist more than two odd-degree vertices, pseudo-edges are added within the graph. In the above netlist there are four odd-degree vertices (3 4 7 and 8). Pseudo-edges are added between the nodes which have greater number of edges; this way the total number of paths (not Euler paths) in the graph will be reduced and the computational time to find the Euler paths will also be less. So, one pseudo-edge (X) is added between the node 3 and node 8. The netlist becomes

(((1 G4 2) (1 G3 3)) ((2 G4 8) (2 G8 3) (2 G3 8) (2 G4 1)) ((3 G7 5) (3 G3 4) (3 G3 1) (3 G4 4) (3 G8 2) (3 X 8)) ((4 G3 3) (4 G4 3) (4 G3 6)) ((5 G7 7) (5 G7 3)) ((6 G4 7) (6 G3 4)) ((7 G8 8) (7 G7 5) (7 G4 6)) ((8 X 3) (8 G3 2) (8 G8 7) (8 G4 2))). Figure 4.3(b) shows the graph with the pseudo-edge. Now we are ready to obtain the Euler path(s) from the above netlist. The path(s) will start from node 4 and will finish at node 7 and vice versa.

Starting from one element in the sublist, which has the information of node 4, we trace the subsequent node to find an Euler path. If the node starts with the element 4 G3 3 (the first element in the fourth sublist), the next element will be the first one in the sublist of node 3, which is 3 G7 5. Then we look for the next element in node 5 as the last number of the previous element is 5. This procedure continues until a complete path is obtained. If it covers all the edges once and only once then the path is an Euler path. For example, (4 G4 3 G3 1 G4 2 G3 8 X 3 G8 2 G4 8 G8 7 G7 5 G7 3 G3 4 G3 6 G4 7) is one Euler path\(^1\). As the number of elements in most of the sublists is more than one, the number of sets of paths which cover the graph (circuit) could be much higher than one. In other words, the number of Euler paths increases with the increase of the number of edges. The next step is to choose the path which we are interested in. For this, certain criteria should be met. In the Euler path node 2 and node 7 are isolated from each other i.e if lines are drawn between the same nodes existing in the path, only the lines for node 2 and for node 7 will not be touching each other. So metal wires for 2 and 7 will be in the same track in the layout. We will discuss the procedure to find the tracks in Section 4.6. Figure 4.4 shows the corresponding layout.

---

1. In fact it is an addition of two paths connected with a pseudo edge
4.5 **The Algorithm to Find the Euler Path**

An exact algorithm to find the Euler path for any arbitrary graph is presented in this section which is implemented in SKILL language in Cadence design environment.

Step 1: Feed a portion of the SPICE netlist. Change the nodes in sequential order. Keep all the elements starting with the same node in one sublist. Put the sublists in sequential order to form a list. This is called the `input_list`.

Step 2: From the `input_list` find the sublist where the number of elements are odd. The total number of odd degree vertices must be even. If there are no odd-degree vertices or the number of odd degree vertices is two, keep the `input_list` as is. If the number of odd degree vertices is $n$ ($n > 2$) add $n-1$ number of pseudo-edges between the nodes. This list is called `input_pseudo_list`.

Step 3: Take the `input_pseudo_list` and change the position of the elements in each sublist in anti clock wise direction so that each element in that sublist occupies different position each time and the sublist is divided into $N$ number of sublists, where $N$ is the number of elements in the original sublist. Follow the procedure for all the sublists in the original list and cascade them together sequentially to form a new list. This is called `mult_comb_list`. 
Step 4: Modify the `mult_comb_list` to obtain another list where the sublists are formed by all the possible combinations of different but sequential elements. This list is the `working_list`. This is a list of lists where each sublist is comprised of lists which are combination of elements starting with the same node.

Step 5: Take the sublists from the `working_list` one at a time and, as explained in the previous section, find the path; this may or may not cover the entire graph. As the sublists are used one at a time, if same path(s) are obtained more than once discard paths which are redundant. Also discard any path which does not cover the entire graph.

4.6 The Track Assignment Problem

4.6.1 The algorithm

```plaintext
car(list) --> first element of 'list'.
cdr(list) --> 'list' except the first element
cons(a list) --> add element 'a' to 'list'.
length(list) --> total number of element in 'list'
== --> equal to
!= --> not equal to
&& --> logical and
|| --> logical or
```

Step1: Let LIST 1=the Euler path (from the previous section).

if(car(LIST 1) == [gate] of the transistor, then
   'ignore'
else
   LIST 2=cons(car(LIST 1) LIST 2); end if
Step 2: LIST 3 = (elements of LIST 2 without repeating)
Step 3: LIST 4 = (LIST A1) (LIST A2) ... (LIST AN)
                 where; 1st and last element of LIST An == nth element of LIST 3
                 pth element of LIST An = pth element after the first nth element in LIST 2
                 n = 1, 2, 3...N; p = 2, 3,... (length(LIST2)-1)
Step 4: if (1st element of LIST An == member of LIST Ai, then
                 LIST 5 = cons((1st element of LIST An & & 1st element LIST Ai) LIST 5)
                 ; end if; i! = n; n = 1, 2, 3...N; i = 1, 2,3...N
Step 5: LIST 6 = flatten(LIST 5); remove the hierarchy
Step 6: if nth element of LIST 6 & & ith element of LIST 6 == member (kth element
                 of LIST5), then
                 LIST 7 = cons((nth element of LIST 6 & & ith element of LIST 6) LIST 7);
                 i != n; n = 1, 2,...N; i = 1, 2,...N; k = 1, 2,3...

4.6.2 An example

The steps mentioned above are to be followed to obtain the track list for a certain Euler
path (or a number of paths connected with pseudo-edge(s)). We now apply the procedures
to our example to find the metal track list of the Euler path (1 G1 2 G2 3 G3 1 G4 4 G5 6
G6 2 G7 3 G8 5 G9 6 G10 7 G11 5 G12 9).

Step 1: Get the path (1 G1 2 G2 3 G3 1 G4 4 G5 6 G6 2 G7 3 G8 5 G9 6 G10 7 G11 5 G12
9). Remove the gates (edges) from the path. So the modified path becomes, (1 2 3
1 4 6 2 3 5 6 7 5 9).

Step 2: Find the number of distinct nodes in the path. In the example it is 5 (1, 2, 3, 6, and
5). We refer to this as track_info.

Step 3: Make a list of lists where each sublist starts from one distinct element (node) and
finishes with the same element. In our example, the list contains 5 sublists, each of
which starts and ends with the same element (node). So the list is, ([1 2 3 1]
4 6 2) (3 1 4 6 2 3) (6 2 3 5 6) (5 6 7 5)). This list is called the node_track_list.
Figure 4.5 (a), (b), (c), (d) and (e) show the individual nodal information for nodes 1, 2, 3, 5, and 6. Since we are not interested in any node which has just one element in the list, shown in step 2, nodal information for nodes 4 and 7 will be redundant.

Figure 4.5 Node conflicts

Step 4: If the first member of a sublist of node_track_list is not a member of another sublist then the first number of the two sublists forms another list. For example the first number of the first sublist of node_track_list is not a member of the 4th and 5th sublist, so, two lists are obtained, each of which always contains two members. These are (1 6) and (1 5). In the same way we apply this method to all the sublists, and two more lists are obtained which are (2 5) and (3 5). These lists are called no_conflict_track_lists. It is not necessary to search for the reverse of the no_conflict_track_list, i.e. if we obtain (1 6), this will be the same as (6 1), so, (6 1) will be redundant this case. Now, all the no_conflict_track_lists are combined together, and the new no_conflict_track_list becomes ((1 6) (1 5) (2 5) (3 5)). We now take the no_conflict_track_list and flatten it (hierarchically), making sure that no element in the list is repeated. The new list is (1 6 5 3 2). We refer to this list as the flattened_no_conflict_track_list.
Step 5: Take each element of flattened_no_conflict_track_list. Example 1 is the first element of the list. Then take the second element of the list which is 6. 1 and 6 will form a list of two elements. Now check if (1 6) is a member of no_conflict_track_list. If it is, compare 5 with the element of the list and see if both (1 5) and (6 5) are members of no_conflict_track_list. If they are, 5 is added to the list. If not, discard 5 and compare with 3. This process is repeated until the last element has been examined.

Step 6: Take the difference of the flattened_no_conflict_track_list and the list that was obtained in step 6. Then repeat the step 6. This process proceeds until flattened_no_conflict_track_list is null. Now we have the tracks we are interested in. They are (1 6) and (5 2). So, the final_track_list will be ((1 6) (5 2)). We are not interested in the members of the final_track_list but in the maximum number of elements in the list. Since the members of the flattened_no_conflict_track_list (1, 6, 5, 3 and 2) exist more than once in the original Euler path, and 1, 6, 5 and 2 exist in two different sublists in final_track_list, the number of metal tracks in the layout will be 3. The first two tracks come from the two sublists of the final_track_list and the third track comes from the remaining element(s) from the flattened_no_conflict_track_list.

If the number of metal tracks in the two different paths, which cover the entire graph (with or without the pseudo-edges), is identical, then the path or the set of paths, where the total metal length is smaller than that of the others, should be chosen. The total length of the metal paths is the summation of the individual metal lengths; that is, the summation of the distances between the same type of nodes. The metal length for node 1 is 3 units since it crosses three gates, G1, G2 and G3; where each unit is the spacing between two adjacent polysilicon wires. Similarly, the metal length for node 2 is 4, 5 for node 3, 3 for node 5, and 5 for node 6. So the total metal length for the path (1 G1 2 G2 3 G3 1 G4 4 G5 6 G6 2 G7 3 G8 5 G9 6 G10 7 G11 5 G12 9) is 3+4+5+3+5 =20 units. Figure 4.6 shows the layout obtained from the Euler path, the metal tracks and the metal lengths for different nodes.
Figure 4.6 A layout with the length of the metals

Track 3
Track 2
Track 1

a→ metal for node 1 (metal length=3)
b→ metal for node 2 (metal length=4)
c→ metal for node 3 (metal length=5)
d→ metal for node 5 (metal length=3)
e→ metal for node 6 (metal length=5)

metal distance

4.7 Comparison with the Switching Tree

In this section we compared the layouts obtained using the Euler path approach with those using the switching tree approach.

Table 4.1. Comparison of Euler Path and Switching Tree approach

<table>
<thead>
<tr>
<th></th>
<th>Area of Sum bit of a full Adder (10 Nfets)</th>
<th>Area of Mod 7 multiplier (87 Nfets)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Euler Path</td>
<td>$44 \times 88$</td>
<td>$131 \times 741$</td>
</tr>
<tr>
<td>Switching Tree</td>
<td>$41 \times 80$</td>
<td>$150 \times 227$</td>
</tr>
</tbody>
</table>

Table 4.1. shows the area comparison between these two methods for the sum bit of a full adder and for the core of a Mod 7 multiplier in CMOS 1.2 micron technology. The layouts

1. The technology is CMOS 1.2 micron
are shown in Appendix B. From the table, we can see that switching tree produces much more area efficient layouts than the Euler path approach. For example, the layout obtained for the Mod 7 multiplier in the switching tree has a 65% less area than that for the Euler path method. As the circuit becomes larger this relation becomes more obvious. For one-dimensional functional cells, as the number of transistors increases, the width of the cell also increases; this plays a significant role in the increase of the overall area, because, for the Euler path approach, the transistors are placed in a single row and do not share a common polysilicon gate even if they receive the same input signal; this makes the Euler Path method quite area inefficient. Also, the increase in the number of transistors will increase the total number of nodes, and this will increase the number metal tracks; which will effectively increase the height of the cell. For the switching tree style, the transistors with the same input will have a common polysilicon gate and they are aligned in one column. This will play a significant role in reducing the width of the cell, which will reduce the overall area of the design. It is probably not a fair comparison to directly compare the switching tree layout technique with other, more general, methods. The Switching Tree technique has its own style of optimizing the logic, designing a circuit and although similar to gate-matrix [10], it has a unique layout style. Although the one-dimensional functional cell does not produce the best layout, the Euler path for any arbitrary circuit will help the designer to obtain the relative placement of the transistors and the connectivity between them, which can be very useful information, even for hand crafted full-custom layout.

4.8 Summary

In this chapter we have explored the one-dimensional functional cell layout style for any arbitrary dynamic circuit. An algorithm is developed for finding Euler paths of a circuit, which can be represented as a graph. A track algorithm has been proposed and implemented to minimize the height of the cell. The Euler path and the track information for the path are used to generate the layout of the functional cell. The chapter concludes with a comparison between the one-dimensional functional cell and the switching tree,
and the observation that the switching tree style produces layouts which are more area efficient than the one-dimensional functional cell style.
Chapter 5

Simulation Results

5.1 Introduction

Many factors control the possible speed of CMOS integrated circuits such as device dimensions, logic circuit style, clocking strategy, architecture, clock distribution etc. Clocking strategy plays a very significant role in building high-speed digital systems. Clock frequency is maximized in order to achieve high system performance. For pipelined arithmetic structures, the clocking scheme has a profound impact on the design and the implementation of the system. Clock skew becomes a major design problem as the die size increases because of longer interconnection lengths. One step to attain a high clock frequency is to provide a fast storage element (latch, flip-flop, register) and a robust clocking scheme that is free from race conditions. In this chapter, the structure of two pseudo single phase latches are described and their performance is observed for switching trees of different heights.
5.2 Pipelining

Pipelining is a technique commonly used to improve the execution rate of a processing unit where the logic function is divided into smaller pieces, called stages, that operate concurrently [16]. Figure 5.1 illustrates the concept of pipelining. Boxes labeled CL represent processing networks which perform a logic function, while the circles represent the latches. The busses connecting the elements are represented by the solid lines. The important aspect of a pipeline is that all the stages are active simultaneously, working on different parts of the data stream. Data are fed into the pipeline at a rate determined by the slowest stage of the pipeline. The time required for complete data processing is not reduced by pipelining, but the rate at which data is processed improves because of the concurrency of the operations. Providing the pipeline can be kept full, then throughput rate improves linearly with number of pipeline stages.

![Figure 5.1 Pipelining](image)

5.2.1 Pipelined CMOS gates

Figure 5.2 shows a dynamic Switching Tree NFET block embedded in two different latch structures. In Figure 5.2(a), the top of the tree is connected to Vdd where the bottom part is connected to a current steering master-slave latch. The tree conducts current, or not, based on the input vector and the truth table from which the tree was constructed. The output of the tree is sensed by the current detector inside the current latch; its output is a voltage of either VDD, representing a logic high, or zero, representing a logic low. In Figure 5.2(b) the latch is voltage sensitive and there is a pull-up transistor (TP) connected to the top and a pull-down transistor (TN) connected to the bottom of the tree. These latch structures will be discussed in the next section.
Figure 5.2 Switching tree embedded in a latch

(a) N-block with current steering Master-Slave latch

(b) N-block with voltage sensitive Master-Slave latch

Figure 5.3 Timing diagram

Figure 5.2(b) shows a pipelined dynamic CMOS gate. The gate operates as follows: during precharge the clock is low as shown in the timing diagram of Figure 5.3. Since TP is on and TN (the ground switch) is off the output node O is charged to voltage Vdd. This process is referred to as precharging the dynamic gate, and during precharge all inputs to the logic block must stabilize to their input logic level. The latch at the output will ensure the input stabilization to the next cell. The voltages to which the internal nodes of the logic
block are precharged depend on the input conditions and capacitive distribution of the logic block.

When the clock makes a low to high transition, the gate enters the evaluation phase. The node O remains high if no path is formed by the input variables within the logic block. The logic high state is kept constant providing the inputs do not change. When a path is formed in the tree, node O is pulled down to VSS (zero voltage). When the clock makes a high to low transition, the output latch stores the resulting voltage on the evaluation node as a logic 1 or a logic 0, depending on how much charge has been drained away.

5.2.2 Clock skew

Clock skew can be defined as the difference in arrival times of a clock signal at different points on a chip, and it can occur between clock phases, or it may involve only one phase. This should not be confused with clock delay, since a clock signal arriving from two different paths with equal delay will have zero skew. Usually it occurs because of unequally loaded clock lines and unequal lengths of clock interconnect. In heavily pipelined systems, local connectivity is predominant, and thus the clock skew associated with the long global clock lines may be a chief performance limiting factor [5]. Clock skew worsens as integration levels increase due to longer interconnection paths, and as feature size shrinks, due to a higher RC constant per unit length of interconnection [1].

It is known that the critical problem in CMOS pipeline systems is the race problem [9]. Especially in multiple clock schemes, the generation and distribution of multiple clock phases requires additional design efforts to avoid the race problem. Inadequate design procedures can result in system clock skew, which often lead to serious race problems. The threat of race problems results in difficulties in increasing circuit speed, increasing throughput rate of pipelined systems, or decreasing logic faults.

Pseudo single phase clocking\textsuperscript{1} has been very popular, and has incited the development of many pipelined dynamic logic design. In the following section two pseudo-single phase
clocked latches will be discussed and later simulation results will be presented and discussed.

5.3 Latch Structures

5.3.1 CMOS current latch

The current latch consists of a current detector, which is configured as an n-FET latch, and a conventional TSPC p-FET latch [27]. Figure 5.1 shows the complete structure of the current steering latch. This latch requires a pseudo single phase clock and this is generated under race-free conditions by a local inverter. The M1, M2, M3, M4, M5 and M6 transistors form the current detector. The bottom of the tree is connected to the drain of M1 and M3; current conducted through the tree will be steered through M1 during pre-charge and through M3 during evaluation. M4 and M5 form a current mirror and M2 will ensure that the current mirror is off during pre-charge. During evaluate, any current conducted by the tree will be mirrored into current/voltage converter, M5, M6 and thus pull-down will be achieved if current flows in the tree. The current detector, and M7, M8, M9 transistor form the n-latch. The last stage is the p-latch, which is constructed by M10, M11, M12, M13, M14 and M15.

The current steering latch senses static current flowing in the tree and latches its output accordingly, which is inherently a faster mode of operation for larger trees [5]. This can be appreciated by considering the following case. For a low value to appear on the evaluation node of a tree embedded in a pseudo single phase voltage sensing latch, it must be discharged through the tree. Thus, a delay is incurred which is dependent on the time it takes to remove sufficient charge to reduce the evaluation node voltage, and this node cannot be sampled before this delay has passed. With the new structure, however, as soon as current begins to flow, it can be detected, and the latch can react accordingly.

1. It requires a $\phi$ (clock) and a $\bar{\phi}$ (inverse clock)
5.3.2 The voltage sensing master-slave latch

Figure 5.5 shows the master-slave latch [26] connected to the N-FET logic block. In this circuit M1 is the pull-up transistor and M2 is the pull down transistor. For a switching tree, the ground switches act as the pull-down transistors. M3 works as a voltage sensor. M5, M6, M7, M8 and M9, M10, M11, M12 form a master-slave latch. When the clock signal ($\phi$) is low, the circuit is in its pre-charge phase. The output from the slave latch depends on the result held by the master latch since M10 and M11 are on. The voltage at the top of the N-FET block will be pulled up to VDD since M1 is turned on while M2 is off. This will turn M3 off and since M4 is on, the voltage at the drain of M4 will become logic zero.

When the clock signal is low ($\bar{\phi}$ is high) the circuit is in its evaluation phase. The slave latch holds the previous result and the voltage from the output of the master latch depends on the output from the logic block. If the output from the logic block is high, M3 will stay off and therefore, the output of the master latch will be high (previous result). On the other hand, if the output from the logic block is low, the node at the top of the block will be pulled down to zero. In this case M3 will sense the change of the voltage at the same node and when the gate voltage is less than 3.0V, it turns on. Since M4 is off, it will force the output of the master latch to become low.
5.4 Simulation Results

5.4.1 CMOS 1.2 micron technology

In this section, simulation results for trees of various heights and widths are illustrated for CMOS 1.2 micron technology. The latch used for each stage of the pipelining process is voltage sensitive master-slave latch. Several test structures were obtained from the original trees and simulation results were studied for those structures. All the simulations were performed from extracted layouts.

Figure 5.6 shows the simulation results of the extracted circuit from the layout of a MOD 7 multiplier (generated by the DMS module generator). It is a six high tree and contains 87 transistors including the ground switches. All the transistors in the tree are of the same size and they are fed with either true or complement input signals through inverters, depending on their position in the tree. The structure was clocked successfully up to 142 Mhz with a rise time of 1ns on the clock waveform; TSPC latches were used. The layout of the MOD 7 multiplier is shown in Appendix B.
Two test structures are shown in Figure 5.7 and Figure 5.8. They are built using our target 1.2 micron CMOS technology. Each structure is a part of a MOD17 multiplier, a 10 high tree, and contain 1055 transistors. These structures were generated using DMS. Figure 5.9 and Figure 5.7 show the simulation results of Figure 5.10 and Figure 5.8 respectively. The frequency of the clock for both cases is 100 MHz with fall and rise time of 1ns each.
Figure 5.7 Test structure 1 (10 high)

Figure 5.8 Test structure 2 (10 high)

Figure 5.9 Simulation of test structure 1 @ 100 Mhz

f1:in6 in volts

f1:in7 in volts

f1:out in volts

f1:clk in volts

0  50n  100n  150n
Figure 5.10 Simulation of test structure 2 @ 100 Mhz

It is essential that the cells perform normal operation when they are cascaded in pipeline chains. We cascaded several similar cells to check this performance; i.e. the outputs from the previous stage provide inputs to the following stage. The clock signal and the data are passed in the same direction. To check the functionality and the timing behaviour, this process is done in such a way that we expect to see the same result (output) after every stage; i.e. the outputs expected at each stage will have the same logic value as the specified inputs to the same stage.

Figure 5.11 Test structure 3 (Indirect)
Figure 5.12 shows the simulation results of the tree of Figure 5.11, where four cells are cascaded together. At a 70 Mhz clock rate, the simulations show correct functionality. An increase of frequency to 83 Mhz resulted in a failure (Figure 5.13). So, we can conclude that trees of height 10, generated using the Indirect Mapping Style, can be expected to operate in a pipelined array up to a clock frequency of 70 Mhz.

Figure 5.12 Simulation of 4 cascaded 10 high trees (test structure 3) @ 70 Mhz

Figure 5.13 Simulation of 4 cascaded 10 high trees (test structure 3) @ 83 Mhz
We also observed correct operation at 70MHz for a cascade of three 12 high trees (Figure 5.14). Figure 5.15 shows the layout of the test cells. Although every stage in the switching tree network is pipelined the maximum operating clocking speed gets slower when they are cascaded. Dependencies between the data streams prevent the pipeline from achieving its maximum performance. To achieve a high efficiency in the pipeline, the data from the previous stage should be available before the next stage starts to activate. Due to memory latency and other dependencies, the efficiency of a typical pipeline of multiple stages is less than that for just one stage (data is processed for single stage only). Feeding the clock signal opposite to the data stream could improve the overall performance.

**Figure 5.14 Simulation of 3 cascaded test structures (4) @ 70 Mhz**

**Figure 5.15 Test structure 4 (12 high)**
5.4.2 Mitel 1.5 micron technology

In this section simulation results are shown for the Mitel 1.5 micron technology, with layouts generated by the module generators. The latches used at the output stage(s) are either voltage sensitive or current steering master-slave type. Several test structures were obtained from the original trees and simulation results were studied for those structures. All the simulations were performed from extracted layouts.

**Using voltage sensing master-slave latch**

Figure 5.16 shows the expected simulation results of a cell where 3 test structures (Figure 5.17), each 10 high, are cascaded together. The clock frequency is 100Mhz and the rise and fall time of the clock is set at 1ns.

**Figure 5.16 Simulation results of 4 cascaded 1m test structures @ 100 Mhz**

- f1:in9 in volts
- f1:out in volts
- f1:out1 in volts
- f1:out2 in volts
- f1:clk1 in volts

**Figure 5.17 Test structure 1m (10 high)**

[Diagram showing test structure with latches and inverters]
Using current steering master-slave latch

A test structure of height 12 (Figure 5.18) has been chosen to observe the performance of the current steering latch. At 125 Mhz clock frequency the cell operates normally and produces the expected output (out in Figure 5.19). But the same circuit fails as we increase the frequency from 125 MHz to 142 Mhz (out in Figure 5.20).

Figure 5.18 Test structure 3m (12 high)

Figure 5.19 Simulation results of test structure 3m @ 125 Mhz

\[
\begin{align*}
&f1: in7 \text{ bar in volts} \\
&0.25 \\
&f1: in5 \text{ in volts} \\
&0.25 \\
&f1: in9 \text{ in volts} \\
&0.25 \\
&f1: out \text{ in volts} \\
&0.45 \\
&f1: clk1 \text{ in volts} \\
&0.25
\end{align*}
\]
Table 5.1 summarizes the simulation results of switching trees of various heights for CMOS 1.2 micron technology.

<table>
<thead>
<tr>
<th>Switching tree height</th>
<th>Number of stages</th>
<th>Operating frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>142</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>70</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>70</td>
</tr>
</tbody>
</table>

5.5 Summary

In this chapter the clocking strategies from pipelined arithmetic structures were briefly discussed. Also two master-slave latches, with pseudo single phase clocking scheme were discussed and extensive simulations were done to observe their performance for different switching tree test structures. These test structures were obtained from the module generator discussed in chapter 3. They were cascaded together to verify the pipelining effect and their simulation results were observed for different clock frequencies. A switching tree of height 10 can perform normal operation at 100 MHz clock frequency. If
four of these trees are cascaded together the maximum operating clock frequency reduces to 70 Mhz.

Almost similar performance was observed for both voltage sensitive and current steering master-slave latches. But for high trees, the current steering latch tends to produce better results than its voltage sensing counterpart.
6.1 Conclusion

The objective of this work was to develop automatic layout generators for high throughput rate complex pipelined logic blocks. A parameterized module generator has been developed for the switching tree circuit design style where the trees can be of arbitrary heights. The automatic generation of the layout of a cell is provided input from a previously developed symbolic layout switching tree synthesizer [6]. Two layout styles; direct and indirect, are suggested to produce area efficient and compacted layouts. In the direct mapping style, the transistors are placed exactly as directed in the topological matrix (symbolic layout) and the routing between them is also mapped accordingly. The indirect mapping style, compact the layout in the vertical direction in order to make it more area efficient. Leaf cells are placed in the area which is saved due to this compaction. The relative placements of the transistors remain unchanged. This module generator has been provided with user friendly interface, and is capable of producing layouts for 1.2 micron Northern Telecom CMOS, 1.5 micron Mitel CMOS, and Northern Telecom 0.8 micron BiCMOS technologies. It is also capable of producing layouts for other technologies.
providing that an appropriate technology file is attached to the module generator.

A one-dimensional functional cell generator was also developed based on a graph theoretic approach. Two algorithms have been suggested and implemented for this purpose: the first one finds the Euler paths for any arbitrary graph and the other one finds the minimum number of metal tracks to minimize the height of the cell. This functional cell layout was implemented in CMOS 1.2 micron technology.

Comparisons were made between the switching tree and the one-dimensional functional cell layout styles. It was observed that for the same design, the switching tree approach produced much more compacted and area efficient layouts than its one-dimensional functional cell counterpart. As the design becomes larger this comparison becomes more pronounced.

Finally, clocking strategies for pipeline systems were briefly reviewed, and two latch structures based on pseudo and true single phase clocking schemes were discussed. Simulations were then carried out for switching trees of various heights using these latches. Several test structures were generated and were simulated for this purpose. The maximum height of the tree that was tested was 12. It functioned correctly at a frequency of 125 Mhz using a current steering latch.

### 6.2 Future Work

For both the switching tree and the one-dimensional functional cells the following are suggestions for future work to improve the layout style.
6.2.1 Switching tree

1. The inverters used in the design can be parameterized; i.e the inverter sizes should size depends on the number of transistors in each row. The higher this number, the larger the inverter should be. This will reduce the over driving tendency of large inverters for a smaller number of transistors.

2. Although the indirect mapping has a better layout style than the direct mapping, with the increase of tree height the layout area can still be inefficient. This can be avoided by breaking up the design in the middle and placing each section vertically. But in this case the overall routing structure is critical and automating intra cell routing is not a simple task.

3. The transistors can be parameterized in the design and they can be scaled depending on the height of the tree. This should have a positive effect on optimizing the operating speed of switching tree based circuits.

4. If non local merging can be implemented in the switching tree synthesis, the layout style should be changed accordingly. Any technology with more than two metal layers could be used in implementing the switching tree from symbolic description to mask layout.

5. Higher switching trees should be simulated with alternate latches to search for better pipelining performance.

6.2.2 One-dimensional functional cell

1. For one-dimensional functional cells, the transistors are placed in one row and are fed by primary inputs. If the transistors with common inputs share the same polysilicon gate, this will optimize the area and will also reduce the overall routing channel area. The transistors can also be assigned a scaling factor and their sizes varied depending on the requirement of the circuits.
2. Rather than finding all the possible paths which cover the entire graph, a heuristic can be established to indicate only the paths the user is interested in; this will reduce the execution time of the program significantly.
REFERENCES


Appendix A

How to Use the Package

A.1 Switching Tree

A.1.1 Which program should be used

Run the decision program. To run the program type include "-lcmos4sill\decision" and include "-lcmos4sill\read_it0" in the CIW. Make sure that the technology file read_it0 has all the necessary information for a specific technology. Load the original matrix in the ‘show file’ window. Necessary information will pop-up in the CIW window. If the suggestion is to use the Direct Mapping Style see section A.2 and if the you are interested in the Indirect Mapping Style check the section A.3. You can skip this section if you know exactly which program should be used.

A.1.2 Direct mapping style

CMOS 1.2 micron

1) Type include "-lcmos4sill\opus_1.2_full" in CIW.
2) type the executing command. (make)
3) Fillup the 'show file' form with the matrix file.
4) Fill up the 'open design' form with library name, design name and view name.

BiCMOS 0.8 micron

1) Type include "-lcmos4sill\opus_bicmos_full" in CIW.
follow (2), (3) and (4) from CMOS 1.2
Mitel 1.5 micron

1) Type include "-/cmos4s/ill/opus_mitel_test_full_novia" in CIW.
follow (2), (3) and (4) from CMOS 1.2.

Other Technologies

1) Type include "-/cmos4s/ill/opus_full_general_character_a".
2) Open the file "read_it0" and provide necessary information, then in the CIW window
include "-/cmos4s/ill/read_it0".
follow (2), (3) and (4) from CMOS 1.2.

A.1.3 Indirect mapping style

FOR ALL THE PROCESSES

1) Divide the matrix in two sections.
2) The first section will have first 6 (six) rows from the original matrix. Store it in a file.
   Also store rest of the rows in another file. You have two files with the information of
   upper and lower section of the design.

CMOS 1.2 micron

1) Type include "-/cmos4s/ill/opus_1.2_short" and include "-/cmos4s/ill/opus_instance_1.2_test" in the CIW.
2) Type the executing command in the CIW window (make)
3) Fillup the 'show file' form with the bigger matrix file.
4) Fill up the 'open design' form with library name, design name and view name.
5) Wait for a while and fillup again the 'show file' form with the smaller matrix file.

Bicmos 0.8 micron
1) Type `include `-lcmos4s/iil/opus_bicmos_short"` and `include `-lcmos4s/iil/opus_instance_bicmos"` in the CIW.

follow (2), (3), (4) and (5) from CMOS 1.2

**Mitel 1.5 micron**

_for voltage sensitive latch_

1) Type `include `-lcmos4s/iil/opus_mitel_short_novia"` and `include `-lcmos4s/iil/opus_instance_mitel_novia"` in the CIW.

follow (2), (3), (4) and (5) from CMOS 1.2

_for current steering latch_

1) `include `-lcmos4s/iil/opus_mitel_short_c_l"` and `include `-lcmos4s/iil/opus_instance_mitel_c_l"`

follow (2), (3), (4) and (5) from CMOS 1.2

**Other Technologies**

1) Type `include `-lcmos4s/iil/opus_short_general_character_a"` and `include `-lcmos4s/iil/opus_instance_general_character_a"` in CIW.

2) Edit the file "read_it0" and provide necessary information, then in the CIW window type `include `-lcmos4s/iil/read_it0". You have to edit the "read_it0" file only once for a specific technology and then store it as a different file name. Next time while reading this file read it as "-lcmos4s/iil/NEW_FILE_NAME".

3) Type the executing command in the CIW window. (make)

4) Fillup the 'show file' form with the bigger matrix file.

5) Fill up the 'open design' form with library name, design name and view name.

6) Wait for a while and fillup again the 'show file' form with the smaller matrix file.
A menu can be added to the CIW window which helps the user to run the program directly from the menu. All the necessary files mentioned above should be included in the .cdsinit file in the current directory of a specific technology. In addition to those files, the executing command to invoke a menu should also be included in .cdsinit file (see section C.1.4 and C.1.5). This way the menu STLAS (Switching Tree Layout Synthesis) is included in the menu bar whenever that technology is invoked. Figure A.1 shows a sample menu in the CIW in Mitel 1.5 micron technology. Under STLAS there is a README file and EXECUTE to execute both DMS and IMS directly without loading the files manually.

A.2 The Euler Path Approach

Load the following programs in the CIW in OPUS. All of them are in ~/cmos4s/il directory. Load a part of the spice netlist in read_spice (see sample in read_spice file)

1) list2.bak
2) mult_comb.use_it
   mult_comb_one_euler (for one Euler path)
3) same_track
4) read_spice

Type the executing command \( (\text{read_it}()) \)

To generate the layout, take the Euler path and the track info, and place them in proper place in \( \text{lay_gen_euler} \) file (see the sample in this file). Run it with \( \text{n_tram_main1()} \).
Appendix B

Examples and Layouts

B.1 Switching Tree

B.1.1 The leaf cells

Figure B.1 The half_p and half_n inverters (direct mapping style)

(a) half_n_inverter  (b) half_p_inverter
Figure B.2 Inverters for the indirect mapping style

(a) inverter for driving the transistors in the upper section

(b) inverter for driving the transistors in the lower section

Figure B.3 Current steering master-slave latch
B.1.2 Direct Mapping Style

**sum bit of a full adder**

Number of transistors (in the core) = 9

Tree height = 3

Number of inputs = 3

Number of outputs = 1

Area (in Mitel technology) = 143 × 147 sq. micron (including the leaf cells)

---

Figure B.5 The topological matrix of the sum bit of a full adder

(WWbb
RbGb
WbWb
GRGR
bWbb
RbGR
WWWb
bbBR)
Mod 7 multiplier

Number of transistors (in the core) = 87
Tree height = 6
Number of inputs = 6
Number of outputs = 3
Area (in Mitel technology) = 313 × 200 sq. micron (including the leaf cells)
Mod 17 multiplier

Number of transistors (in the core) = 1055
Tree height = 10
number of inputs = 10
number of outputs = 5
Area (in Mitel teschnology) = 2228 \times 263 \text{ sq. micron (including the leaf cells)}

Figure B.9 Layout of Mod 17 multiplier (direct method)

B.1.3 Indirect Mapping Style
Mod 17 multiplier

Area (in Mitel teschnology) = 2147 \times 246 \text{ sq. micron (including the leaf cells)}

Area saved over the direct mapping style = 9.86%

Figure B.10 Layout of Mod 17 multiplier (indirect method)

B.2 The Functional Cell

B.2.1 An example
(The sum bit of a full adder)
Figure B.11 The precharged CMOS implementation of the full adder (sum bit)

Spice Netlist representation of the transistor schematic:

M37 0 17 10
M38 10 6 9
M39 9 7 8
M40 8 4 16
M41 16 5 10
M42 10 1 12
M43 12 2 13
M44 13 21 9
M45 8 3 14
M46 14 20 13

From Figure B.11, we have:

node 1: ((1 X1’ 2) (1 X1 7))
node 2: ((2 X1’1) (2 X2 3) (2 X2’ 5))
node 3: ((3 X2 2) (3 X3’4))
node 4: ((4 X3’3) (4 X3 5) (4 X3’6) (4 GS 8))
node 5: ((5 X2’ 2) (5 X2 7) (5 X3 4))
node 6: ((6 X2’ 7) (6 X3’ 4))
node 7: ((7 X1 1) (7 X2 5) (7 X2’ 6))
node 8: ((8 GS 4))
The netlist:

\[( (((1 \times 1') 2) (1 \times 1) 7) ) (((2 \times 1') 1) (2 \times 2 3) (2 \times 2' 5)) (((3 \times 2) 2) (3 \times 3') 4) ((4 \times 3') 3) (4 \times 3 5) (4 \times 3' 6) (4 \times 3 8')) (((5 \times 2') 2) (5 \times 2 7) (5 \times 3 4)) (((6 \times 2') 7) (6 \times 3' 4)) (((7 \times 1 1) (7 \times 2 5) (7 \times 2' 6)) (8 \times 3 4))\]

One set of paths to cover the graph:

\((2 \times 1' 1 \times 1 7 \times 2 5) \text{ and } (8 \times 3 4 \times 3 5 \times 2' 2 \times 2 3 \times 3' 4 \times 3' 6 \times 2' 7).\)

If we combine them together the path is\((2 \times 1' 1 \times 1 7 \times 2 5 \times 2 8 \times 3 4 \times 3 5 \times 2' 2 \times 2 3 \times 3' 4 \times 3' 6 \times 2' 7).\) Where, \(\times\times\) = pseudo-edge (diffusion gap). Figure B.12 shows the layout.

**Figure B.12 Layout of the sum bit of a full adder (Euler path approach)**

*Diagram showing layout of the sum bit of a full adder.*

---

**Mod 7 multiplier (the core)**

**Figure B.13 The layout of Mod 7 multiplier (the core section)**

*Diagram showing layout of Mod 7 multiplier (the core section).*
Appendix C

The SKILL Codes

C.1 Switching Tree

C.1.1 Direct mapping style

/* MAIN PROGRAM */

procedure( make() /* main prog */

read_it() /* reads the technology file */

/* THE FOLLOWING LINES ARE USED TO LOAD THE LEAF CELLS */
hiOpenFileForm->file->value= ""
hiViewfile()
if( hiOpenFileForm->file->value != "" then 

deOpen()
cv = dbOpenCellView(deOpenForm->deLibName->value deOpenForm->deCellName->value
deOpenForm->deViewName->value nil "w")
cv_half_p= dbOpenCellView(c.library c-half_p_invert "layout")
cv_rev_half_p=dbOpenCellView(c.library c-half_p_reverse "layout")
cv_half_n=dbOpenCellView(c.library c-half_n_invert "layout")
cv_rev_half_n=dbOpenCellView(c.library c-half_n_reverse "layout")
cv_latch_first=dbOpenCellView(c.library c-latch_direct_first "layout")
cv_latch=dbOpenCellView(c.library c-latch_direct "layout")
inport=infile(hiOpenFileForm->file->value)
inport=infile(hiOpenFileForm->file->value)
hiChangeBannerLabel(hiGetCurrentWindow() "Switching Tree (WRITE THE
TECH."") 0)

/* THIS SECTION IS READING THE FILE */

starting_list='()
n='()
Total_lines=0
b=getc(inport)
while( b != nil
  :if( b== 'n
    if( b == 'E
      Total_lines=Total_lines+1
  )
  b=getc(inport)
)
:println(Total_lines)
close(inport)

for( j l Total_lines
  b=getc(inport)
  while( b != 'n
    :while( b != 'E
      if( b != ' ' then
        starting_list=cons(b starting_list)
      )
      b=getc(inport)
    )
  n=cons(reverse(starting_list) n)
  starting_list='()

  n=reverse(n)
  println(length(n))
close(inport)

  /* END OF READING FILE */

println(n)
l1=round(length(n)-2)/2
p2=sortl(n)
printf("length = %2d\n\n" length(p2))
p3=reverse(p2)
p3=add_short_circuit1(p3)
p4=remove_met(p3)
p6=remove_tran(reverse(p3) p4)
l=length(p4)
m=length(car(p4))
polysilicon1(l m
latch_clk=out3(car(reverse(p3)) cadr(reverse(p4)) length(p4)
car(reverse(p4)))
metal2_0(1 m length(p4) latch_clk)
dev2(m)
con(m)
met(m)
p7=tran_match_2(p4)
p8=met_matching(p6)
column=length(car(p3))
ii=1
y_pos=a.ROW_SPA+a.REF_PT_HALF_P_Y
x_pos=a.REF_PT_HALF_P_X
for( j 1 round(length(p4)/2)
if( oddp(ii) then
r=dbCreateInst(cv cv_half_p nil list(x_pos y_pos) "R0" 1)
else
r=dbCreateInst(cv cv_rev_half_p nil list(x_pos y_pos) "R0" 1)
)
ii=ii+1
y_pos=y_pos+2*a.ROW_SPA
)
jj=1
y_pos_n=a.ROW_SPA+a.REF_PT_HALF_N_Y
x_pos_n=a.COL_SPA*column+a.REF_PT_HALF_N_Y
for( j 1 round(length(p4)/2)
if( oddp(jj) then
r=dbCreateInst(cv cv_half_n nil list(x_pos_n y_pos_n) "R0" 1)
else
r=dbCreateInst(cv cv_rev_half_n nil list(x_pos_n y_pos_n) "R0" 1)
)
jj=jj+1
y_pos_n=y_pos_n+2*a.ROW_SPA
)
dbSave(cv)
)

/* ADDING THE SHORT CIRCUITS BETWEEN THE POLYSILICON AND THE METAL2 */

procedure( add_short_circuit1(a)
prog( ()
pdev_temp_list=' ()
while( a != ' ()
temp_list=' ()
i=1
CAR=car(a)
LENGTH_CAR=length(CAR)
ADD=round(length(CAR)/14)
while( CAR != ' ()
;if( mod(i 7) == 0 then
if( mod(i 14) == 1 then
temp_list=cons(car(CAR) temp_list)
temp_list=cons('X temp_list)
else
temp_list=cons(car(CAR) temp_list)
)
i=i+1
ii=i-1
CAR=cdr(CAR)
)
if( mod(ii 14) == 0 || mod(ii 14) > 8 then
temp_list=cons('X cdr(temp_list))
)
temp_list-reverse(temp_list)
pdev_temp_list-cons(temp_list pdev_temp_list)
    a=cdr(a)
)
pdev_temp_list-reverse(pdev_temp_list)
return(pdev_temp_list)


/* THE OUTPUT(S) FROM THE DESIGN */

procedure( out3(x y_list z yl_list)
    ( prog ()
        latch_x-=a.\text{HALF\_N\_WIDTH}/2
        number =0
        incre=0
        for(i 1 length(x)
            if( (nthelem(i x) == 'W && nthelem(i+1 x) == 'b) || (nthelem(i x) == 'b
                && nthelem(i-1 x) != 'W && nthelem(i-1 x) != 'X && (nthelem(i y_list)
                == 'G || nthelem(i y_list) == 'R)) || (nthelem(i x) == 'b && nthelem(i-1
                x) != 'W && nthelem(i-1 x) != 'X && (nthelem(i y_list) == 'G ||
                nthelem(i y_list) == 'R)) || (nthelem(i x) == 'W && nthelem(i+1 x) == 'X
                && nthelem((i+2) x) == 'b) then

            y=(z-1)*a.\text{ROW\_SPA}+a.\text{UP\_CON\_EDGE}-(a.\text{MIN\_MET1}-a.\text{MIN\_CON})/2+a.\text{MIN\_VIA\_MET2}
            x1=i*a.\text{COL\_SPA}-1.5*a.\text{MIN\_MET2}
            dbCreateRect(cv b.via list(x1:y x1+a.\text{MIN\_VIA}:y+a.\text{MIN\_VIA}))
            y2=(z-1)*a.\text{ROW\_SPA}+a.\text{UP\_CON\_EDGE}-(a.\text{MIN\_MET1}-a.\text{MIN\_CON})/2
            x2=i*a.\text{COL\_SPA}-a.\text{MIN\_VIA\_MET2}-1.5*a.\text{MIN\_MET2}
            dbCreateRect(cv b.met1 list(x2:y2)
            x2+a.\text{MIN\_VIA}+2*a.\text{MIN\_VIA\_MET2}:y2+a.\text{MIN\_VIA}+2*a.\text{MIN\_VIA\_MET2})
            dbCreateRect(cv b.met2 list(x2:y2)
            x2+a.\text{MIN\_VIA}+2*a.\text{MIN\_VIA\_MET2}:y2+a.\text{MIN\_VIA}+2*a.\text{MIN\_VIA\_MET2})

            if( number == 0 then
                r=dbCreateInst(cv cv latch_first nil list(latch_x
                y2+a.\text{ADJ\_LAT\_Y\_POS\_DIRECT} "R0" 1)
            else
                r=dbCreateInst(cv cv latch nil list(latch_x y2+a.\text{ADJ\_LAT\_Y\_POS\_DIRECT})
                "R0" 1)

        )

    dbCreateRect(cv b.met2 list(x2:y2 x2+a.\text{MIN\_MET2}:y2+2*a.\text{MIN\_MET2}+incre))
    dbCreateRect(cv b.met2 list(x2:y2+a.\text{MIN\_MET2}+incre
    latch_x+a.\text{DIST\_IN\_LAT}:y2+2*a.\text{MIN\_MET2}+incre))
    dbCreateRect(cv b.met2 list(latch_x+a.\text{DIST\_IN\_LAT}:y2+a.\text{MIN\_MET2}+incre
    latch_x+a.\text{DIST\_IN\_LAT}+a.\text{MIN\_MET2}:y2+a.\text{ADJ\_LAT\_Y\_POS\_DIRECT}+a.\text{ADJ\_INP\_DISTR\_Y\_DIRECT\_LAT})

Appendix C  The SKILL Codes 98
if( number < 1 then
  incr=0
else
  incr=incr+a.MIN_MET2+a.MIN_MET2_MET2
)
number=number+1
latch_x=latch_x+a.LAT_WIDTH_DIRECT
)
)
dbCreateRect(cv b.met1 list(latch_x:y2+a.ADJ_LAT_Y_POS_DIRECT
length(y_list)*a.COL_SPA+a.HALF_N_WIDTH:y2+a.ADJ_LAT_Y_POS_DIRECT+3*a.M
IN_MET1))
latch_x=latch_x-a.LAT_WIDTH_DIRECT
return(latch_x)
)

/*/ REMOVES METAL INFO FROM THE MAIN LIST */

procedure( remove_met(x)
n='(')
while( x != ')' )
if( member('R car(x)) || member('G car(x)) then
n=cons(car(x) n)
)
x=cdr(x)
)
reverse(n)
)

/*/ REMOVES TRANSISTOR INFO FROM THE MAIN LIST */

procedure( remove_tran(a b)
prog( ()
p=setof(e a (! member(e b)))
return(cdr(reverse(p)))
)
)

/*/ SPLITTING THE T SET INTO TR AND TG ROWS */

procedure( sort1(x)
prog(()
in='(')
y=1 /* G */
y1=2 /* R */
y2=4 /* R */
y3=5 /* G */
while( x != ')' )
if( member('R car(x)) || member('G car(x)) then

Appendix C The SKILL Codes 99
xl=copy(car(x))
in=cons(car(x) cons(xl in))
else
in=cons(car(x) in)
) /* end if */
x=cdr(x)) /* end while */
xl=in

while( (y < length(xl))
p=subst('b 'R nthelem(y xl))
x=insertR(p nthelem(y xl) xl)
xl=rember(nthelem(y x) x)
/*print(xl)* /
y=y+6 )

while( (y1 < length(xl))
p=subst('b 'G nthelem(y1 xl))
x=insertR(p nthelem(y1 xl) xl)
xl=rember(nthelem(y1 x) x)
/*print(xl)* /
y1=y1+6 )

while( (y2 < length(xl))
p=subst('b 'G nthelem(y2 xl))
x=insertR(p nthelem(y2 xl) xl)
xl=rember(nthelem(y2 x) x)
/*print(xl)* /
y2=y2+6 )

while( (y3 < length(xl))
p=subst('b 'R nthelem(y3 xl))
x=insertR(p nthelem(y3 xl) xl)
xl=rember(nthelem(y3 x) x)
/*print(xl)* /
y3=y3+6 )

return(reverse(xl))
)
) /* END PROG */
)

/* ADDS A NEW ELEMENT AT THE RIGHT OF AN EXISTING ELEMENT */

procedure(insertR(new old lat) /* insert an element at the right */
(cond
((equal lat '()) t)
((equal car(lat) old)
cons(old cons(new (cdr lat)))
)
(t cons(car(lat) insertR(new old (cdr lat)))))
)
/* REMOVES AN ELEMENT FROM A LIST */

procedure(remer(a lat) /* remove the element */
(cond
  ((equal lat '()) '())
  ((equal car(lat) a) cdr(lat))
  (t cons(car(lat) remer(a (cdr lat))))))

/* CHECKS THE MAIN LIST AND PLACES THE TRANSISTORS, VERTICAL METALS AND THE SHORT-CIRCUITS */

procedure( tran_match_2(x) )
j=1
k=1
s=0
while( x != '(')
a=car(x)
p=length(a)
for( i 1 p
  b=nthelem(i a)
  if( (j=1 || mod(j 2)==1) then
    if(b == 'G then
      mkmasteru2((k-1)*a.COL_SPA (j-1)*a.ROW_SPA )
      s=s+1
    )
    if(b == 'R then
      mkmasteru2((k-1)*a.COL_SPA (j-1)*a.ROW_SPA )
      s=s+1
    )
    if(b == 'W mkmet((k-1)*a.COL_SPA (j-1)*a.ROW_SPA ))
    if(b == 'X short_circuitl((k-1)*a.COL_SPA+(a.RIGHT_DEV_X-
      a.COL_SPA+0.8*a.MIN_MET1) (j-1)*a.ROW_SPA-(a.MIN_CON+2*a.MIN_CON_POL-
      a.MIN_POL)/2 ))
   )
  if( (j=2 || mod(j 2)==0) then
    if(b == 'G then
      mkmasterdl((k-1)*a.COL_SPA (j-1)*a.ROW_SPA )
      s=s+1
    )
    if(b == 'R then
      mkmasterdl((k-1)*a.COL_SPA (j-1)*a.ROW_SPA )
      s=s+1
    )
    if(b == 'W mkmet((k-1)*a.COL_SPA (j-1)*a.ROW_SPA ))

Appendix C The SKILL Codes 101
if(b == 'X short_circui((k-1)*a.COLSPA+(a.RIGHT_DEV_X-
a.COLSPA+0.8*a.MIN_MET1) (j-1)*a.ROW_SPA-(a.MIN_CON+2*a.MIN_CON_POL-
a.MIN_POL)/2 ))}

k=k+1
j=j+1
x=cdr(x)
k=1
)
println(s) /* end while */
)

/* CHECKS THE MAIN LIST AND PLACES THE HORIZONTAL METALS */

procedure( met_matching(y)
j=1
k=1
while( y != '()' a=car(y) p=length(a) for( i 1 p b=nthelem(i a) bl=nthelem(i+1 a) if(j then

if(b == 'W & & bl != 'X then
met_hor((k-1)*a.COL_SPA (j-1)*a.ROW_SPA*2) )

if(b == 'W & & bl == 'X then
met_hor_extension((k-1)*a.COL_SPA (j-1)*a.ROW_SPA*2 )

x=k+1
j=j+1
y=cdr(y)
k=1
) /* end while */
)

/* TRANSISTOR IN THE UPPER ROW */

procedure( mkmasteru2(x y)
dbCreateRect(cv b.poly_list(x:y x+a.POL_LENGTH:y+a.MIN_POL))
dbCreateRect(cv b.dev_list(x+a.LEFT_DEV_X:y-a.DOWN_DEV_Y x+a.RIGHT_DEV_X:y-a.UP_DEV_Y))
dbCreateRect(cv b.met1_list(x+a.LEFT_DEV_X:y-a.DOWN_DEV_Y x+a.RIGHT_DEV_X:y+a.UP_DEV_Y))
dbCreateRect(cv b.met1_list(x+a.LEFT_DEV_X:y-a.DOWN_DEV_Y x+a.RIGHT_DEV_X:y+a.UP_DEV_Y))
dbCreateRect(cv b.met1_list(x+a.LEFT_DEV_X:y-a.DOWN_DEV_Y x+a.RIGHT_DEV_X:y+a.UP_DEV_Y))
dbCreateRect(cv b.met1_list(x+a.LEFT_DEV_X:y-a.DOWN_DEV_Y x+a.RIGHT_DEV_X:y+a.UP_DEV_Y))
dbCreateRect(cv b.met1_list(x+a.LEFT_DEV_X:y-a.DOWN_DEV_Y x+a.RIGHT_DEV_X:y+a.UP_DEV_Y))
dbCreateRect(cv b.met1_list(x+a.LEFT_DEV_X:y-a.DOWN_DEV_Y x+a.RIGHT_DEV_X:y+a.UP_DEV_Y))
})

Appendix C The SKILL Codes 102
/* TRANSISTOR IN THE LOWER ROW */

procedure( mkmasterd1(x y)
dbCreateRect(cv b.poly list(x:y x+a.POL_LENGTH:y+a.MIN_POL))
dbCreateRect(cv b.dev list(x+a.LEFT_DEV_X:y-a.DOWN_DEV_Y x+a.RIGHT_DEV_X:y+a.UP_DEV_Y))
dbCreateRect(cv b.met1 list(x+a.LEFT_DEV_X:y-a.DOWN_DEV_Y x+a.RIGHT_DEV_X:y-a.DOWN_MET_Y))
dbCreateRect(cv b.met1 list(x+a.LEFT_DEV_X:y+a.UP_MET_Y x+a.RIGHT_DEV_X:y+a.UP_DEV_Y))
dbCreateRect(cv b.met1 list(x+a.LEFT_MET_X:y+a.DOWN_MET_TRAN_DOWN x+a.LEFT_MET_X:y+a.MIN_MET1:y+a.UP_MET_TRAN_DOWN))
dbCreateRect(cv b.con list(x+a.LEFT_CON_EDGE:y+a.UP_CON_EDGE x+a.LEFT_CON_EDGE+a.MIN_CON:y+a.UP_CON_EDGE+a.MIN_CON))
dbCreateRect(cv b.con list(x+a.LEFT_CON_EDGE:y-a.DOWN_CON_EDGE x+a.LEFT_CON_EDGE+a.MIN_CON:y-a.DOWN_CON_EDGE+a.MIN_CON))
)

/* HORIZONTAL POLYSILICON LINES */

procedure( polysilicon1(p q)
poly1_Height = a.MIN_POL
poly1_Width = q*a.COL_SPA
x=0
y=0
for( row l p
dbCreateRect(cv b.poly list(x:y x+poly1_Width:y+poly1_Height))
y=y+a.ROW_SPA
)
x=x-1

;;;;;IF YOU USE N-WELL PUT COMMENT BEFORE THE NEXT LINE;;;;;

dbCreateRect(cv b.pwell list(x:-a.DOWN_MET_TRAN_UP-3*a.MIN_MET1 x+poly1_Width+a.MIN_DEV_PNELL:y))

;;;;;DEPENDING ON WHAT KIND OF DOPING COMMENT/UNCOMMENT THE FOLLOWING LINES ;;;;;

dbCreateRect(cv b.nplus list(x:-a.ROW_SPA x+poly1_Width:y))
;dbCreateRect(cv b.pplus list(x:-a.ROW_SPA x+poly1_Width:y))
)

/* HORIZONTAL METAL2S AND THE LINE FOR THE CLOCK */
procedure(metall2_0(p q height XX)
metal2_Height = a.MIN_MET2
metal2_Width = q*a.COL_SPA
x = -(a.MIN_MET2-a.MIN_POL)/2
y = -(a.MIN_MET2-a.MIN_POL)/2
y1=y
count=1
for(row 1 p
if( count == 1 then
dbCreateRect(cv b.met2 list(x:y
x+metal2_Width+a.HALF_N_WIDTH:y+metal2_Height))
dbCreateRect(cv b.met2 list(x+metal2_Width+a.HALF_N_WIDTH:y1
x+metal2_Width+a.HALF_N_WIDTH+a.MIN_MET2:y1+height*a.ROW_SPA+a.ADJ_LAT_IN_HEIGHT+a.ADJ_LAT_Y_POS_DIRECT))
dbCreateRect(cv b.met2 list(x+metal2_Width+a.HALF_N_WIDTH+a.MIN_MET2:y1+height*a.ROW_SPA+a.ADJ_LAT_IN_HEIGHT+a.ADJ_LAT_Y_POS_DIRECT-a.MIN_MET2
XX+a.LAT_WIDTH DIRECT*0.7:y1+height*a.ROW_SPA+a.ADJ_LAT_IN_HEIGHT+a.ADJ_LAT_Y_POS_DIRECT))
else
dbCreateRect(cv b.met2 list(x:y x+metal2_Width:y+metal2_Height))
)
y=y+a.ROW_SPA
count=+count+1
)
*/ HORIZONTAL METALL WITHOUT PASSING OVER A SHORT-CIRCUIT */

procedure(met_hor(x y)
met_H = a.MIN_MET1
met_W = a.COL_SPA+a.MIN_CON
p=x+a.LEFT_MET_X
q=y+a.UP_CON_EDGE-(a.MIN_MET1-a.MIN_CON)/2
dbCreateRect(cv b.met1 list(p:q p+met_W:q+met_H))
)

*/ HORIZONTAL METALL PASSING OVER A SHORT-CIRCUIT */

procedure(met_hor_extension(x y)
met_H = a.MIN_MET1
met_W = 2*(a.COL_SPA+a.MIN_CON)
p=x+a.LEFT_MET_X
q=y+a.UP_CON_EDGE-(a.MIN_MET1-a.MIN_CON)/2
dbCreateRect(cv b.met1 list(p:q p+met_W:q+met_H))
)

*/ VERTICAL METALL */

procedure(mkmet(x y)
met_H = a.DOWN_DEV_Y+a.UP_DEV_Y-a.MIN_CON
met_W = a.MIN_MET1
p=x+a.LEFT_MET_X
q=y-(a.DOWN_DEV_Y+a.UP_DEV_Y-a.MIN_CON-a.MIN_CON_POL)/2
dbCreateRect(cv b.metl list(p:q p+met_width:q+met_height))
)

/* DIFFUSION */

procedure( dev2(z)
    width=z*a.COL_SPA-a.MIN_DEV_PPLUS*2
    height=a.MIN_MET1*3-2*a.MIN_DEV_CON
    x=a.MIN_DEV_PPLUS
    y=a.DOWN_MET_TRAN_UP-2*a.MIN_MET1
    dbCreateRect(cv b.dev list(x:y x+width:y+height))

    ;;;: DEPENDING ON WHAT KIND OF DOPING COMMENT/UNCOMMENT THE FOLLOWING
    ;;;
    dbCreateRect(cv b.pplus list(0:y-a.MIN_DEV_PPLUS
    x+width+a.MIN_DEV_PPLUS:y+height+a.MIN_DEV_PPLUS))
    ;dbCreateRect(cv b.nplus list(0:y-a.MIN_DEV_PPLUS
    x+width+a.MIN_DEV_PPLUS:y+height+a.MIN_DEV_PPLUS))
)

/* CONTACTS */

procedure( con(z)
    f=a.COL_SPA
    f1=a.DOWN_MET_TRAN_UP-a.MIN_MET1-a.MIN_DEV_CON
    con_width=a.MIN_CON
    con_height=a.MIN_CON
    for( row 1 z-1
        dbCreateRect(cv b.con list(f:f1 f+con_width:f1+con_height))
        f=f+a.COL_SPA
    )
)

/* METAL1 FOR GROUND LINE */

procedure( met(z)
    width=z*a.COL_SPA+a.HALF_P_WIDTH+a.HALF_N_WIDTH
    height=a.MIN_MET1*3
    x=a.HALF_N_WIDTH
    y=a.DOWN_MET_TRAN_UP-2*a.MIN_MET1-a.MIN_DEV_CON
    dbCreateRect(cv b.metl list(x:y x+width:y+height))
    dbCreateRect(cv b.metl list(x+width-height:y
    x+width:y+height+a.HALF_N_WIDTH))
)

/* SUBROUTINE FOR THE SHORT-CIRCUIT BETWEEN THE METAL2 AND
POLYSILICON */

Appendix C

The SKILL Codes

105
procedure( short_circuit1(x y)
Variable=x.MIN_CON+2*a.MIN_CON_POL
An_Variable=((a.MIN_VIA+2*a.MIN_VIA_MET2)-Variable)/2
dbCreateRect(cv b.poly list(x:y x*Variable:y+Variable))
dbCreateRect(cv b.met1 list(x:y x+Variable+a.MIN_MET1:y+Variable))
dbCreateRect(cv b.con list(x+a.MIN_CON_POL+y+a.MIN_CON_POL
x+a.MIN_CON_POL+a.MIN_CON:y+a.MIN_CON_POL+a.MIN_CON))
dbCreateRect(cv b.met2 list(x+a.MIN_CON+a.MIN_CON_POL:y- An_Variable
x+a.MIN_CON+a.MIN_CON_POL+y+a.MIN_VIA_CON+a.MIN_VIA+a.MIN_VIA_MET2:y-
An_Variable+a.MIN_VIA+2*a.MIN_VIA_MET2))
dbCreateRect(cv b.met1 list(x+a.MIN_CON+a.MIN_CON_POL:y- An_Variable
x+a.MIN_CON+a.MIN_CON_POL+y+a.MIN_VIA_CON+a.MIN_VIA+a.MIN_VIA_MET2:y-
An_Variable+a.MIN_VIA+2*a.MIN_VIA_MET2))
dbCreateRect(cv b.via list(x+a.MIN_CON+a.MIN_CON_POL+y-
An_Variable+a.MIN_VIA_CON:y-
x+a.MIN_CON+a.MIN_CON_POL+a.MIN_VIA_CON+y-
An_Variable+a.MIN_VIA_MET2)
)

C.1.2 Indirect mapping style

Lower portion

procedure( make() /* MAIN PROGRAM */
read_it()
hiOpenFileForm->file->value= ""
hiViewFile()
if( hiOpenFileForm->file->value != "" then
deOpen()
cv = dbOpenCellView(deOpenForm->deLibName->value deOpenForm->deCellName->
value
deOpenForm->deViewName->value nil "w")
import-infile(hiOpenFileForm->file->value)
inlimport-infile(hiOpenFileForm->file->value)

hiChangeBannerLabel(hiGetCurrentWindow() "Switching Tree (Mitel 1.5)" 0)

/* THIS SECTION IS READING THE FILE */

starting_list='()
n=()
Total_lines=0
b=getc(inport)
while( b != nil
 ;if( b == '\n
 if( b == 'E
 Total_lines=Total_lines+1
 )
b=getc(inport)
);
println(Total_lines)
close(inport)

for( j 1 Total_lines
 b=getc(inport)
 while( b != '\n
 ;while( b != 'E
 if( b != '\#' then
 starting_list=cons(b starting_list)
 )
b=getc(inport)
)
n=cons(reverse(starting_list) n)
starting_list=()()
)n=reverse(n)
println(length(n))
close(inport)

/* END OF READING FILE */

p2=sortl(n)
p3=reverse(p2)
println("LLLL")
println(length(car(p3)))
pX=remove_met(p3)
Row_Height=length(pX)
Row_Height1=Row_Height
ll=round(length(pX)/2)
p3=add_short_circuit(p3 ll)

Width=length(car(p3))*a.COL_SPA
P_SEND=copy(p3)
p4=remove_met(p3) /* length(p4) --> no of rows */
P_SEND1=copy(p4)

if( oddp(ll) then
 ll=ll+1
 )

l=length(p4)
p6=remove_tran(reverse(p3) p4)
m=length(car(p4))
poly_2(l m)

Met_count=metal_Count1(car(reverse(p3)) cadr(reverse(p4)))
car(reverse(p4))) /* down-going met */

/* DETERMINES THE WIDTH OF THE METALS BETWEEN THE UPPER AND LOWER SECTIONS */

if( evenp(Met_count)
  if( mod(Met_count-2 4) == 0 then
    Ref_pt=(Met_count-2)/2+1
    InterM_Widh=((Ref_pt-1)/2+1)*(a.MIN_MET2+a.MIN_MET2_MET2)
  )
)

if( evenp(Met_count)
  if( mod(Met_count-2 4) != 0 then
    Ref_pt=Met_count/2+1
    InterM_Widh=((Ref_pt-1)/2+1)*(a.MIN_MET2+a.MIN_MET2_MET2)
  )
)

if( oddp(Met_count)
  if( mod(Met_count-1 4) == 0 then
    Ref_pt=(Met_count-1)/2+1
    InterM_Widh=((Ref_pt-1)/2+1)*(a.MIN_MET2+a.MIN_MET2_MET2)
  )
)

if( oddp(Met_count)
  if( mod(Met_count-1 4) != 0 then
    Ref_pt=(Met_count-1)/2
    InterM_Widh=((Ref_pt-1)/2+2)*(a.MIN_MET2+a.MIN_MET2_MET2)
  )

/* ;;;;;;;;;;;;;;;;;;;;;;END METAL WIDTH SECTION;;;;;;;;;;;;;;;;;;;;;*/

height=length(p4)*a.ROW_SPA
Core_height=height+a.MIN_CON+a.MIN_VIA_CON+a.MIN_MET2_MET2
Tot_height=Core_height+InterM_Widh

/* CHECK THE MEANING FOR THE VARIABLES LATER IN THIS PROGRAM */

dmet2(l m 11 Tot_height Row_Height)
dev2(m)
conl(m)
metl(m Tot_height)

p7=tran_match0(p4)
p8=match_(p6)

REV_SHORT=car(reverse(p3))

column=length(car(p3))
end_of_cell=column*a.COL_SPA
mid_pt=mid_pt2(car(reverse(P_SEND)) cadr(reverse(P_SEND1))) Core_height
Tot_height Ref_pt Met_count car(reverse(P_SEND1))

cv_upper=make_shrink(Row_Height1 Width Ref_pt mid_pt ll 1)
r=dbCreateInst(cv cv_upper nil list(mid_pt-a.VERT_MET DIST*Ref_pt-
a.HOR_UPPER_ADJ Tot_height+a.VERT_UPPER_ADJ) "RO" 1)
dbSave(cv)
;
)

/*/ THE TOTAL NUMBER OF METALS GOING UP FROM THE UPPER T ROW OF THE LOWER
SECTIN */

procedure( Metal_Count1(x y y1)
  prog( ()
  count=0
  for(i 1 length(x)
  if( i > 1 then

    if( (nthelem(i x) == 'W && nthelem(i+1 x) == 'b) || (nthelem(i x) == 'b
    && nthelem(i-1 x) != 'W && nthelem(i-1 x) != 'X && (nthelem(i y) == 'G ||
    nthelem(i y) == 'R)) || (nthelem(i x) == 'b && nthelem(i-1 x) != 'W &&
    nthelem(i-1 x) != 'X && (nthelem(i y) == 'G || nthelem(i y) == 'R))
    || (nthelem(i x) == 'W && nthelem(i+1 x) == 'X && nthelem((i+2) x) == 'b)
    then

      count=count+1
    )
  )
  if( i ==1 then
  if( (nthelem(i x) == 'W && nthelem((i+1) x) == 'b) || (nthelem(i x) == 'b
  && (nthelem(i y) == 'G || nthelem(i y) == 'R)) || (nthelem(i x) == 'b &&
  nthelem(i-1 x) != 'W && nthelem(i-1 x) != 'X && (nthelem(i y) == 'G ||
  nthelem(i y) == 'R)) then

    count=count+1
  )
  )
  return(count)
  )
)

/*/ FINDS THE MIDDLE POINT OF THE UPPER SECTION */

procedure( mid_pt2(x y Core_height Tot_height Ref_pt Met_count yl_list)
  prog( ()
  Constant=2
  Constant_2=1
  init_count=0
  temp_x=x
  DIST=0
for(j 1 length(temp_x)
if( (ntthelem(j temp_x) == 'W' §§ ntthelem(j+1 temp_x) == 'b) || (ntthelem(j temp_x) == 'b' §§ ntthelem(j-1 temp_x) != 'W' §§ ntthelem(j-1 temp_x) != 'X' §§ (ntthelem(j y) == 'G' || ntthelem(j y) == 'R')) || (ntthelem(j temp_x) == 'b' §§ ntthelem(j-1 temp_x) != 'W' §§ ntthelem(j-1 temp_x) != 'X' §§ (ntthelem(j y1_list) == 'G' || ntthelem(j y1_list) == 'R')) || (ntthelem(j temp_x) == 'W' §§ ntthelem(j+1 temp_x) == 'X' §§ ntthelem(j+2 temp_x) == 'b) then

init_count=init_count+1
println(init_count)
)

if( init_count == Ref_pt §§ Constant == 2 then
middle_point=DIST
MID_PT=middle_point
end_point=middle_point
Constant=1
)

DIST=DIST+a.COL_SPA
)
diff=Met_count-Ref_pt

for( jj 1 Ref_pt
middle_point-middle_point-a.VERT_MET_DIST /* a.VERT_MET_DIST --
HOR_MET_DIST */
)
Y_Diff=(a.MIN_VIA+2*a.MIN_VIA_MET2+a.MIN_MET2_MET2)
X_Distance=0
Y_Distance=Tot_height
Y_R_Distance=Core_height

interm_met_count=0
interm_met_count_R=0

count=0

for(i 1 length(x)
if( (ntthelem(i x) == 'W' §§ ntthelem(i+1 x) == 'b) || (ntthelem(i x) == 'b' §§ ntthelem(i-1 x) != 'W' §§ ntthelem(i-1 x) != 'X' §§ (ntthelem(i y) == 'G' || ntthelem(i y) == 'R')) || (ntthelem(i x) == 'b' §§ ntthelem(i-1 x) != 'W' §§ ntthelem(i-1 x) != 'X' §§ (ntthelem(i y1_list) == 'G' || ntthelem(i y1_list) == 'R')) || (ntthelem(i x) == 'W' §§ ntthelem(i+1 x) == 'X' §§ ntthelem(i+2 x) == 'b) then

count=count+1
if( count == Met_count then

interm_met_right_met(X_Distance+a.ADJ_INTERM_MET end_point
Y_R_Distance Core_height Tot_height)
end_point=end_point+a.VERMET_DIST
)

if( count == Met_count-1 then

if( oddp(Met_count) then
interm_met_right_met(X_Distance+a.ADJ_INTERM_MET end_point
Y_R_Distance Core_height Tot_height)
end_point=end_point+a.VERMET_DIST
)

if( count ! = l && count <= Ref_pt then
interm_met_count=interm_met_count+1
if( mod(interm_met_count 2) ==l then
interm_met_left_dmet2(X_Distance+a.ADJ_INTERM_MET middle_point
Y_Distance-Y_Diff Core_height Tot_height)
else
interm_met_left_dmet2(X_Distance+a.ADJ_INTERM_MET middle_point
Y_Distance-Y_Diff Core_height Tot_height)
)
middle_point=middle_point+a.VERMET_DIST
)

if( evenp(Met_count) then
if( count > Ref_pt then
interm_met_count_R=interm_met_count_R+1
if( mod(interm_met_count_R 2) ==l then
interm_met_right_dmet(X_Distance+a.ADJ_INTERM_MET end_point Y_R_Distance
Core_height Tot_height)
else
interm_met_right_met(X_Distance+a.ADJ_INTERM_MET end_point
Y_R_Distance Core_height Tot_height)
)
end_point=end_point+a.VERMET_DIST
)

if( oddp(Met_count) then
if( count > Ref_pt && count < Met_count-1 then
interm_met_count_R=interm_met_count_R+1
if( mod(interm_met_count_R 2) ==l then
interm_met_right_dmet(X_Distance+a.ADJ_INTERM_MET end_point Y_R_Distance
Core_height Tot_height)
else
interm_met_right_met(X_Distance+a.ADJ_INTERM_MET end_point
Y_R_Distance Core_height Tot_height)
)
end_point=end_point+a.VERMET_DIST
if( count == 1 then
  intern_met_left_met2(x_distance+a.adj_intern_met middle_point
  tot_height-a.min_met2-a.min_met2 min_met2 core_height tot_height)
  middle_point = middle_point + a.vert_met_dist
)

if( intern_met_count_r == 0 & mod(intern_met_count_r 2) == 0 then
  y_r_distance = y_r_distance+a.min_met2+a.min_met2
)

if( mod(intern_met_count 2) == 0 then
  y_distance = y_distance-a.min_met2-a.min_met2
)

x_distance = x_distance+a.col_spa

return(mid_pt)

/* ADDING THE SHORT CIRCUITS BETWEEN THE POLYSILICON AND THE METAL2 */

procedure( add_short_circuit(a ll)
  prog( ()
pdev_temp_list = '() while( a != '('
temp_list = '() i=1
  car = car(a)
  length_car = length(car)
  add = round(length(car)/4)
  while( car != '()
    if( mod(i 7) == 0 then
      if( i == 11 || (i == 1 & i < length_car-11 & mod(i 14) == 1) || i == length_car-11)
        temp_list = cons(car(car) temp_list)
      temp_list = cons('x temp_list)
    else
      temp_list = cons(car(car) temp_list)
    )
i = i+1
ii = i-1
  car = cdr(car)
)
temp_list=reverse(temp_list)
pdev_temp_list=cons(temp_list pdev_temp_list)
    a=cdr(a)
)
pdev_temp_list=reverse(pdev_temp_list)

return(pdev_temp_list)
)

(/^

/* CHECKS THE MAIN LIST AND PLACES THE TRANSISTORS, VERTICAL METALS
AND THE SHORT-CIRCUITS */

procedure( tran_match0(x)
    j=1
    k=1
    s=0
    while( x != '()'
        a=car(x)
        p=length(a)
        for( i 1 p
            b=nthelem(i a)
            if( (j==1 || mod(j 2)==1) then
                if(b == 'G then
                    mkmasteru0((k-1)*a.COL_SPA (j-1)*a.ROW_SPA )
                    s=s+1
                
                /* 'COMMENT' THE FOLLOWING 'IF' STATEMENT IF YOU WANT TO USE CURRENT
                STEERING LATCH */
                if(b == 'R then
                    mkmasteru0((k-1)*a.COL_SPA (j-1)*a.ROW_SPA )
                    s=s+1
                )
            
            /****THE FOLLOWING SECTION IS FOR THE CURRENT STEERING LATCH IF YOU WANT
            TO USE IT 'UNCOMMENT' THE LINES*****/

            /******************************BEGIN*******************************/
            ;if(b == 'R & & j != 1 then
            ;mkmasteru0((k-1)*a.COL_SPA (j-1)*a.ROW_SPA )
            ;s=s+1
            ;)
            ;if(b == 'R & & j == 1 then
            ;mkmet_ground((k-1)*a.COL_SPA (j-1)*a.ROW_SPA )
            ;s=s+1
            ;)
            
            /**********************END OF CURRENT LATCH SECTION*************/

            if(b == 'W mkmet((k-1)*a.COL_SPA (j-1)*a.ROW_SPA ))

Appendix C The SKILL Codes
if(b == 'X shorint1((k-1)*a.COL_SPA+(a.RIGHT_DEV_X-
  a.COL_SPA+0.8*a.MIN_MET1) (j-1)*a.ROW_SPA-(a.MIN_CON+2*a.MIN_CON_POL-    
  a.MIN_POL)/2 ))
}

if( (j==2 || mod(j 2)==0) then
if(b == 'G then
mkmasterd((k-1)*a.COL_SPA (j-1)*a.ROW_SPA )
s=s+1
)

if(b == 'R then
mkmasterd((k-1)*a.COL_SPA (j-1)*a.ROW_SPA )
s=s+1
)

if(b == 'W mkmet((k-1)*a.COL_SPA (j-1)*a.ROW_SPA ))

if(b == 'X shorint1((k-1)*a.COL_SPA+(a.RIGHT_DEV_X-
  a.COL_SPA+0.8*a.MIN_MET1) (j-1)*a.ROW_SPA-(a.MIN_CON+2*a.MIN_CON_POL-    
  a.MIN_POL)/2 ))
)

k=k+1

j=j+1
x=cdr(x)
k=l
println(s) /* end while */

/* CHECKS THE MAIN LIST AND PLACES THE HORIZONTAL METALS */
procedure( match_(y)
  j=1
  k=1
  while( y != '() a=car(y)
p=length(a)
for( i l p
  b=nthelem(i a)
b=nthelem(i+1 a)
if(j then

  if(b == 'W && bl != 'X then
    met_hor((k-1)*a.COL_SPA (j-1)*a.ROW_SPA*2) )

Appendix C  The SKILL Codes
if(b == 'W && bl == 'X then
met_hor_extension((k-1)*a.COL_SPA (j-1)*a.ROW_SPA*2))
k=k+1
}
j=j+1
y= cdr(y)
x=1
} /* end while */

/ * TRANSISTOR IN THE UPPER ROW */

procedure( mkmasteru0(x y)

dbCreateRect(cv b.poly list(x:y x+a.POL_LENGTH:y+a.MIN_POL))
dbCreateRect(cv b.dev list(x+a.LEFT_DEV_X:y-a.DOWN_DEV_Y
x+a.RIGHT_DEV_X:y+a.UP_DEV_Y))
dbCreateRect(cv b.met1 list(x+a.LEFT_DEV_X:y-a.DOWN_DEV_Y
x+a.RIGHT_DEV_X:y-a.DOWN_MET_Y))
dbCreateRect(cv b.met1 list(x+a.LEFT_DEV_X:y+a.UP_MET_Y
x+a.RIGHT_DEV_X:y+a.UP_DEV_Y))
dbCreateRect(cv b.met1 list(x+a.LEFT_MET_X:y-a.DOWN_MET_TRAN_UP
x+a.LEFT_MET_X+a.MIN_MET1:y-a.UP_MET_TRAN_UP))
dbCreateRect(cv b.con list(x+a.LEFT_CON_EDGE:y+a.UP_CON_EDGE
x+a.LEFT_CON_EDGE+a.MIN_CON:y+a.UP_CON_EDGE+a.MIN_CON))
dbCreateRect(cv b.con list(x+a.LEFT_CON_EDGE:y-a.DOWN_CON_EDGE
x+a.LEFT_CON_EDGE+a.MIN_CON:y-a.DOWN_CON_EDGE+a.MIN_CON))
)

/ * TRANSISTOR IN THE LOWER ROW */

procedure( mkmasterd(x y)

dbCreateRect(cv b.poly list(x:y x+a.POL_LENGTH:y+a.MIN_POL))
dbCreateRect(cv b.dev list(x+a.LEFT_DEV_X:y-a.DOWN_DEV_Y
x+a.RIGHT_DEV_X:y+a.UP_DEV_Y))
dbCreateRect(cv b.met1 list(x+a.LEFT_DEV_X:y-a.DOWN_DEV_Y
x+a.RIGHT_DEV_X:y-a.DOWN_MET_Y))
dbCreateRect(cv b.met1 list(x+a.LEFT_DEV_X:y+a.UP_MET_Y
x+a.RIGHT_DEV_X:y+a.UP_DEV_Y))
dbCreateRect(cv b.met1 list(x+a.LEFT_MET_X:y-a.DOWN_MET_TRAN_DOWN
x+a.LEFT_MET_X+a.MIN_MET1:y+a.UP_MET_TRAN_DOWN))
dbCreateRect(cv b.con list(x+a.LEFT_CON_EDGE:y+a.UP_CON_EDGE
x+a.LEFT_CON_EDGE+a.MIN_CON:y+a.UP_CON_EDGE+a.MIN_CON))
dbCreateRect(cv b.con list(x+a.LEFT_CON_EDGE:y-a.DOWN_CON_EDGE
x+a.LEFT_CON_EDGE+a.MIN_CON:y-a.DOWN_CON_EDGE+a.MIN_CON))
)

/ * HORIZONTAL POLYSILICON LINES */

procedure( poly_2(p q)

Appendix C  The SKILL Codes

115
b.poly_Height = a.MIN_POL
b.poly_Width = q*a.COL_SPA
x=0
y=0
for(row l p
dbCreateRect(cv b.poly list(x:y x+b.poly_Width:y+b.poly_Height))
y=y+a.ROW_SPA
}
x=x-l

;;;;; if you use n-well put comment before the next line;;;;;

dbCreateRect(cv b.pwell list(x:-a.DOWN_MET_TRAN_UP-3*a.MIN_MET1
x+b.poly_Width+a.MIN_DEV_PWELL:y))

;;;;; depending on what kind of doping you are using comment/uncomment the
following lines ;;;;;

dbCreateRect(cv b.nplus list(x:-a.ROW_SPA x+b.poly_Width:y))
;dbCreateRect(cv b.pplus list(x:-a.ROW_SPA x+b.poly_Width:y))
)

/* HORIZONTAL METAL2S AND THE LINE FOR THE CLOCK */

procedure( dmet2(p q ll height Row)
dmet_Height =a.MIN_MET2
dmet_Width = q*a.COL_SPA
x ==-(a.MIN_MET2-a.MIN_POL)/2
y ==-(a.MIN_MET2-a.MIN_POL)/2
Diff==(a.MIN_MET2-a.MIN_VIA)
PP=p
counter=1

for(row l p

if( row <= ll+1 then
if( counter l=1 then

dbCreateRect(cv b.met2 list(x:y x+dmet_Width:y+dmet_Height))

dbCreateRect(cv b.met2 list(x:y x+a.MIN_MET2:height))
else

dbCreateRect(cv b.met2 list(x:y
x+dmet_Width+2*a.MIN_VIA_MET2+a.MIN_MET2_MET2:y+dmet_Height))

dbCreateRect(cv b.met2 list(x:y x+a.MIN_MET2:height))

dbCreateRect(cv b.met2
list(x:dmet_Width+2*a.MIN_VIA_MET2+a.MIN_MET2_MET2:y
x+dmet_Width+a.MIN_MET2+2*a.MIN_VIA_MET2+a.MIN_MET2_MET2:height))
)
dmet_Width=dmet_Width-a.MIN_VIA-2*a.MIN_VIA_MET2-a.MIN_MET2_MET2
x=x-a.MIN_VIA+2*a.MIN_VIA_MET2+a.MIN_MET2_MET2
counter=counter+1
)

Appendix C The SKILL Codes
if ( row > ll+1 then
dbCreateRect(cv b.met2 list(x:y x+dmet_width+Diff:y+dmet_height))
dbCreateRect(cv b.met2 list(x+dmet_width+Diff-a.min_met2:y
x+dmet_width+Diff:height))
dmet_width=dmet_width-a.min_met2_met2-2*a.min_via_met2-a.min_via
)
y=y+a.row_spa
}
corner_begin(height-a.adj intern met ll+1 x-a.min_via-2*a.min_via_met2-
a.min_met2_met2 row)
corner_endl(height-a.adj intern met pp-ll-1
x+dmet_width+2*a.min_via_met2-a.min_met2_met2 row)

/*@ HORIZONTAL METALL WITHOUT PASSING OVER A SHORT-CIRCUIT */

procedure( met_hor(x y)
met_height = a.min_met1
met_width = a.col_spa+a.min_con
p=x+a.left_met_x
q=y+a.up_con_edge-(a.min_met1-a.min_con)/2
dbCreateRect(cv b.met1 list(p:q p+met_width:q+met_height))
)

/*@ HORIZONTAL METALL PASSING OVER A SHORT-CIRCUIT */

procedure( met_hor_extension(x y)
met_height = a.min_met1
met_width = 2*(a.col_spa+a.min_con)
p=x+a.left_met_x
q=y+a.up_con_edge-(a.min_met1-a.min_con)/2
dbCreateRect(cv b.met1 list(p:q p+met_width:q+met_height))
)

/*@ VERTICAL METAL */

procedure( mkmet(x y)
met_height=a.down_dev_y+a.up_dev_y-a.min_con
met_width=a.min_met1
p=x+a.left_met_x
q=y-(a.down_dev_y+a.up_dev_y-a.min_con-pol)/2
dbCreateRect(cv b.met1 list(p:q p+met_width:q+met_height))
)

/*@ PLACING A METAL INSTEAD OF A GROUND SWITCH.. THIS IS TRUE FOR CURRENT STEERING LATCH */

procedure( mkmet_ground(x y)
mw_height=2*(a.down_dev_y+a.up_dev_y-a.min_con)
met_Width=a.MIN_MET1
p=x+a.LEFT_MET_X
q=y-a.DOWN_MET_TRAN_UP-2*a.MIN_MET1
dbCreateRect(cv "metall" list(p:q p+met_Width:q+met_Height))
)

/* DIFFUSION */

procedure( dev2(z)
width=z*a.COL_SPA-a.MIN_DEV_PPLUS*2
height=a.MIN_MET1*3-2*a.MIN_DEV_CON
x=a.MIN_DEV_PPLUS
y=a.DOWN_MET_TRAN_UP-2*a.MIN_MET1
dbCreateRect(cv b.dev list(x:y x+width:y+height))
)

; ; ; ; DEPENDING ON WHAT KIND OF DOPING YOU ARE USING COMMENT/UNCOMMENT THE FOLLOWING LINES ; ; ; ;

; ; dbCreateRect(cv b.nplus list(0:y-a.MIN_DEV_PPLUS
x+width+a.MIN_DEV_PPLUS:y+height+a.MIN_DEV_PPLUS))
; ; dbCreateRect(cv b.pplus list(0:y-a.MIN_DEV_PPLUS
x+width+a.MIN_DEV_PPLUS:y+height+a.MIN_DEV_PPLUS))
)

/* CONTACTS */

procedure( con1(z)
    f=a.COL_SPA
fl=a.DOWN_MET_TRAN_UP-a.MIN_MET1-a.MIN_DEV_CON
    con_width=a.MIN_CON
    con_height=a.MIN_CON
for( row 1 z-2
    dbCreateRect(cv b.con list(f:fl f+con_width:fl+con_height))
    f=f+a.COL_SPA
)
)

/* METALL FOR GROUND LINE */

procedure( metall(z upper)
    upper=upper+a.GROUND_UPPER
width=z*a.COL_SPA+4*a.MIN_MET1
height=4*a.MIN_MET1
x=-3*a.MIN_MET1
y=a.DOWN_MET_TRAN_UP-2*a.MIN_MET1-a.MIN_DEV_CON
dbCreateRect(cv b.met1 list(x:y x+width:y+height))
; dbCreateRect(cv b.met1 list(x-a.MIN_MET1:y x+2*a.MIN_MET1:y+upper+2.6))
; dbCreateRect(cv b.met1 list(x+width:y x+width+3*a.MIN_MET1:y+upper+2.6))
; dbCreateRect(cv b.met1 list(x:y+upper-2.35 x+380:y+upper+2.7))
; dbCreateRect(cv b.met1 list(x+width+4:y+upper-2.35 x+width-280:y+upper+2.7))
/* SUB-Routine FOR THE SHORT-CIRCUIT BETWEEN THE METAL2 AND POLYSILICON */

procedure(shorting1(x y)
Variable=a.MIN_CON+2*a.MIN_CON_POL
An_Variable=((a.MIN_VIA+2*a.MIN_VIA_MET2)-Variable)/2
dbcCreateRect(cv b.poly list(x;y x+Variable;y+Variable))
dbcCreateRect(cv b.metl list(x;y x+Variable+a.MIN_MET1;y+Variable))
dbcCreateRect(cv b. con list(x+a.MIN_CON_POL;y+a.MIN_CON_POL
x+a.MIN_CON_POL+a.MIN_CON;y+a.MIN_CON_POL+a.MIN_CON))

dbcCreateRect(cv b.met2 list(x+a.MIN_CON+a.MIN_CON_POL:y-An_Variable
x+a.MIN_CON+a.MIN_CON_POL+a.MIN_VIA_CON+a.MIN_VIA+2*a.MIN_VIA_MET2:y-
An_Variable+a.MIN_VIA+2*a.MIN_VIA_MET2))

dbcCreateRect(cv b.metl list(x+a.MIN_CON+a.MIN_CON_POL:y-An_Variable
x+a.MIN_CON+a.MIN_CON_POL+a.MIN_VIA_CON+a.MIN_VIA+2*a.MIN_VIA_MET2:y-
An_Variable+a.MIN_VIA+2*a.MIN_VIA_MET2))

dbcCreateRect(cv b. via list(x+a.MIN_CON+a.MIN_CON_POL+a.MIN_VIA_CON:y-
An_Variable+a.MIN_VIA_MET2
x+a.MIN_CON+a.MIN_CON_POL+a.MIN_VIA_CON+a.MIN_VIA:y-
An_Variable+a.MIN_VIA_MET2+a.MIN_VIA))
)

/* THE CORNER CELL AT THE LEFT END */

procedure(corner_begin(y XX x Row)
println("SKS")
Lowest_Point=y+2*a.MIN_MET2
Lowest_Point1=y
End_pt=x+a.AD J DIST_INV LAST_CORNER_LEFT
p=round(XX/2)
j=0

for(j 1 XX-1
x=x-a.MIN_VIA-2*a.MIN_VIA_MET2-a.MIN_MET2_MET2
y=y+a.MIN_MET2+a.MIN_MET2_MET2
)

ii=0
for(i 1 round(XX)

dbcCreateRect(cv b.met2 list(x:Lowest_Point1
x+a.MIN_MET2:y+a.IN V_IN_HEIGHT_FROM_REF+a.MIN_MET2))

dbcCreateRect(cv b.met2 list(x:y+a.IN V_IN_HEIGHT_FROM_REF
End_pt:y+a.IN V_IN_HEIGHT_FROM_REF+a.MIN_MET2))

if( mod(i 4) ==3 || mod(i 4) ==0 then

vial(y+a.IN V_IN_HEIGHT_FROM_REF-a.MIN_VIA-a.MIN_VIA MET2 x+ii)


dbcCreateRect(cv b.metl list(x+ii:y+a.IN V_IN_HEIGHT_FROM_REF

Appendix C The SKILL Codes 119
x+ii+a.MIN_MET1:Lowest_Point+p*2*(a.MIN_MET2+a.MIN_MET2_MET2)+a.INV_IN_HEIGHT_FROM_REF+2*a.MIN_MET2))

vial(Lowest_Point+p*2*(a.MIN_MET2+a.MIN_MET2_MET2)+a.INV_IN_HEIGHT_FROM_REF+2*a.MIN_MET2-a.MIN_VIA-a.MIN_VIA_MET2 x+ii)

j=j+1
)
y=y-a.MIN_MET2-a.MIN_MET2_MET2
x=x+a.MIN_VIA+2*a.MIN_VIA_MET2+a.MIN_MET2_MET2
)
)

/* THE CORNER CELL AT THE RIGHT END */

procedure( corner_end1(y XX x Row)
println("SKSI")
println(XX)
Row_Height=(Row-1)/2
Lowest_Point=y+2*a.MIN_MET2
Lowest_Point1=y
p=round(XX/2)
End_pt=x-a.ADJ_DIST_INV_LAST_CORNER_RIGHT
ii=0

for(i l round(XX)
dbCreateRect(cv b.met2 list(x:Lowest_Point1 x+a.MIN_MET2:y+a.INV_IN_HEIGHT_FROM_REF+a.MIN_MET2))
dbCreateRect(cv b.met2 list(x:y+a.INV_IN_HEIGHT_FROM_REF
End_pt:y+a.INV_IN_HEIGHT_FROM_REF+a.MIN_MET2))

if( oddp(Row_Height) then
if( mod(i 4) ==1 || mod(i 4) ==0 then
vial(y+a.INV_IN_HEIGHT_FROM_REF-a.MIN_MET2_MET2 x+ii)

dbCreateRect(cv b.met1 list(x+ii:y+a.INV_IN_HEIGHT_FROM_REF
x+ii+a.MIN_MET1:Lowest_Point+p*2*(a.MIN_MET2+a.MIN_MET2_MET2)+a.INV_IN_HEIGHT_FROM_REF+2*a.MIN_MET2))

vial(Lowest_Point+p*2*(a.MIN_MET2+a.MIN_MET2_MET2)+a.INV_IN_HEIGHT_FROM_REF+2*a.MIN_MET2-a.MIN_VIA-a.MIN_VIA_MET2 x+ii)
)
)

if( evenp(Row_Height) then
if( mod(i 4) ==2 || mod(i 4) ==3 then
vial(y+a.INV_IN_HEIGHT_FROM_REF-a.MIN_MET2_MET2 x+ii)

dbCreateRect(cv b.met1 list(x+ii:y+a.INV_IN_HEIGHT_FROM_REF
x+ii+a.MIN_MET1:Lowest_Point+p*2*(a.MIN_MET2+a.MIN_MET2_MET2)+a.INV_IN_HEIGHT_FROM_REF+2*a.MIN_MET2))

vial(Lowest_Point+p*2*(a.MIN_MET2+a.MIN_MET2_MET2)+a.INV_IN_HEIGHT_FROM_REF+2*a.MIN_MET2-a.MIN_VIA-a.MIN_VIA_MET2 x+ii)
)
)

if( oddp(Row_Height) then
if( mod(i 4) ==1 || mod(i 4) ==0 then
vial(y+a.INV_IN_HEIGHT_FROM_REF-a.MIN_MET2_MET2 x+ii)

dbCreateRect(cv b.met2 list(x+ii:y+a.INV_IN_HEIGHT_FROM_REF
x+ii+a.MIN_MET2:y+a.INV_IN_HEIGHT_FROM_REF+a.MIN_MET2))

dbCreateRect(cv b.met1 list(x+ii:y+a.INV_IN_HEIGHT_FROM_REF
x+ii+a.MIN_MET1:Lowest_Point+p*2*(a.MIN_MET2+a.MIN_MET2_MET2)+a.INV_IN_HEIGHT_FROM_REF+2*a.MIN_MET2))

vial(Lowest_Point+p*2*(a.MIN_MET2+a.MIN_MET2_MET2)+a.INV_IN_HEIGHT_FROM_REF+2*a.MIN_MET2-a.MIN_VIA-a.MIN_VIA_MET2 x+ii)
)
)

if( evenp(Row_Height) then
if( mod(i 4) ==2 || mod(i 4) ==3 then
vial(y+a.INV_IN_HEIGHT_FROM_REF-a.MIN_MET2_MET2 x+ii)

dbCreateRect(cv b.met1 list(x+ii:y+a.INV_IN_HEIGHT_FROM_REF
x+ii+a.MIN_MET1:Lowest_Point+p*2*(a.MIN_MET2+a.MIN_MET2_MET2)+a.INV_IN_HEIGHT_FROM_REF+2*a.MIN_MET2))

vial(Lowest_Point+p*2*(a.MIN_MET2+a.MIN_MET2_MET2)+a.INV_IN_HEIGHT_FROM_REF+2*a.MIN_MET2-a.MIN_VIA-a.MIN_VIA_MET2 x+ii)
)
)

if( oddp(Row_Height) then
if( mod(i 4) ==1 || mod(i 4) ==0 then
vial(y+a.INV_IN_HEIGHT_FROM_REF-a.MIN_MET2_MET2 x+ii)

dbCreateRect(cv b.met2 list(x+ii:y+a.INV_IN_HEIGHT_FROM_REF
x+ii+a.MIN_MET2:y+a.INV_IN_HEIGHT_FROM_REF+a.MIN_MET2))

dbCreateRect(cv b.met1 list(x+ii:y+a.INV_IN_HEIGHT_FROM_REF
x+ii+a.MIN_MET1:Lowest_Point+p*2*(a.MIN_MET2+a.MIN_MET2_MET2)+a.INV_IN_HEIGHT_FROM_REF+2*a.MIN_MET2))

vial(Lowest_Point+p*2*(a.MIN_MET2+a.MIN_MET2_MET2)+a.INV_IN_HEIGHT_FROM_REF+2*a.MIN_MET2-a.MIN_VIA-a.MIN_VIA_MET2 x+ii)
)
)

if( evenp(Row_Height) then
if( mod(i 4) ==2 || mod(i 4) ==3 then
vial(y+a.INV_IN_HEIGHT_FROM_REF-a.MIN_MET2_MET2 x+ii)

dbCreateRect(cv b.met1 list(x+ii:y+a.INV_IN_HEIGHT_FROM_REF
x+ii+a.MIN_MET1:Lowest_Point+p*2*(a.MIN_MET2+a.MIN_MET2_MET2)+a.INV_IN_HEIGHT_FROM_REF+2*a.MIN_MET2))

vial(Lowest_Point+p*2*(a.MIN_MET2+a.MIN_MET2_MET2)+a.INV_IN_HEIGHT_FROM_REF+2*a.MIN_MET2-a.MIN_VIA-a.MIN_VIA_MET2 x+ii)
)
)

if( oddp(Row_Height) then
if( mod(i 4) ==1 || mod(i 4) ==0 then
vial(y+a.INV_IN_HEIGHT_FROM_REF-a.MIN_MET2_MET2 x+ii)

dbCreateRect(cv b.met2 list(x+ii:y+a.INV_IN_HEIGHT_FROM_REF
x+ii+a.MIN_MET2:y+a.INV_IN_HEIGHT_FROM_REF+a.MIN_MET2))

dbCreateRect(cv b.met1 list(x+ii:y+a.INV_IN_HEIGHT_FROM_REF
x+ii+a.MIN_MET1:Lowest_Point+p*2*(a.MIN_MET2+a.MIN_MET2_MET2)+a.INV_IN_HEIGHT_FROM_REF+2*a.MIN_MET2))

vial(Lowest_Point+p*2*(a.MIN_MET2+a.MIN_MET2_MET2)+a.INV_IN_HEIGHT_FROM_REF+2*a.MIN_MET2-a.MIN_VIA-a.MIN_VIA_MET2 x+ii)
)
)

if( evenp(Row_Height) then
if( mod(i 4) ==2 || mod(i 4) ==3 then
vial(y+a.INV_IN_HEIGHT_FROM_REF-a.MIN_MET2_MET2 x+ii)

dbCreateRect(cv b.met1 list(x+ii:y+a.INV_IN_HEIGHT_FROM_REF
x+ii+a.MIN_MET1:Lowest_Point+p*2*(a.MIN_MET2+a.MIN_MET2_MET2)+a.INV_IN_HEIGHT_FROM_REF+2*a.MIN_MET2))

vial(Lowest_Point+p*2*(a.MIN_MET2+a.MIN_MET2_MET2)+a.INV_IN_HEIGHT_FROM_REF+2*a.MIN_MET2-a.MIN_VIA-a.MIN_VIA_MET2 x+ii)
)
)
HEIGHT_FROM_REF+2*a.MIN_MET2))

vial(Lowest_Point+p*2*(a.MIN_MET2+a.MIN_MET2_MET2)*a.INV_IN_HEIGHT_FROM _REF+2*a.MIN_MET2 x+ii)
)
)

y=y+a.MIN_MET2+a.MIN_MET2_MET2
x=x+a.MIN_VIA+2*a.MIN_VIA_MET2+a.MIN_MET2_MET2
)
dbCreateRect(cv b.met2 list(x:Lowest_Point1
x+a.MIN_MET2:y+a.INV_IN_HEIGHT_FROM_REF+a.MIN_MET2))
dbCreateRect(cv b.met2 list(x:y+a.INV_IN_HEIGHT_FROM_REF
End_pt:y+a.INV_IN_HEIGHT_FROM_REF+a.MIN_MET2))
)

/* METALLS FOR THE LEFT SET AT THE INTERMEDIATE CONNECTION */

procedure( intern_met_left_met2(x xl y yl HIGH) /* left */
dbCreateRect(cv b.met1 list(x:y x1:y+a.MIN_MET1))
dbCreateRect(cv b.met1 list(x:y+a.MIN_MET1 x+a.MIN_MET1:yl-
0.5*a.ROW_SPA-a.MIN_MET2_MET2-a.MIN_CON-a.MIN_VIA_CON))
dbCreateRect(cv b.met1 list(x1:y x1+a.MIN_MET1:HIGH+a.ADJ_INTERM_VERT))
)

/* METAL2S FOR THE LEFT SET AT THE INTERMEDIATE CONNECTION */

procedure( intern_met_left_dmet2(x xl y yl HIGH) /* left */
dbCreateRect(cv b.met2 list(x:y x1:y+a.MIN_MET2))
dbCreateRect(cv b.met2 list(x:y+a.MIN_MET2 x+a.MIN_MET2:yl-
0.5*a.ROW_SPA-a.MIN_MET2_MET2))
via2(yl-a.MIN_MET2_MET2-0.5*a.ROW_SPA x-0.5*a.MIN_CON)

vial(y xl)
dbCreateRect(cv b.met1 list(x1:y x1+a.MIN_MET1:HIGH+a.ADJ_INTERM_VERT))
)

/* METALLS FOR THE RIGHT SET AT THE INTERMEDIATE CONNECTION */

procedure( intern_met_right_met(x xl y yl HIGH) /* right */
dbCreateRect(cv b.met1 list(x1:y x+a.MIN_MET1:y+a.MIN_MET1))
dbCreateRect(cv b.met1 list(x:y+a.MIN_MET1 x+a.MIN_MET1:yl-
0.5*a.ROW_SPA-a.MIN_MET2_MET2-a.MIN_CON-a.MIN_VIA_CON))
dbCreateRect(cv b.met1 list(x1:y x1+a.MIN_MET1:HIGH+a.ADJ_INTERM_VERT))
)

/* METAL2S FOR THE RIGHT SET AT THE INTERMEDIATE CONNECTION */

procedure( intern_met_right_dmet(x xl y yl HIGH) /* right */
dbCreateRect(cv b.met2 list(x1:y x+a.MIN_MET2:y+a.MIN_MET2))
dbCreateRect(cv b.met2 list(x:y+a.MIN_MET2 x+a.MIN_MET2:yl-
0.5*a.ROW_SPA-a.MIN_MET2_MET2))
via2(y1-a.MIN_MET2_MET2-0.5*a.ROW_SPA x-0.5*a.MIN_CON)
via1(y x1)
dbCreateRect(cv b.met1 list(x1:y x1+a.MIN_MET1:HIGH+a.ADI(...)*/

procedure( via(y x)
dbCreateRect(cv b.met1 list(x:y
x+2*a.MIN_VIA_MET2+a.MIN_VIA:y+2*a.MIN_VIA_MET2+a.MIN_VIA))
dbCreateRect(cv b.met2 list(x:y
x+2*a.MIN_VIA_MET2+a.MIN_VIA:y+2*a.MIN_VIA_MET2+a.MIN_VIA))
dbCreateRect(cv b.via list(x+a.MIN_VIA_MET2:y+a.MIN_VIA_MET2
x+a.MIN_VIA_MET2+a.MIN_VIA:y+a.MIN_VIA_MET2+a.MIN_VIA))
)

procedure( via2(y x)
dbCreateRect(cv b.met1 list(x:y-a.MIN_MET1
x+2*a.MIN_VIA_MET2+a.MIN_VIA:y+2*a.MIN_VIA_MET2+a.MIN_VIA))
dbCreateRect(cv b.met2 list(x:y
x+2*a.MIN_VIA_MET2+a.MIN_VIA:y+2*a.MIN_VIA_MET2+a.MIN_VIA))
dbCreateRect(cv b.via list(x+a.MIN_VIA_MET2:y+a.MIN_VIA_MET2
x+a.MIN_VIA_MET2+a.MIN_VIA:y+a.MIN_VIA_MET2+a.MIN_VIA))
)

Upper portion

--------------------INDIRECT MAPPING STYLE;-----------------------------
--------------------UPPER PORTION;--------------------------------------
--------------------SHAKIL SIDDIQ;-------------------------------------
--------------------VLSI RESEARCH GROUP;-------------------------------

procedure( make_shrink(height Width Ref_pt mid_pt l1 total_corners) /*
main program .. making a link with 'opus_short_general_character_a' */
println(total_corners)
( prog ()
tot_height=0
clk_line_right=0
cv_shrink= dbOpenCellView(c.library c.test "layout" nil "w")
*/ 45 */
Left_end_pt=(-1)*(mid_pt-a.VERT_MET DIST*Ref_pt-
(l1+l1)*(a.MIN_MET2+a.MIN_VIA+2*a.MIN_VIA_MET2))
*/ 35 */
Right_end_pt=Width-mid_pt+a.VERT_MET DIST*Ref_pt-(total_corners-l1-
1)*(a.MIN_MET2+a.MIN_VIA+2*a.MIN_VIA_MET2)
inv_rotate= dbOpenCellView(c.library c.inv_small_indirect_rotate
"layout")

Appendix C The SKILL Codes 122
inv= dbOpenCellView(c.library c.inv_small_indirect "layout")
inv_via_rotate=dbOpenCellView(c.library c.inv_first "layout")
inv_via=dbOpenCellView(c.library c.inv_first_rotate "layout")
cv_shrink_latch= dbOpenCellView(c.library c.latch_indirect "layout")
cv_rotate_latch=dbOpenCellView(c.library c.latch_indirect_rotate "layout")
inv4= dbOpenCellView(c.library c.inv_big "layout")
inv4_rotate= dbOpenCellView(c.library c.inv_big_rotate "layout")

hiOpenFileForm->file->value= ""
hiViewfile()
if( hiOpenFileForm->file->value != "" then

import=infile(hiOpenFileForm->file->value)
import=infile(hiOpenFileForm->file->value)

hiChangeBannerLabel(hiGetCurrentWindow() x.technology_name 0)

/* THIS SECTION IS READING THE FILE */

starting_list='()

n='()
Total_lines=0
b=getc(import)
while( b != nil
 ;if( b == '\n
 if( b == 'E
 Total_lines=Total_lines+1
 )
b=getc(import)
)
println(Total_lines)
close(import)

for( j 1 Total_lines
 b=getc(import)
 while( b != '\n
 ;while( b != 'E

 if( b != 'E then

 starting_list=cons(b starting_list)
 )
b=getc(import)
)
n=cons(reverse(starting_list) n)
starting_list='()

reverse(n)
println(length(n))
close(import)

/* END OF READING FILE */

for(j 1 ll
 tot_height=tot_height+a.MIN_MET2+a.MIN_MET2_MET2
for(j 1 total_corners-1-1
clk_line_right=clk_line_right+a.MIN_MET2+a.MIN_MET2\_MET2
)

XX=shrink(n)
n=copy(XX)
p2=sorti\_shrink(n)
printf("length = %2d\n\n" length(p2))
p3=reverse(p2)
p4=remove\_met\_shrink(p3)

row\_height=height+length(p4)
height=round(row\_height/2)
printf("ROW\_HEIGHT")
printf(height)
p6=remove\_tran\_shrink(p2 p4)
l=length(p4)
m=length(car(p4))
poly2\_l(l m)
metail21\_shrink(l m)
p7=match\_shrink1(p4)
p8=matchi\_l\_shrink(p6)
latch=latch\_no(car(p2) cadr(reverse(p4)) length(p4) car(reverse(p4)))
out\_shrink1(car(p2) cadr(reverse(p4)) length(p4) car(reverse(p4)) latch)
column=length(car(p2))

x=-3*a.SMALL\_INV\_WIDTH
ii=1
yy=-(a.MIN\_MET2-a.MIN\_POL)/2

/* PLACES THE INVERTERS TO DRIVE THE TRANSISTORS IN THE UPPER SECTION OF
THE DESIGN */

for( j 1 l
if( evenp(height) then
r=dbCreateInst(cv_shrink inv\_via\_rotate nil list(x 0) "R0" 1)
else
r=dbCreateInst(cv_shrink inv\_via nil list(x 0) "R0" 1)
in\_first\_three=x+a.DIST\_REF\_IST\_IN\_X

$dbCreateRect(cv\_shrink "metal2" list(x+a.DIST\_OUT\_INV\_VIA\_X:0
x+a.DIST\_OUT\_INV\_VIA\_X+a.MIN\_MET2:yy))$
$dbCreateRect(cv\_shrink "metal2" list(x+a.DIST\_OUT\_INV\_VIA\_X:yy


0:yy+a.MIN_MET2))
yy=yy+a.ROW_SPA
dbCreateRect(cv_shrink "metal2" list(x:0 x+a.MIN_MET2:yy))

} )
x=x-a.SMALL_INV_WIDTH
ii=ii+1
yy=yy+a.ROW_SPA

for( i 1 length(p4)/2-1

if( evenp(height) then
if( evenp(ii) then
r=dbCreateInst(cv_shrink inv nil list(x 0) "RO" 1)
dbCreateRect(cv_shrink "metal2" list(x+a.DIST_SMALL_IN_OUT:0
x+a.DIST_SMALL_IN_OUT+a.MIN_MET2:yy))
dbCreateRect(cv_shrink "metal2" list(x+a.DIST_SMALL_IN_OUT:yy
0:yy+a.MIN_MET2))
yy=yy+a.ROW_SPA

dbCreateRect(cv_shrink "metal2" list(x:0 x+a.MIN_MET2:yy))
dbCreateRect(cv_shrink "metal2" list(x:yy 0:yy+a.MIN_MET2))
else
r=dbCreateInst(cv_shrink inv_rotate nil list(x 0) "RO" 1)
dbCreateRect(cv_shrink "metal2" list(x+a.DIST_SMALL_IN_OUT:0
x+a.DIST_SMALL_IN_OUT+a.MIN_MET2:yy))
dbCreateRect(cv_shrink "metal2" list(x+a.DIST_SMALL_IN_OUT:yy
0:yy+a.MIN_MET2))
yy=yy+a.ROW_SPA

dbCreateRect(cv_shrink "metal2" list(x:0 x+a.MIN_MET2:yy))
dbCreateRect(cv_shrink "metal2" list(x:yy 0:yy+a.MIN_MET2))
)
)

if( oddp(height) then
if( evenp(ii) then
r=dbCreateInst(cv_shrink inv_rotate nil list(x 0) "RO" 1)
dbCreateRect(cv_shrink "metal2" list(x+a.DIST_SMALL_IN_OUT:0
x+a.DIST_SMALL_IN_OUT+a.MIN_MET2:yy))
dbCreateRect(cv_shrink "metal2" list(x+a.DIST_SMALL_IN_OUT:yy
0:yy+a.MIN_MET2))
yy=yy+a.ROW_SPA

dbCreateRect(cv_shrink "metal2" list(x:0 x+a.MIN_MET2:yy))
dbCreateRect(cv_shrink "metal2" list(x:yy 0:yy+a.MIN_MET2))
else
r=dbCreateInst(cv_shrink inv nil list(x 0) "RO" 1)
dbCreateRect(cv_shrink "metal2" list(x+a.DIST_SMALL_IN_OUT:-(-a.MIN_MET2-
a.MIN_POL)/2 x+a.DIST_SMALL_IN_OUT+a.MIN_MET2:yy))
dbCreateRect(cv_shrink "metal2" list(x+a.DIST_SMALL_IN_OUT:yy
0:yy+a.MIN_MET2))
yy=yy+a.ROW_SPA

dbCreateRect(cv_shrink "metal2" list(x:0 x+a.MIN_MET2:yy))
dbCreateRect(cv_shrink "metal2" list(x:yy 0:yy+a.MIN_MET2))
)
)
\[ x = x - a \cdot \text{SMALL}\_\text{INV}\_\text{WIDTH} \]
\[ ii = ii + 1 \]
\[ yy = yy + a \cdot \text{ROW}\_\text{SPA} \]

\[ x = x - (a \cdot \text{LAT}\_\text{WIDTH}\_\text{INDIRECT} - a \cdot \text{SMALL}\_\text{INV}\_\text{WIDTH} + 3 \cdot a \cdot \text{MIN}\_\text{MET2}) \]
\[ y_{\text{height of latch in}} = a \cdot \text{MIN}\_\text{MET2}\_\text{MET2} \]
\[ y_{\text{height of latch in right}} = a \cdot \text{MIN}\_\text{MET2}\_\text{MET2} \]
\[ \text{left_latch_height} = y_{\text{height of latch in}} \]
\[ \text{right_latch_height} = y_{\text{height of latch in right}} \]

\[ \text{left_latch_count} = \text{round}(\text{latch/2}) \]

for (jj 1 left_latch_count
  left_latch_height = left_latch_height + a \cdot \text{MIN}\_\text{MET2} + a \cdot \text{MIN}\_\text{MET2}\_\text{MET2} \]

for (j 1 round(latch/2)
  right_latch_height = right_latch_height + a \cdot \text{MIN}\_\text{MET2} + a \cdot \text{MIN}\_\text{MET2}\_\text{MET2} \]

\[ /* \text{THIS SECTION IS FOR THE PLACEMENT OF THE LATCHES} */ \]

for (j 1 left_latch_count
  r = dbCreateInst(cv_shrink cv_shrink_latch nil list(x 0) "R0" 1)
  dbCreateRect(cv_shrink "metal2"
    list(x+a.DIST_IN_LAT_X:y_{\text{height of latch in}}+a.DIST_IN_LAT_Y
      a.SMOOTH\_\text{INV}\_\text{WIDTH}:y_{\text{height of latch in}}+a.DIST_INLAT_Y+a.MIN\_\text{MET2}
      x+a.DIST_INLAT_X+a.MIN\_\text{MET2}:y_{\text{height of latch in}}+a.DIST_INLAT_Y+a.MIN\_\text{MET2})
    )
  )

  dbCreateRect(cv_shrink "metal1"
    list(x+a.DIST_OUT_LAT_X:a.LAT_HEIGHT
      x+a.DIST_OUT_LAT_X+a.MIN\_\text{MET1}:left\_\text{latch\_height}+a.LAT\_\text{HEIGHT})
    via(left\_\text{latch\_height}+a.LAT\_\text{HEIGHT} x+a.DIST\_\text{OUT_LAT_X})
    y_{\text{height of latch in}}=y_{\text{height of latch in}}+a.MIN\_\text{MET2}+a.MIN\_\text{MET2}\_\text{MET2}
    x=x-a.LAT\_\text{WIDTH}\_\text{INDIRECT} \]

  )

  dbCreateRect(cv_shrink "metal2"
    list(x+a.LAT\_\text{WIDTH}\_\text{INDIRECT}:a.CLK\_\text{HEIGHT}\_\text{FROM}\_\text{REF}
      x+a.LAT\_\text{WIDTH}\_\text{INDIRECT}+a.MIN\_\text{MET2}:\text{tot\_height})
    )
  dbCreateRect(cv_shrink "metal2"
    list(x+a.LAT\_\text{WIDTH}\_\text{INDIRECT}+a.MIN\_\text{MET2}:\text{tot\_height}
      Left\_\text{end_pt}:\text{tot\_height}+a.MIN\_\text{MET2})
    )

  right\_\text{position}=\text{column}\_a.COL\_\text{SPA}\_\text{UPPER}
  right\_\text{point}=right\_\text{position}

  for (j 1 round(latch/2)
    r = dbCreateInst(cv_shrink cv_rotate_latch nil list(right\_\text{position} 0) "R0"

Appendix C  The SKILL Codes  126
dbCreateRect(cv_shrink "metal2"
list(right_position+a.DIST_IN_LAT_X_ROTATE+a.MIN_MET2:y_height_of_latch_in_right+a.DIST_IN_LAT_Y
right_point:y_height_of_latch_in_right+a.DIST_IN_LAT_Y+a.MIN_MET2))

dbCreateRect(cv_shrink "metal2"
list(right_position+a.DIST_IN_LAT_X_ROTATE:a.LAT_HEIGHT
right_position+a.DIST_IN_LAT_X_ROTATE+a.MIN_MET2:y_height_of_latch_in_right+a.DIST_IN_LAT_Y+a.MIN_MET2))

dbCreateRect(cv_shrink "metal1"
list(right_position+a.DIST_OUT_LAT_ROTATE_X:a.LAT_HEIGHT
right_position+a.DIST_OUT_LAT_ROTATE_X+a.MIN_MET1:right_latch_height+a.LAT_HEIGHT+3*a.MIN_MET1)

via(right_latch_height+a.LAT_HEIGHT+2*a.MIN_MET1
right_position+a.DIST_OUT_LAT_ROTATE_X)

y_height_of_latch_in_right=y_height_of_latch_in_right+a.MIN_MET2+a.MIN_MET2_MET2
right_position=right_position+a.LAT_WIDTH_INDIRECT)

dbCreateRect(cv_shrink "metal2" list(right_position-3*a.MIN_MET2:a.CLK_HEIGHT_FROM_REF right_position-3*a.MIN_MET2+a.MIN_MET2:clk_line_right))
dbCreateRect(cv_shrink "metal2" list(right_position-3*a.MIN_MET2:clk_line_right Right_end_pt:clk_line_right+a.MIN_MET2))

/* THIS IS FOR THE POWER LINE */

dbCreateRect(cv_shrink "metal1"
list(in_first_three+a.ADJ_DIST_FIRST_IN_POWER:y_height_of_latch_in_right+a.LAT_HEIGHT-a.MIN_MET1
right_point:y_height_of_latch_in_right+a.LAT_HEIGHT+4*a.MIN_MET1))
dbCreateRect(cv_shrink "active"
list(in_first_three+a.ADJ_DIST_FIRST_IN_POWER:y_height_of_latch_in_right+a.LAT_HEIGHT-a.MIN_MET1
right_point:y_height_of_latch_in_right+a.LAT_HEIGHT+4*a.MIN_MET1))

dbCreateRect(cv_shrink "nplus"
list(in_first_three+a.ADJ_DIST_FIRST_IN_POWER-a.MIN_DEV_NPLUS:y_height_of_latch_in_right+a.LAT_HEIGHT-a.MIN_MET1-a.MIN_DEV_NPLUS
right_point+a.MIN_DEV_NPLUS:y_height_of_latch_in_right+a.LAT_HEIGHT+4*a.MIN_MET1+a.MIN_DEV_NPLUS))

dbCreateRect(cv_shrink "metal1"
list(right_point+4*a.MIN_MET1:a.LAT_HEIGHT-2*a.MIN_MET1
right_point:y_height_of_latch_in_right+a.LAT_HEIGHT+4*a.MIN_MET1))
for( ii 1 3
    dbCreateRect(cv_shrink "metall" list(in_first_three:a.LAT_HEIGHT
    in_first_three*2:y_height_of_latch_in+a.LAT_HEIGHT))
    via_(_y_height_of_latch_in+y_height_of_latch_in+a.LAT_HEIGHT in_first_three)
    in_first_three+in_first_three+a.ADJ_DIST_FIRST_THREE_IN
    )

for( jj 1 round(right_point/a.SMALL_INV_WIDTH)
    substrate:in_first_three+a.SMALL_INV_WIDTH
    y_height_of_latch_in_right+a.LAT_HEIGHT+2*a.MIN_MET1
    in_first_three+in_first_three+a.SMALL_INV_WIDTH
    )

more_invert=height_length(p4)/2

if( oddp(more_invert) then
    Left_invert=round(more_invert/2)+1
    Right_invert=round(more_invert/2)
else
    Left_invert=more_invert/2
    Right_invert=more_invert/2
    jj=1
    x=x+a.INV3_WIDTH

invert_left_y=(Left_invert*2-1)*(a.MIN_MET2+a.MIN_MET2_MET2)

/* THIS SECTION IS FOR THE PLACEMENT OF THE INVERTERS DRIVING
TRANSISTORS AT THE LOWER SECTION */

for( j 1 Left_invert
    if( evenp(jj) then
        r=dbCreateInst(cv_shrink inv4_rotate nil list(x 0) "R0" 1)
        dbCreateRect(cv_shrink "metal2" list(x+a.DIST_IN_OUT_INV3:0
        x+a.DIST_IN_OUT_INV3+a.MIN_MET2:invert_left_y))
        dbCreateRect(cv_shrink "metal2"
        list(x+a.DIST_IN_OUT_INV3+a.MIN_MET2:invert_left_y
        Left_end_pt:invert_left_y+a.MIN_MET2))
        invert_left_y=invert_left_y+a.MIN_MET2-a.MIN_MET2_MET2
        dbCreateRect(cv_shrink "metal2" list(x:0 x+a.MIN_MET2:invert_left_y))
        dbCreateRect(cv_shrink "metal2" list(x+a.MIN_MET2:invert_left_y
        Left_end_pt:invert_left_y+a.MIN_MET2))
    else
        r=dbCreateInst(cv_shrink inv4 nil list(x 0) "R0" 1)
        dbCreateRect(cv_shrink "metal2" list(x+a.DIST_IN_OUT_INV3:0
        x+a.DIST_IN_OUT_INV3+a.MIN_MET2:invert_left_y))
        dbCreateRect(cv_shrink "metal2"
        list(x+a.DIST_IN_OUT_INV3+a.MIN_MET2:invert_left_y
        Left_end_pt:invert_left_y+a.MIN_MET2))
    )

Appendix C
The SKILL Codes
128
invert_left_y = invert_left_y - a.MIN_M2 - a.MIN_M2_M2
dbCreateRect(cv_shrink "metal2" list(x:0 x+a.MIN_M2:invert_left_y))
dbCreateRect(cv_shrink "metal2" list(x+a.MIN_M2:invert_left_y
Left_end_pt:invert_left_y+a.MIN_M2))
}
jj = jj + 1
x = x - a.INV3_WIDTH
invert_left_y = invert_left_y - a.MIN_M2 - a.MIN_M2_M2
}
invert_right_y = (a.MIN_M2 + a.MIN_M2_M2) * (Right_invert*2 - 1)
right_position = right_position + 3 * a.MIN_M2
for( j 1 Right_invert
if( evenp(jj) then
r = dbCreateInst(cv_shrink inv4 nil list(right_position 0) "R0" 1)
dbCreateRect(cv_shrink "metal2" list(right_position:0
right_position+a.MIN_M2:invert_right_y)
dbCreateRect(cv_shrink "metal2" list(right_position:invert_right_y
Right_end_pt:invert_right_y+a.MIN_M2))
invert_right_y = invert_right_y - a.MIN_M2 - a.MIN_M2_M2
dbCreateRect(cv_shrink "metal2" list(right_position+a.DIST_IN_OUT_INV3:0
right_position+a.DIST_IN_OUT_INV3+a.MIN_M2:invert_right_y)
dbCreateRect(cv_shrink "metal2"
list(right_position+a.DIST_IN_OUT_INV3:invert_right_y
Right_end_pt:invert_right_y+a.MIN_M2))
else
r = dbCreateInst(cv_shrink inv4_rotate
   nil list(right_position 0) "R0" 1)
dbCreateRect(cv_shrink "metal2" list(right_position:0
right_position+a.MIN_M2:invert_right_y)
dbCreateRect(cv_shrink "metal2" list(right_position:invert_right_y
Right_end_pt:invert_right_y+a.MIN_M2))
invert_right_y = invert_right_y - a.MIN_M2 - a.MIN_M2_M2
dbCreateRect(cv_shrink "metal2" list(right_position+a.DIST_IN_OUT_INV3:0
right_position+a.DIST_IN_OUT_INV3+a.MIN_M2:invert_right_y)
dbCreateRect(cv_shrink "metal2"
list(right_position+a.DIST_IN_OUT_INV3:invert_right_y
Right_end_pt:invert_right_y+a.MIN_M2))
}
right_position = right_position + a.INV3_WIDTH
jj = jj + 1
invert_right_y = invert_right_y - a.MIN_M2 - a.MIN_M2_M2
}
dbSave(cv_shrink)
}
return(cv_shrink)
/* FINDS THE NUMBER OF LATCHES NEEDED IN THE DESIGN */

procedure( latch_no(x y_list z y1_list) 
    prog() 
    no_latch=0 
    for(i 1 length(x) 
        if( (nthelem(i x) == 'W' && nthelem(i+1 x) == 'b') then 
            if( (nthelem(i x) == 'W && nthelem(i+1 x) == 'b' || (nthelem(i x) == 'b && nthelem(i+1 x) != 'W && nthelem(i+1 x) != 'X && nthelem(i y_list) == 'G || nthelem(i y_list) == 'R)) || (nthelem(i x) == 'b && nthelem(i+1 x) != 'W && nthelem(i+1 x) != 'X && nthelem(i y_list) == 'R) || (nthelem(i x) == 'W && nthelem(i+1 x) == 'X && nthelem(i+2 x) == 'b) then 
        no_latch=no_latch+1 
    end 
    return(no_latch) 
end

/* OUTPUT(S) OF THE DESIGN */

procedure( out_shrink1(x y_list z y1_list latch_no) 
    count=1 
    y_height_of_latch_in=a.ADJ_LEFT_LATCH_IN 
    y_height_of_latch_in_right=a.ADJ_RIGHT_LATCH_IN+(round(no_latch/2)-1)*a.MIN_MET2+a.MIN_MET2_MET2 
    for(i 1 length(x) 
        if( (nthelem(i x) == 'W && nthelem(i+1 x) == 'b) || (nthelem(i x) == 'b && nthelem(i+1 x) != 'W && nthelem(i+1 x) != 'X && nthelem(i y_list) == 'G || nthelem(i y_list) == 'R)) || (nthelem(i x) == 'b && nthelem(i+1 x) != 'W && nthelem(i+1 x) != 'X && nthelem(i y_list) == 'R) || (nthelem(i x) == 'W && nthelem(i+1 x) == 'X && nthelem(i+2 x) == 'b) then 
            y=(z-1)*a.ROW_SPA+a.UP_CON_EDGE-(a.MIN_MET1-a.MIN_CON)/2+a.MIN_VIA_MET2 
            x1=i*a.COL_SPA_UPPER 
            dbCreateRect(cv_shrink "via" list(x1:y x1+a.MIN_VIA:y+a.MIN_VIA)) 
            y2=(z-1)*a.ROW_SPA+a.UP_CON_EDGE-(a.MIN_MET1-a.MIN_CON)/2 
            x2=i*a.COL_SPA_UPPER-a.MIN_VIA_MET2 
            dbCreateRect(cv_shrink "metall" list(x2:y2 
            x2+a.MIN_VIA+2*a.MIN_VIA_MET2:y2+a.MIN_VIA+2*a.MIN_VIA_MET2)) 
            dbCreateRect(cv_shrink "metall2" list(x2:y2 
            x2+a.MIN_VIA+2*a.MIN_VIA_MET2:y2+a.MIN_VIA+2*a.MIN_VIA_MET2)) 
    end 
    if( count <= round(latch_no/2)+1 then 
        dbCreateRect(cv_shrink "metall2" list(x2:y2 
        x2+a.MIN_VIA+2*y_height_of_latch_in+2)) 
        dbCreateRect(cv_shrink "metall2" list(x2:y2+y_height_of_latch_in 
        
Appendix C The SKILL Codes 130
0:y2+y_height_of_latch_in+a.MIN_MET2)
y_height_of_latch_in=y_height_of_latch_in+a.MIN_MET2+a.MIN_MET2_MET2
else
dbCreateRect(cv_shrink "metal2" list(x2+a.MIN_VIA+2*a.MIN_VIA_MET2:y2
x2+a.MIN_VIA+2*a.MIN_VIA_MET2-
a.MIN_MET2:y2+y_height_of_latch_in_right+a.MIN_MET2))
dbCreateRect(cv_shrink "metal2" list(x2+a.MIN_VIA+2*a.MIN_VIA_MET2-
a.MIN_MET2:y2+y_height_of_latch_in_right
length(x)*a.COL_SPA_UPPER:y2+y_height_of_latch_in_right+a.MIN_MET2))
y_height_of_latch_in_right=y_height_of_latch_in_right-a.MIN_MET2-
a.MIN_MET2_MET2
)
count=count+1
)
)
)

/* REMOVES THE HORIZONTAL METAL INFO FROM THE MATRIX */

procedure( remove_met_shrink(x)
n=() 
while( x != '() 
if( member( 'R car(x) ) || member( 'G car(x) ) then
n=cons(car(x) n)
)
x=cdr(x)
)
reverse(n)
)

/* REMOVES THE TRANSISTOR INFO FROM THE MATRIX */

procedure( remove_trans_shrink(a b)
prog( ()
p=setof(e a (! member(e b)))
return(reverse(p))
)
)

/* SPLITTING THE T SET INTO TR AND TG ROWS */

procedure( sortl_shrink(x)
prog(()
in= '() 
y=1 /* G */
y1=2 /* R */
y2=4 /* R */
y3=5 /* G */
while( x != '() 
if( member( 'R car(x) ) || member( 'G car(x) ) then
x1=copy(car(x))
in=cons(car(x) cons(x1 in))
else

Appendix C The SKILL Codes 131
in=cons(car(x) in)
) /* end if */
x=cdr(x) /* end while */
x1=in

while( y < length(x1))
p=subst('b 'R nthelem(y x1))
x=insertR(p nthelem(y x1) x1)
x1=rember(nthelem(y x) x)
/*print(x1)*/
y=y+6

while( y1 < length(x1))
p=subst('b 'G nthelem(y1 x1))
x=insertR(p nthelem(y1 x1) x1)
x1=rember(nthelem(y1 x) x)
/*print(x1)*/
y1=y1+6

while( y2 < length(x1))
p=subst('b 'G nthelem(y2 x1))
x=insertR(p nthelem(y2 x1) x1)
x1=rember(nthelem(y2 x) x)
/*print(x1)*/
y2=y2+6

while( y3 < length(x1))
p=subst('b 'R nthelem(y3 x1))
x=insertR(p nthelem(y3 x1) x1)
x1=rember(nthelem(y3 x) x)
/*print(x1)*/
y3=y3+6

return(reverse(x1))
)
) /* end prog */

/* ADDS A NEW ELEMENT AT RIGHT OF AN EXISTING ELEMENT */

procedure(insertR(new old lat) /* insert an element at the right */
(cond
((equal lat '()) t)
((equal car(lat) old)
 cons(olde cons(new (cdr lat))))
)
(t cons(car(lat) insertR(new old (cdr lat)))))

>::::::::::::::::::::::::::; REMOVES AN ELEMENT FROM A LIST :::::::::::::;

procedure(rember(a lat) /* remove the element */
(cond
((equal lat '()) '())
((equal car(lat) a) cdr(lat))
(t cons(car(lat) rember(a (cdr lat))))

/* CHECKS THE MAIN LIST AND PLACES THE TRANSISTORS, VERTICAL METALS
AND THE SHORT-CIRCUITS */

procedure(match_shrinkl(x))
j=1
k=1
s=0
while( x != '() 
a=car(x)
p=length(a)
for( i 1 p
b=nthelem(i a)
if( (j==1 || mod(j 2)==1) then
if(b == 'G then
mkmasterd2_shrink((k-1)*a.COL_SPA_UPPER (j-1)*a.ROW_SPA )
s=s+1
)
if(b == 'R then
mkmasterd2_shrink((k-1)*a.COL_SPA_UPPER (j-1)*a.ROW_SPA )
s=s+1
)
if(b == 'W mkmet_shrink((k-1)*a.COL_SPA_UPPER (j-1)*a.ROW_SPA ))
)
if( (j==2 || mod(j 2)==0) then
if(b == 'G then
mkmasteru_shrink((k-1)*a.COL_SPA_UPPER (j-1)*a.ROW_SPA )
s=s+1
)
if(b == 'R then
mkmasteru_shrink((k-1)*a.COL_SPA_UPPER (j-1)*a.ROW_SPA )
s=s+1
)
if(b == 'W mkmet_shrink((k-1)*a.COL_SPA_UPPER (j-1)*a.ROW_SPA ))
)
if( mod(k 15) == 0
short_shrink((k-1)*a.COL_SPA_UPPER-2*a.MIN_MET2 (j-1)*a.ROW_SPA-
(a.MIN_CON+2*a.MIN_CON_POL-a.MIN_POL)/2 )
)
k=k+1
)
j=j+1
x=cdr(x)
k=1
)
println(s) /* end while */
PROCEDURE matching1_shrink(y)
  j=1
  k=1
  while( y != '()' )
    a=car(y)
    p=length(a)
    for( i 1 p
      b=thelem(i a)
    if(j then
     if(b == 'W met_hor_shrink((k-1)*a.COL SPA_UPPER (j-1)*a.ROW SPA*2) ))
       k=k+1
     j=j+1
     y=cdr(y)
    k=1
  /* end while */

PROCEDURE mkmasteru_shrink(x y)

  dbCreateRect(cv_shrink b.poly list(x:y x+a.POL_LENGTH:y+a.MIN_POL))
  dbCreateRect(cv_shrink b.dev list(x+a.LEFT_DEV_X:y-a.DOWN_DEV_Y
    x+a.RIGHT_DEV_X:y+a.UP_DEV_Y))
  dbCreateRect(cv_shrink b.met1 list(x+a.LEFT_DEV_X:y-a.DOWN_DEV_Y
    x+a.RIGHT_DEV_X:y+a.UP_DEV_Y))
  dbCreateRect(cv_shrink b.met1 list(x+a.LEFT_DEV_X:y+a.DOWN_MET_Y
    x+a.RIGHT_DEV_X:y+a.UP_DEV_Y))
  dbCreateRect(cv_shrink b.met1 list(x+a.LEFT_MET_X:y-a.DOWN_MET_TRAN_UP
    x+a.LEFT_MET_X+y-a.UP_MET_TRAN_UP))
  dbCreateRect(cv_shrink b.con list(x+a.LEFT_CON_EDGE:y+a.UP_CON_EDGE
    x+a.LEFT_CON_EDGE+y+a.UP_CON_EDGE+a.MIN_CON))
  dbCreateRect(cv_shrink b.con list(x+a.LEFT_CON_EDGE:y-a.DOWN_CON_EDGE
    x+a.LEFT_CON_EDGE+y-a.DOWN_CON_EDGE+a.MIN_CON))

PROCEDURE mkmasterd2_shrink(x y)

  dbCreateRect(cv_shrink b.poly list(x:y x+a.POL_LENGTH:y+a.MIN_POL))
  dbCreateRect(cv_shrink b.dev list(x+a.LEFT_DEV_X:y-a.DOWN_DEV_Y
    x+a.RIGHT_DEV_X:y+a.UP_DEV_Y))
  dbCreateRect(cv_shrink b.met1 list(x+a.LEFT_DEV_X:y-a.DOWN_DEV_Y
    x+a.RIGHT_DEV_X:y-a.DOWN_MET_Y))
  dbCreateRect(cv_shrink b.met1 list(x+a.LEFT_DEV_X:y+a.UP_MET_Y
    x+a.RIGHT_DEV_X:y+a.UP_DEV_Y))
  dbCreateRect(cv_shrink b.met1 list(x+a.LEFT_DEV_X:y+a.UP_MET_Y
    x+a.RIGHT_DEV_X:y+a.UP_DEV_Y))
  dbCreateRect(cv_shrink b.met1 list(x+a.LEFT_MET_X:y+a.DOWN_MET_TRAN_DOWN
    x+a.RIGHT_MET_X:y+a.DOWN_MET_TRAN_DOWN
    x+a.RIGHT_DEV_X:y+a.UP_DEV_Y))
  dbCreateRect(cv_shrink b.met1 list(x+a.LEFT_MET_X:y-a.DOWN_MET_TRAN_UP
    x+a.RIGHT_MET_X:y-a.UP_MET_TRAN_UP))
  dbCreateRect(cv_shrink b.con list(x+a.LEFT_CON_EDGE:y+a.UP_CON_EDGE
    x+a.LEFT_CON_EDGE+y+a.UP_CON_EDGE+a.MIN_CON))
  dbCreateRect(cv_shrink b.con list(x+a.LEFT_CON_EDGE:y-a.DOWN_CON_EDGE
    x+a.LEFT_CON_EDGE+y-a.DOWN_CON_EDGE+a.MIN_CON))

/* CHECKS THE MAIN LIST AND PLACES THE HORIZONTAL METALS */
/* HORIZONTAL POLYSILICON LINES */

procedure( poly2_1(p q)
poly1_Height = a.MIN_POL
poly1_Width = q*a.COLSPA_UPPER
x=0
y=0
for(row 1 p
dbCreateRect(cv_shrink "poly1" list(x:y x+poly1_Width:y+poly1_Height))
;dbCreateRect(cv_shrink "poly1" list(x:y x+poly1_Width:y+poly1_Height))
y=y+a.ROW SPA)
x=x-3*a.MIN_DEV_PWELL
dbCreateRect(cv_shrink "pwell" list(x:-a.DOWN_MET_TRAN_UP-a.MIN_MET1-a.MIN_DEV_CON x+poly1_Width:y))
dbCreateRect(cv_shrink "nplus" list(x:-a.ROW SPA x+poly1_Width:y))
)

/* HORIZONTAL METAL2S*/

procedure( metal21_shrink(p q)
m2_Height = a.MIN_MET2
m2_Width = q*a.COLSPA_UPPER
x =-(a.MIN_MET2-a.MIN_POL)/2
y =-(a.MIN_MET2-a.MIN_POL)/2

for(row 1 p
dbCreateRect(cv_shrink "metal2" list(x:y x+m2_Width:y+m2_Height))
y=y+a.ROW SPA)
)

/* HORIZONTAL METALL WITHOUT PASSING OVER A SHORT-CIRCUIT */

procedure( met_hor_shrink(x y)
m Heating = a.MIN_MET1
m_Width = a.COL SPA+a.MIN_CON
p=x+a.LEFT_MET_X
q=y+a.UP_MET_EDGE-(a.MIN_MET1-a.MIN_CON)/2+a.ROW SPA
dbCreateRect(cv_shrink "metall" list(p:q p+m_Width:q+m_Height))
)

/* VERTICAL METAL */

procedure( mkmet_shrink(x y)
met_Height=a.DOWN_DEV_Y+a.UP_DEV_Y-a.MIN_CON
met_Width=a.MIN_MET1
p=x+a.LLEFT_MET_X
q=y-(a.DOWN_DEV_Y+a.UP_DEV_Y-a.MIN_CON-a.MIN_CON_POL)/2
dbCreateRect(cv_shrink "metall" list(p:q p+met_Width:q+met_Height))

/* THIS SECTION SHRINKS DOWN THE FIRST SIX ROWS OF THE MATRIX */

procedure( shrink(a)
  ( prog ()

  declare(LIST[length(a)])
  for( i 0 length(a)-1
    LIST[i]='(0)
  )

  p=0
  P=car(last(a))
  Q=copy(P)
  while( P != '()    
    if( ( car(P) == 'G || car(P) == 'R) then
      p=p+1
    )
    P=cdr(P)
  )
  i=0
  pl=1
  declare( A[p])
  while( Q != '()    
    if( ( car(Q) == 'G || car(Q) == 'R) then
      A[i]=pl
      i=i+1
    )
    Q=cdr(Q)
    pl=pl+1
  )

  NUMBER=0

  while( a != '()    
    for( i 0 p-1
      if( member('G car(a)) || member('R car(a)) then
        if( (nthelem(A[i] car(a)) == 'G || nthelem(A[i] car(a)) == 'R) then
          LIST[NUMBER]=cons(nthelem(A[i] car(a)) LIST[NUMBER])
          LIST[NUMBER]=cons('b LIST[NUMBER])
          LIST[NUMBER]=cons('b LIST[NUMBER])
          LIST[NUMBER]=cons('b LIST[NUMBER])
          LIST[NUMBER]=cons('b LIST[NUMBER])
        )
      )
    )
  )
  if( (nthelem(A[i] car(a)) != 'G & nthelem(A[i] car(a)) != 'R) then
    for( i 1 5
LIST[NUMBER]=cons('b LIST[NUMBER])
)
)
}

if( (! member('G car(a))) || (! member('R car(a))) ) then
if( ntshaled(A[i] car(a)) == 'W then
LIST[NUMBER]=cons(ntshaled(A[i] car(a)) LIST[NUMBER])
LIST[NUMBER]=cons('W LIST[NUMBER])
LIST[NUMBER]=cons('W LIST[NUMBER])
LIST[NUMBER]=cons('W LIST[NUMBER])
LIST[NUMBER]=cons('W LIST[NUMBER])
)
if( ntshaled(A[i] car(a)) != 'W then
for(j = 1..5
LIST[NUMBER]=cons('b LIST[NUMBER])
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
An_Variable
x+a.MIN_CON+a.MIN_CON_POL+a.MIN_VIA_CON+a.MIN_VIA+a.MIN_VIA_MET2:y-
An_Variable+a.MIN_VIA+2*a.MIN_VIA_MET2))

dbCreateRect(cv_shrink "via"
list(x+a.MIN_CON+a.MIN_CON_POL+a.MIN_VIA_CON:y-
An_Variable+a.MIN_VIA_MET2
x+a.MIN_CON+a.MIN_CON_POL+a.MIN_VIA_CON+a.MIN_VIA:y-
An_Variable+a.MIN_VIA_MET2+a.MIN_VIA))

*/ METALL1+METAL2+VIA */

procedure( via_(y x)
dbCreateRect(cv_shrink "metall" list(x:y
x+2*a.MIN_VIA_MET2+a.MIN_VIA:y+2*a.MIN_VIA_MET2+a.MIN_VIA))
dbCreateRect(cv_shrink "metal2" list(x:y
x+2*a.MIN_VIA_MET2+a.MIN_VIA:y+2*a.MIN_VIA_MET2+a.MIN_VIA))
dbCreateRect(cv_shrink "via" list(x:a.MIN_VIA_MET2:y+a.MIN_VIA_MET2
x+a.MIN_VIA_MET2+a.MIN_VIA:y+a.MIN_VIA_MET2+a.MIN_VIA))
)

procedure( substrate(x y)

dbCreateRect(cv_shrink "contact" list(x:y x+a.MIN_CON:y+a.MIN_CON))
)

C.1.3 A Sample Technology File (Mitel 1.5 micron)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

/* Anything with .ADJ can be adjusted to get a better layout */

/* MAIN PROGRAM*/
procedure( read_it() )
/* WHAT TECHNOLOGY ARE YOU USING ? */
x.technology_name="Switching Tree (1.5 MITEL)"
/* THESE ARE FOR INDIRECT MAPPING STYLE */

/* THE CELL NAME FOR THE UPPER SECTION */
c.test="testx"
/* THE SMALL INVERTERS IN THE UPPER SECTION EXCEPT FOR THE FIRST ONE */
c.inv_small_indirect="inv"
c.inv_small_indirect_rotate="inv_rotate" /* ROTATED INVERTER */
/* FIRST SMALL INVERTER IN THE UPPER SECTION */
c.inv_first="inv_vial"
c.inv_first_rotate="inv_via_rotate" /* ROTATED FIRST INVERTER */
/* LATCH FOR THE INDIRECT MAPPING STYLE */
c.latch间接="latch_inv_novia"
/* SAME LATCH BUT ROTATED BY 180 DEGREE */
c.latch间接_rotate="latch_inv_rotate_novia"
/* THE BIGGER INVERTERS TO DRIVE THE TRANSISTORS IN THE LOWER SECTION */
c.inv_big="inv3"
/* ROTATED BIG INVERTERS */
c.inv_big_rotate="inv3_rotate"
/* THESE ARE FOR DIRECT MAPPING STYLE */
c.latch_direct_first="full_db_first"
c.latch_direct="full_db1"
c.library="New_lib" /* ALSO FOR INDIRECT MAPPING STYLE */
c.half_n_reverse="half_n_invert_reverse"
c.half_n_invert="half_n_invert"
c.half_p_invert="half_p_invert"
c.half_p_reverse="half_p_invert_reverse"
/* GIVE THE APPROPRIATE LAYER NAMES */
b.poly="poly1" /* POLYSILICON */
b.dev="active" /* DIFFUSION */
b.met1="metal1" /* METAL 1 */
b.met2="metal2" /* METAL 2 */
b.con="contact" /* CONTACT */
b.via="via" /* VIA */
b.pplus="pplus" /* P-DOPING */
b.nplus="nplus" /* N-DOPING */
b.pwell="pwell" /* P-WELL */
b.nwell="nwell" /* N-WELL */
/* SPACING BETWEEN THE ROWS ; check Figure C.2 */
a.ROW_SPA=8.7
/* SPACING BETWEEN THE COLUMNS; check Figure C.2 */
a.COL_SPA=6.9
/* SPACING IN THE UPPER SECTION*/
a.COL_SPA_UPPER=6.1
/* MINIMUM POLYSILICON WIDTH */
a.MIN_POL=1.5
/* MINIMUM METAL1 WIDTH*/
a.MIN_MET1=2
/* MINIMUM DISTANCE BETWEEN TWO METAL2S */
a.MIN_MET2_MET2=1.8
/* METAL2 WIDTH */
a.MIN_MET2=2
/* CONTACT LENGTH AND WIDTH*/
a.MIN_CON=1.5
/* MINIMUM N-DIFFUSION WIDTH*/
a.MIN_DEV=1.5
/* MINIMUM VIA WIDTH*/
a.MIN_VIA=1.8
/* MINIMUM DISTANCE BETWEEN THE POLYSILICON AND THE CONTACT */
a.MIN_POL_CON=0.8
/* MINIMUM DISTANCE BETWEEN THE VIA AND THE METAL1 */
a.MIN_VIA_MET1=0.8
/* MINIMUM DISTANCE BETWEEN THE VIA AND THE CONTACT */
a.MIN_VIA_CON=1.7
MIN_DEV_PWELL = 3
/* MINIMUM DISTANCE BETWEEN THE VIA AND METAL2 */
MIN_VIA_MET2 = 0.8
/* MINIMUM DISTANCE BETWEEN THE N-DIFFUSION AND THE CONTACT */
MIN_DEV_CON = 1
/* MINIMUM DISTANCE BETWEEN THE N-DIFFUSION AND THE N-DOPING */
MIN_DEV_NPLUS = 1.2
/* MINIMUM DISTANCE BETWEEN THE N-DIFFUSION AND THE P-DOPING */
MIN_DEV_PPLUS = 1.2
/* THE HEIGHT OF THE CLK INPUT TO THE LATCH FROM THE REF. POINT */
ADJ_LAT_IN_HEIGHT = 30.9
/* HALF_P_INVERTER WIDTH */
HALF_P_WIDTH = 40
/* HALF_N_INVERTER WIDTH */
HALF_N_WIDTH = 40
/* WIDTH OF THE LATCH FOR DIRECT STYLE ; HOPEFULLY YOU WILL USE THE SAME LATCH FOR BOTH DIRECT AND INDIRECT STYLE */
LAT_WIDTH_DIRECT = 90
/* WIDTH OF THE LATCH FOR INDIRECT STYLE */
LAT_WIDTH_INDIRECT = 90
/* SMALL INVERTER WIDTH IN THE UPPER SECTION */
SMALL_INV_WIDTH = 26
/* DISTANCE BETWEEN THE INPUT AND THE OUTPUT OF THE SMALL INVERTER INDIRECT */
DIST_SMALL_IN_OUT = 19.2
/* DISTANCE OF THE INPUT IN THE X DIRECTION OF THE LATCH FROM THE REF PT. INDIRECT */
DIST_IN_LAT_X = 55
/* DISTANCE OF THE INPUT IN THE Y DIRECTION OF THE LATCH FROM THE REF PT. INDIRECT */
DIST_IN_LAT_Y = 53
/* DISTANCE OF THE INPUT IN THE X DIRECTION OF THE LATC_H Rotate FROM THE REF PT. INDIRECT */
DIST_IN_LAT_X_ROTATE = 34
/* DISTANCE OF THE OUTPUT IN THE Y DIRECTION OF THE LATCH_ROTATE FROM THE REF PT. INDIRECT */
DIST_OUT_LAT_ROTATE_X = 82.7
/* DISTANCE OF THE OUTPUT IN THE Y DIRECTION OF THE LATCH FROM THE REF PT. INDIRECT */
DIST_OUT_LAT_X = 6.3
/* HEIGHT OF THE LATCH FROM REF INDIRECT */
LAT_HEIGHT = 54
/* FOR INDIRECT LATCH (MASTER) CLOCK LINE HEIGHT FROM REF. POINT */
CLK_HEIGHT_FROM_REF = 26.5
INV3_WIDTH = 40 /* HORIZONTAL DISTANCE BETWEEN THE INPUT OF THE LATCH AND THE REFERENCE POINT DIRECT */
DIST_IN_LAT = 50
/* DISTANCE BETWEEN THE INPUT AND THE OUTPUT OF THE BIG INVERTER */
DIST_IN_OUT_INV3 = 31
/* DISTANCE FROM THE FIRST INPUT TO THE POWER LINE */
ADJ_DIST_FIRST_IN_POWER = 26
/* DISTANCE FROM THE REF PT. TO THE OUTPUT OF THE INV_VIA WITH EXTENSIONS IN X DIRECTION */
a.DIST_OUT_INV_VIA_X=20
/* ADJUST THE INPUT LINE TO MATCH THE INPUT OF THE LEFT LATCH */
a.ADJ_LEFT_LATCH_IN=7.25
/* MATCH THE INPUT LINE OF THE RIGHT LATCH */
a.ADJ_RIGHT_LATCH_IN=7.25
/* DISTANCE BETWEEN THE FIRST THREE INPUTS */
a.ADJ_DIST_FIRST_THREE_IN=6
/* DISTANCE BETWEEN THE REF PT OF INV_VIA AND THE LINE FOR THE 1ST INPUT */
a.DIST_REF_IST_IN_X=32.15
/* THESE ARE THE REF POINTS FOR HALF_P_INVERT AND HALF_N_INVERT BOTH IN X AND Y DIRECTION */
a.REF_PT_HALF_P_Y=-0.1
a.REF_PT_HALF_P_X=0
a.REF_PT_HALF_N_Y=0
a.REF_PT_HALF_N_X=0
/* THE DEVIATION OF THE REF POINT OF THE DIRECT LATCH FROM THE 0,0 POINT IN Y DIRECTION */
a.ADI_LAT_Y_POS_DIRECT=15
/* CHECK THE DISTANCE OF THE INPUT OF THE LATCH TO THE REF. POINT OF THE LATCH DIRECT IN Y DIRECTION */
a.ADJ_INF_DIST_Y_DIRECT_LAT=18
/* ADJUSTMENT FOR THE VERTICAL METALS IN THE INTERMEDIATE CONNECTIONS */
a.ADJ_INTERM_VERT=6 /* start with 3a.MIN_MET1 */
/* IT MATCHES THE METALS IN THE INTERMEDIATE CONNECTIONS */
a.ADJ_INTERM_MET=3.45 /* start with one 3a.MIN_MET1 */
/* SPACING BETWEEN THE VERTICAL METALS IN THE INTERMEDIATE CONNECTIONS */
a.VER_MET_DIST=30.5
/* DISTANCE FROM THE LOWEST METAL IN THE CORNER CELL TO THE HIGHEST METAL IN THE INTERMEDIATE CONNECTION */
a.INV_IN_HEIGHT_FROM_REF=14 /* START WITH 7a.MIN_MET2 */
/* ADJUSTMENT FOR THE REF. PT OF THE UPPER SECTION */
a.VER_UPPER_ADJ=10.55
a.HOR_UPPER_ADJ=3.4
/* EXTENSION OF THE GROUND LINE FROM THE LOWER SECTION */
a.GROUND_UPPER=23
/* PROJECTION FROM THE CORNER CELL */
a.ADJ_DIST_INV_LAST_CORNER_LEFT=15 /* INITIALLY START WITH 10a.MIN_MET2 */
a.ADJ_DIST_INV_LAST_CORNER_RIGHT=20 /* INITIALLY START WITH 10a.MIN_MET2 */
/* TRANSISTOR SPECIFICATIONS-- LOOK AT THE Figure C.1 FOR DETAIL */
a.UP_CON_EDGE=4.3
a.DOWN_MET_tran_DOWN=3.4
a.LEFT_DEV_X=2.4
a.LEFT_MET_X=3.4
a.DOWN_DEV_Y=5.5
a.UP_DEV_Y=7
a.RIGHT_DEV_X=6.5
a.POL_LENGTH=8.9
a.DOWN_MET_Y=1.6
a.UP_MET_Y=3.1
a.UP_MET_tran_DOWN=14.8
a.DOWN_MET_tran_UP=13.6
a.UP_MET_tran_UP=2.4
Figure C.1 Transistor specifications

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>DOWN_CON_EDGE=4.3</td>
</tr>
<tr>
<td>b</td>
<td>LEFT_CON_EDGE=3.7</td>
</tr>
<tr>
<td>c</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td></td>
</tr>
<tr>
<td>g</td>
<td></td>
</tr>
<tr>
<td>h</td>
<td></td>
</tr>
<tr>
<td>i</td>
<td></td>
</tr>
<tr>
<td>j</td>
<td></td>
</tr>
<tr>
<td>k</td>
<td></td>
</tr>
<tr>
<td>l</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td></td>
</tr>
<tr>
<td>n</td>
<td></td>
</tr>
<tr>
<td>o</td>
<td></td>
</tr>
</tbody>
</table>

UP_CON_EDGE=f
DOWN_MET_TRAN_DOWN=n
LEFT_DEV_X=b
LEFT_MET_X=a
DOWN_DEV_Y=i
UP_DEV_Y=h
RIGHT_DEV_X=d
C.1.4 Adding a New Menu in the CIW

```skill
procedure( New_menu() ) ; THIS IS THE EXECUTING COMMAND
let( { ciwPointer
   ) ; CREATE THE DOCUMENTATION SUBMENU
EXECUTE = hiCreatePulldownMenu( 'MITEL15docMenuHandle
"READ_ME"
list(hiCreateMenuItem(}
procedure( INDIRECT_README()  
"SHOWS THE READ_ME FILE FOR THE INDIRECT MAPPING STYLE"  
view("/cmos4s/il/INDIRECT")  
)

procedure( DIRECT_README()  
"SHOWS READ_ME FILE FOR THE DIRECT MAPPING STYLE"  
view("/cmos4s/il/DIRECT")  
)

; end of procedure
C.1.5 A sample .cdsinit file for Mitel 1.5 micron Technology

include "-/cmos4s/il/menu_add"
include "-/cmos4s/il/opus_instance_mitel_novia"
include "-/cmos4s/il/opus_mitel_short_novia"
include "-/cmos4s/il/opus_mitel_test_full_novia"
New_menu()

C.2 Euler Path and the Metal Tracks

C.2.1 Euler path

;;;;;;; ONE SET OF PATHS: ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; FLATTENS A LIST: ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

procedure( flatten2(a)
    prog ()
    ;a='((1 2 3) (2 3 4) (4 5 6))
    for( i 1 (length(a)-1)
        p=append(car(a) cadr(a))
        a=cons(p cdr(cdr(a)))
        b=car(a)
        ;println(b)
        return(b)
    )
)

procedure( same(a)
    ( prog ()
    n='()
    while( a != '() )
        if( car(a) != cadr(a) then
            n=cons(car(a) n)
            a=cadr(a)
        else
            n=cons(car(a) n)
            a=cdr(a)
        )
        r=reverse(n)
        ;println(r)
    )
    return(r)
)

Appendix C The SKILL Codes 145
procedure( make(c) )

prog( ()
    k=0
    d=flatten2(c)
    ;println("d")
    ;println(d)
    while( d != '() )
        if( (length(car(d)) == 3 & mod(length(nthelem(car(car(d))) c)) 2) == 1 ) then
            k=k+1
            n=list(car(d))
        d=cdr(d)
        i=1
    while( i <= length(nthelem(car(last(car(n))) c)) )
        if( ! member(nthelem(i nthelem(car(last(car(n))) c)) n) &
            ! member(reverse(nthelem(i nthelem(car(last(car(n))) c))) n) &
            length(nthelem(i nthelem(car(last(car(n))) c))) == 3 ) then
            n=cons(nthelem(i nthelem(car(last(car(n))) c)) n)
            i=1
        else
            n
        i=i+1
    ) ;end if
) ;end while
p=reverse(n)
c=flatten2(p)
l=same(o)
if( length(l) >= 2*3
    ;println(l)
    return(l) ;1=return_list
) else
    d=cdr(d).
) ) )


if( k == 0 then
    d=flatten2(c)
while( d != '() )
    if( (length(car(d)) == 3 & mod(length(nthelem(car(car(d))) c)) 2) == 0 ) then
        n=list(car(d))
        d=cdr(d)
i=1
while; i <= length(nthelem(car(last(car(n)) c))
if( ( ! member(nthelem(i nthelem(car(last(car(n)))) c)) n) && 
    member(reverse(nthelem(i nthelem(car(last(car(n)))) c)) n) &&
    length(nthelem(i nthelem(car(last(car(n)))) c))) == 3) then

n = cons(nthelem(i nthelem(car(last(car(n)))) c)) n
i=1
else
n
i=i+1
); end if
); end while
p=reverse(n)
o=flatten2(p)
i=same(o)
return(l); l=return_list
;printn(l)
else
d=cdr(d)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
else
y=subst(p q y)
y=cons(q cdr(y))
)
)
m=cons(car(y) m)
y=cdr(y)
)m=reverse(m)
;println(m)
xx='()' yy='()' if(length(m) > 2 declare(o[length(m)-2]) )
init=0
while( length(m) > 2 xx='()' yy='()' first=nthelem(1 m)
second=nthelem(2 m)
xx=cons(first xx)
xx=cons('X xx)
xcons(second xx)
xx=cons(first yy)
xx=cons('X yy)
xx=cons(second yy)
o[init]=cons(reverse(yy) nthelem(first c))
o[init+1]=cons(xx nthelem(second c))
c=subst(o[init] nthelem(first c) c)
c=subst(o[init+1] nthelem(second c) c)
m=remove(second m)
m=remove(first m)
)

println("PASS GET_IT")
if( length(m) < 2 then
p=even_list1(c LENGTH)
else
lets_gol(c LENGTH)
)
)

procedure( even_list1(c LENGTH)

temp_list=copy(c)
;array_length(c)*(length(c)-1)*0.5 declare(even[10])
prog( ()
i=0
while( temp_list != '() temp_list_1=cdr(temp_list)
while( temp_list_1 != '())
cc=copy(c)
M='()
N='()

M=cons(car(car(car(temp_list)))) M)
M=cons('X M)
M=cons(car(car(car(temp_list_l)))) M)
M=reverse(M)

N=cons(car(car(car(temp_list_l)))) N)
N=cons('X N)
N=cons(car(car(car(temp_list)))) N)
N=reverse(N)

even1=cons(M car(temp_list))
even2=cons(N car(temp_list_l))

even_subst(even1 car(temp_list) cc)
even[i]=subst(even2 car(temp_list_l) even_)
i=i+1
temp_list_l= cdr(temp_list_l)
)
temp_list= cdr(temp_list)
)
for(i 0 9
letsgo1(even[i] LENGTH)
println("**************************")
)
)
)

procedure( to_last8(a)
prog( ()
k=0
q=tconc(nil '((X)))
;a='(((3 c 2) (3 f 4)) ((1 a 2) (1 b 2)) ((2 a 1) (2 b 1) (2 c 3)))
;a='((1 2 3) (3 5 6) (1 6 8))
for(i 2 length(a)
q=tconc(q nthelem(i a))
)
q=tconc(q car(a))
return(cdr(car(q)))
;println(cdr(car(q)))
)
)

procedure( rem1() 
c='(((1 a 2) (1 b 2)) ((2 a 1) (2 b 1) (2 a 3) (2 d 3) (2 f 4)) ((3 d 2) (3 c 2) (3 e 4)) ((4 e 3) (4 f 2)))
;x='((4 f 2 a 1 b 2 c 3 d 2)
declare(q[length(c)])
for( i 0 length(c)-1

q[i]="()
"
i=0
y='()
b=copy(c)
p=tconc(nil '((X X) (X X)))
println(p)
while( b != '() 
    e=car(b)
    while( e != '() 
        if( member('a car(e)) then 
            pp=remove('a car(e)) 
            q[i]=cons(pp q[i]) 
            ;println(q)
        else 
            m=car(e)
            q[i]=cons(m q[i])
        )
    e=cdr(e)
)
    b=cdr(b)
p=tconc(reverse(q[i]))
i=i+1
)
println(cdr(car(p)))
)

procedure( rem2() 
    c='(((1 a 2) (1 b 2)) ((2 a 1) (2 b 1) (2 a 3) (2 d 3) (2 f 4)) ((3 d 2) 
        (3 c 2) (3 e 4)) ((4 e 3) (4 f 2)))
    x='(2 a 1 b 2 c 3 d 2)
    declare(g[length(c)])
    for( i 0 length(c)-1 
        q[i]='()
    );
    ii=2
    i=0
    y='()
b=copy(c)
p=tconc(nil '((X X) (X X)))
    ;println(p)
    while( b != '() 
        e=car(b)
        while( e != '() 
            if( member(ntelem(2 car(e)) x) then 
                pp=remove(ntelem(2 car(e)) car(e)) 
                q[i]=cons(pp q[i]) 
            else 
                m=car(e)
                q[i]=cons(m q[i])
            )
        e=cdr(e)
) b=cdr(b)
p=tconc(p reverse(q[i]))
i=i+1
) println(cdr(car(p)))

;;;;;;;;;;;;;;;;;;;;;;FINDS ALL THE POSSIBLE PATHS;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

procedure( mult_comb(c)
prog( )
;c='((1 a 2) (1 b 2)) ((2 a 1) (2 b 1) (2 c 3)) ((3 c 2) (3 f 4)) ((4 f 2))
;c='((1 a 2) (1 b 2)) ((2 a 1) (2 b 1) (2 c 3) (2 d 3) (2 f 4)) ((3 d 2) (3 c 2) (3 e 4)) ((4 e 3) (4 f 2))
 z=copy(c)
BIG_LIST='()
temp_list='()
j=0
k=0
x=1
for(i 1 length(c)
x=x*length(nthelem(i c))
)

; x= # of combinations of the list
;println(flatten_(c))
;println(length(flatten_(c)))
declare(n[length(flatten_(c))])
varia_ble=1
while( c != '() 
p=car(c)
;println("p")
;println(p)
pp=length(p)

;;if( mod(varia_ble 11) == 0 then
;;if( mod(varia_ble 3) == 0 then
;pp=2
;else
;pp=1
;)

if( pp > 2 then
pp=2
else
pp=1
;)

Appendix C  The SKILL Codes 151
;if( mod(length(p) 2) == 0 then
;pp=length(p)/2
;else
;pp=(length(p)-1)/2+1
;
;if( pp >=2
;pp=2
;pp=1
;
)
declare(m[pp])
for(i 1 pp
x=cons(cdr(p) list(list(car(p))))
m[i-1]=flatten_(x)
n[j]=m[i-1]
temp_list=cons(n[j] temp_list)
p=m[i-1]
j=j+1
; j= # of each element of the list
)
;println("temp_list")
;println(temp_list)
BIG_LIST=cons(temp_list BIG_LIST)
k=k+1 ; k= # of element of c
c=cdr(c)
varia_ble=varia_ble+1
temp_list="()"
BIG_LIST=reverse(BIG_LIST)
;println("BIGLIST")
;println(BIG_LIST)
return(BIG_LIST)


;; THE FOLLOWING TWO SUBLISTS MAKE ALL THE POSSIBLE COMBINATIONS OF ALL
THE ELEMENTS IN A LIST;;;

procedure( comb3(x y)
prog( ()

y1=copy(y)
declare(SS[length(x)*length(y)])
i=0
LIST=tconc(nil '((X X) (X X)))
while( x != '() p=car(x) :
while( y != '() q=car(y)
SS[i]=cons(q list(p))
SS[i]=to_last(SS[i])
LIST=tconc(LIST SS[i])

i=i+1
y=car(y)
)
x=cdr(x)
y=y1
)
return(cdr(car(LIST)))
)
)

procedure( comb2(x y)
prog( ()
y1=copy(y)
declare(SS[length(x)*length(y)])
i=0
LIST=tconc(nil '(X X) (X X))
while( x != '() )
p=car(x)
while( y != '() )
q=car(y)
SS[i]=cons(q p)
SS[i]=to_last(SS[i])
LIST=tconc(LIST SS[i])
i=i+1
y=cdr(y)
)
x=cdr(x)
y=y1
)
return(cdr(car(LIST)))
)
)

procedure( to_last(a)
prog( ()
k=0
q=tconc(nil '((X))
for(i 2 length(a)
q=tconc(q nthelem(i a))
)
q=tconc(q car(a))
return(cdr(car(q)))
)
)

procedure( lets_gol(c LENGTH)
redundant_list='()
q=mult_comb(c)
LIST='()
Q=copy(q)
pp=comb3(car(q) cadr(q))
;println("comb3")
;println(pp)
LIST=pp
Q=cddr(Q)
while( Q != ')' 
pl=car(Q)
LIST=comb2(LIST pl)
println("passed one")
Q=cdr(Q)
) 
count=0
while( LIST != ')' 
list_for_make=car(LIST)
p=make(list_for_make)

if( length(p) >= LENGTH 
if( (! member(p redundant_list)) && cadr(p) != 'X' && (! member(reverse(p) redundant_list)) && cadr(reverse(p)) != 'X) then redundant_list=cons(p redundant_list)

;:::;
poport=outfile("~/cmos4s/il/store1" "w")
fprintf(poport "%L\n" p)

;:::;
---end testing---;

;println(p)
met_length1(p)
;track_(p)
)
)

if( mod(count 4) == 0 then
LIST=cddr(LIST)
else
LIST=cdr(LIST)
)
count=+count+1
)
;println("Redundant_list")
;println(redundant_list)
track_(redundant_list)
)

procedure( flatten_(a)
prog( ()
for(i 1 (length(a)-1)
p=append(car(a) cadr(a))
a=cons(p cdr(cadr(a)))
b=car(a)
return(b)
)
procedure( met_length1(a)
  total_length=0
  max_length_save=0

  while( a != '()
    met_list = copy(a)
    max_length=0

    if( member(car(a) cddr(met_list)) then
      while( met_list != '()

        if( car(a) == car(cddr(met_list)) then
          max_length=max_length+1
          total_length=total_length+max_length

        if( max_length_save <= max_length then
          max_length_save=max_length
        else
          max_length_save
        )

        a=cddr(a)
        met_list='()
      else
        max_length=max_length+1
        met_list=cddr(met_list)
      )
    )
  else
    a=cddr(a)
  )
)
println(max_length_save)
println(total_length))

C.2.1 The metal track(s)

procedure( track_(dd)
  ;prog( ()

  while( dd != '()
    a=car(dd)
    println(a)

  Appendix C  The SKILL Codes
r=0
templ='()
temp='()

ii=1

while( ii < length(a)+1
  temp=cons(nthelem(ii a) temp)
  ii=ii+2
)
temp=reverse(temp)
;println(temp)
b=copy(temp)

while( temp != '()
  if( member(car(temp) cdr(temp)) \&\& (! member(car(temp) templ)) then
    r=r+1
    templ=cons(car(temp) templ)
  )
temp=cdr(temp)
)

if( r > 0 then
  declare(discr[r])

  for( i 0 r-1
discr[i]='(')
)

k=0
repeat_list='()
while( b != '()
p=0
P=car(b)

if( member(P cdr(b)) \&\& (! member(P repeat_list)) then
discr[k]=cons(P discr[k])

  c=copy(b)
  while( c != '()
    if( p != '1' then
      if( (cadr(c) != P || (cadr(c) == P \&\& member(cadr(c) cddr(c)))) then
discr[k]=cons(cadr(c) discr[k])
    )
  )

  if( (cadr(c) == P \&\& (! member(cadr(c) cddr(c)))) then
discr[k]=cons(cadr(c) discr[k])
  k=k+1
  p=1
)
)
c=cdr(c)
)
)
repeat_list=cons(P repeat_list)
b=cdr(b)
)

LIST='()
for( i 0 r-1
LIST=cons(discr[i] LIST)
)
;println(LIST)
TEMP='()
while( LIST != ')
LIST1=copy(LIST)
while( LIST1 != ')

TEMP1='()
if(cadr(LIST1)
if( (! member(car(car(LIST)) cadr(LIST1))) then
TEMP1=cons(car(car(LIST)) TEMP1)
TEMPX=cons(car(cadr(LIST1)) TEMP1)
TEMP=cons(TEMPX TEMP)
)

LIST1=cdr(LIST1)
)
LIST=cdr(LIST)
)
;println(TEMP)
XX=all_track_list2(TEMP)
println(XX)
else
println("METALS NOT IN THE SAME TRACK")
); end if(r)
;
dd=cdr(dd)
)
):

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;LISTS ALL THE TRACKS;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

procedure( all_track_list2(TEMP)
prog( ()
;println(TEMP)

if( length(TEMP) != 0 then

declare( a[length(TEMP)])
a[0]=TEMP
q=TEMP
;println(TEMP)
for( i l length(TEMP)-1
  a[i]=flat_it(cdr(a[i-1]) list(list(car(a[i-1])))))
p=the_track_(a[i])
;println(p)

;if( length(p) <= length(q) && length(flat_it(p)) >= 
  length(flat_it(q)) then
  if( length(p) <= length(q) then
    q=p
  )
)

;poport-outfile("/-cmos4s/il/store2" "w")
;fprintf(poport "%L
" q)
;println(q)
return(q)
else
Q=0
return(Q)
;return(println("NO METAL TRACK"))
)
)

;;;;;;;;;;;;;;;;;;;;;;;;THE FINAL TRACK ORDER;;;;;;;;;;;;;;;;;;;;;;;;

procedure( the_track_(TEMP)
  prog( )
  flat_TEMP='()
  flat=flat_it(TEMP)
  track_list='()

  while( flat != '())
  if( ! member(car(flat) flat_TEMP))
    flat_TEMP=cons(car(flat) flat_TEMP)
  )
  flat=cdr(flat)
  Reserve=copy(flat_TEMP)
  while( flat_TEMP != '())
    N=copy(cdr(flat_TEMP))
    Y='()
    Y=cons(car(flat_TEMP) Y)
    XX='()
    reserve=copy(Reserve)

    while( N != '())
      kk=0
      for( i l length(Y)
        P='()
        P=cons(car(N) P)
  )
P = cons(nthelem(i Y) P)
P = reverse(P)
if( (! member(P TEMP)) && (! member(reverse(P) TEMP))
    kk = kk + 1
); end if
); end for

if( kk == 0
    Y = cons(car(N) Y)
); end if

N = cdr(N)
); end while (N)

track_list = cons(Y track_list)
track_list_temp = flat_it(track_list)

while( reserve != '()'
    if( (! member(car(reserve) track_list_temp))
        XX = cons(car(reserve) XX)
    ); end if
    reserve = cdr(reserve)
); end while

flat_TEMP = copy(XX)
); end while (reserve)

return(track_list)
)
;println(track_list)
)

C.2.2 One Dimensional Functional Cell Layout

/* THIS PROGRAM WILL GENERATE A LAYOUT FOR A GIVEN EULER PATH AND THE TRACK INFORMATION */

/* a IS THE EULER PATH AND b IS THE TRACK INFO */

/* N-transistors */

procedure( n_tran_main1()
    deOpen()
    cv = dbOpenCellView(deOpenForm->deLibName->value deOpenForm->deCellName->value deOpenForm->deViewName->value nil "w")
    ;if( hiFormCancel(deOpenForm) != t then
    first_layer = 1
vert_y=2.7
inity=1
k=1
same_track=copy(b)
println(same_track)
temp=copy(a)
pass_a=copy(a)
diff_track_a=copy(a)

while( b != '(')
tempx1=0
initx=-4.8
a=copy(temp)
ac=copy(a)
temp_list1='('
temp_list2='('
metal_temp='('
metal_list=copy(cddr(a))

while( a != '('
    if( car(cdr(a)) != 'Q' then
        initx1=7.2
    else
        initx1=11.3
    )

k=1

if( (member(car(a) metal_list) && car(a) != 'X' && member(car(a) car(b))) then
    while( metal_list != '()' )
        if( car(a) == car(metal_list) then
            inity=vert_y*5
            if(first_layer != 1 then
                if( k == 1 then
                    met_ln(initx inity initx1+initx)
                    met2_n(initx inity)
                    vial(initx inity)
                    met2_n(initx1+initx inity)
                    vial(initx1+initx inity)
                else
                    met_ln(tempx1 inity initx1+initx)
                    met2_n(tempx1 inity+2)
                    vial(tempx1 inity)
                    met2_n(initx1+initx inity+2)
                    vial(initx1+initx inity)
                )

        )

    )

if( first_layer == 1 then
    if( k == 1 then

met_ln(initx inity initx1+initx)
met_lnn_(initx inity+2)
met_lnn_(initx1+initx inity+2)
else
met_ln(tempxl inity initx1+initx)
met_lnn_(tempxl inity+2)
met_lnn_(initx1+initx inity+2)
)
)

k=k+1
 tempxl=initx1+initx
)

if(cadr(cdr(metal_list))
if( cadr(metal_list) != 'Q then
 initx1=initx1+7.2
 else
 initx1=initx1+11.3
 )
)

metal_temp=cons(car(metal_list) metal_temp)
metal_list=cdr(metal_list)
)
)

temp_list2=cons(car(a) temp_list2)
a=subst('X car(a) a)
a=cdr(a)

metal_list=copy(cdr(a))
metal_temp="()"

if( car(cdr(ac)) != 'Q then
 initx=initx+7.2
 else
 initx=initx+11.3
 )

temp_list1=cons(car(ac) temp_list1)
ac=cdr(ac)
)
vert_y=vert_y+1.13
first_layer=first_layer+1
b=cdr(b)
)
same_track=flatten2(same_track)
println(same_track)
a=copy(diff_track_a)
ac=copy(a)
temp_list1='()
temp_list2='()
metal_list=copy(cddr(a))
metal_temp='()
tempxl=0
initx=-4.8

while( a != '() 

if( car(cdr(a)) != 'Q then
initx1=7.2
else
initx1=11.3
)

k=1
if( (member(car(a) metal_list) && car(a) != 'X && (! member(car(a)
same_track))) then
while( metal_list != '() 

if( car(a) == car(metal_list) then
inity=vert_y*5

if(first_layer != 1 then
if( k == 1 then
met_ln(initx inity initx1+initx)
met2_n(initx inity)
vial(initx inity)
met2_n(initx1+initx inity)
vial(initx1+initx inity)
else
met_ln(tempxl inity initx1+initx)
met2_n(tempxl inity+2)
vial(tempxl inity)
met2_n(initx1+initx inity+2)
vial(initx1+initx inity)
)
)

if( first_layer == 1 then
if( k == 1 then
met_ln(initx inity initx1+initx)
met1nn_(initx inity+2)
met1nn_(initx1+initx inity+2)
else
met_ln(tempxl inity initx1+initx)
met1nn_(tempxl inity+2)
met1nn_(initx1+initx inity+2)
)
)
k=k+1
tempxl=initx1+initx
if(cadr(cdr(metal_list)))
if( cadr(metal_list) != 'Q then
initx1=initx1+7.2
else
initx1=initx1+11.3
)

metal_temp=cons(car(metal_list) metal_temp)
metal_list=cddr(metal_list)
)

if( (car(a) != 'X && member(car(a) cdr(a)) )
vert_y=vert_y+1.13
)

first_layer=first_layer+1

)

temp_list2=cons(car(a) temp_list2)
a=subst('X car(a) a)

a=cddr(a)

metal_list=copy(cddr(a))
metal_temp='()

if( car(cdr(ac)) != 'Q then
initx=initx+7.2
else
initx=initx+11.3
)

temp_list1=cons(car(ac) temp_list1)
ac=cddr(ac)
)

println(inity)
n_tran(pass_a inity+2)
)
;

procedure( n_tran(a outy)
k=0.
outx=-4.8
out_first=-4.8
out='1
out_list=subst(Z out a)
out_list_one=copy(out_list)
temp_out='()

while( out_list_one != '() 
if( (car(out_list_one) == 'Z' & & ( (! member(car(out_list_one) 
cdr(out_list_one))) & & (! member(car(out_list_one) temp_out)))) then 
k=1
out_real=out_first
out_met_ln(out_real outy+2.5)
else
out_first=out_first+7.2
)

temp_out=cons(car(out_list_one) temp_out)
out_list_one=cddr(out_list_one)
)

if( k != 1 then
while( out_list != '() 
if( member(Z cdr(out_list))
if( cadr(out_list) != 'Q then
outx=outx+7.2
else
outx=outx+11.3
)
)
out_list=cddr(out_list)
)
out(outx outy)
else
outx=out_real
)

main_list=copy(a)
temp_list='()
hor=0

while( main_list != '() 
if( (member(car(main_list) cdr(main_list)) | | member(car(main_list) 
temp_list))
if( (cadr(cdr(main_list)) & & (! member(cadr(cadr(main_list))
cdddr(main_list)))) & & 
( ! member(cadr(cdr(main_list)) temp_list))
if( cadr(main_list) != 'Q then
left_con_tran2_n(hor)
p=cadr(main_list)
dbCreateLabel(cv "text" hor:-2 get_pname(p) "centerLeft" "R0" "gothic" 3)
hor=hor+7.2
else
hor=hor+11.3
)
if( (member(car(main_list) cdr(main_list)) || member(car(main_list) temp_list))
  if( (cadr(cdr(main_list)) && (member(cadr(cdr(main_list))
cddr(main_list)) ||
        member(cadr(cdr(main_list)) temp_list))
    if( cadr(main_list) != 'Q then
two_con_tran2_n(hor)
p=cadr(main_list)
dbCreateLabel(cv "text" hor:-2 get_pname(p) "centerLeft" "R0" "gothic"
3)
  hor=hor+7.2
else
  hor=hor+11.3
)
)
)

if( ( (! member(car(main_list) cdr(main_list))) && (!
      member(car(main_list) temp_list)))
  if( (cadr(cdr(main_list)) && (! member(cadr(cdr(main_list))
cddr(main_list))) &&
      (! member(cadr(cdr(main_list)) temp_list))
    if( cadr(main_list) != 'Q then
      no_con_tran2_n(hor)
p=cadr(main_list)
dbCreateLabel(cv "text" hor:-2 get_pname(p) "centerLeft" "R0" "gothic"
3)
    hor=hor+7.2
  else
    hor=hor+11.3
  
)
)

if( ( (! member(car(main_list) cdr(main_list))) && (!
      member(car(main_list) temp_list)))
  if( (cadr(cdr(main_list)) && (member(cadr(cdr(main_list))
cddr(main_list)) ||
      member(cadr(cdr(main_list)) temp_list))
    if( cadr(main_list) != 'Q then
      right_con_tran2_n(hor)
p=cadr(main_list)
dbCreateLabel(cv "text" hor:-2 get_pname(p) "centerLeft" "R0" "gothic"
3)
    hor=hor+7.2
  else
    hor=hor+11.3
  )
)
temp_list=cons(car(main_list) temp_list)
main_list=caddr(main_list)
)
p_tran_main1(outy+12 outy outx hor)
)

/* TRANSISTOR WITH CONTACTS ON BOTH SIDES */

procedure( two_con_tran2_n(x)

y=0
dbCreateRect(cv "contact" list(x-2.1:y+3.7 x-3.6:y+5.2))
dbCreateRect(cv "contact" list(x+3.6:y+3.7 x+5.1:y+5.2))
dbCreateRect(cv "poly" list(x:y x+1.5:y+8.9))
dbCreateRect(cv "devwell" list(x-4.8:y+2.4 x+6.3:y+6.5))
dbCreateRect(cv "metal" list(x-4.8:y+2.4 x-0.9:y+6.5))
dbCreateRect(cv "metal" list(x+2.4:y+2.4 x+6.3:y+6.5))
dbCreateRect(cv "via" list(x+3.4:y+7.7 x+5.4:y+9.7))
dbCreateRect(cv "via" list(x-1.8:y+7.7 x-3.8:y+9.7))
dbCreateRect(cv "metal" list(x-0.6:y+6.5 x-5:y+10.9))
dbCreateRect(cv "dmet" list(x-0.6:y+6.5 x-5:y+10.9))
dbCreateRect(cv "metal" list(x+2.2:y+6.5 x+6.6:y+10.9))
dbCreateRect(cv "dmet" list(x+2.2:y+6.5 x+6.6:y+10.9))
input(x y)
dbSave(cv)
)

/* TRANSISTOR WITH CONTACTS ON LEFT SIDE */

procedure( left_con_tran2_n(x)

y=0
dbCreateRect(cv "contact" list(x-2.1:y+3.7 x-3.6:y+5.2))
dbCreateRect(cv "poly" list(x:y x+1.5:y+8.9))
dbCreateRect(cv "devwell" list(x-4.8:y+2.4 x+6.3:y+6.5))
dbCreateRect(cv "metal" list(x-4.8:y+2.4 x-0.9:y+6.5))
dbCreateRect(cv "via" list(x-1.8:y+7.7 x-3.8:y+9.7))
dbCreateRect(cv "metal" list(x-0.6:y+6.5 x-5:y+10.9))
dbCreateRect(cv "dmet" list(x-0.6:y+6.5 x-5:y+10.9))
dbSave(cv)
input(x y)
)

/* TRANSISTOR WITH CONTACTS ON RIGHT SIDES */

procedure( right_con_tran2_n(x)

y=0
dbCreateRect(cv "contact" list(x+3.6:y+3.7 x+5.1:y+5.2))
dbCreateRect(cv "poly" list(x:y x+1.5:y+8.9))
dbCreateRect(cv "devwell" list(x-4.8:y+2.4 x+6.3:y+6.5))
dbCreateRect(cv "metal" list(x+2.4:y+2.4 x+6.3:y+6.5))
dbCreateRect(cv "via" list(x+3.4:y+7.7 x+5.4:y+9.7))
dbCreateRect(cv "metal" list(x+2.2:y+6.5 x+6.6:y+10.9))

Appendix C The SKILL Codes 166
dbCreateRect(cv "dmet" list(x:2.2 y:6.5 x:6.6 y:10.9))
dbSave(cv)
input(x y)
)

/* TRANSISTOR NO CONTACTS */

procedure no_con_tran2_n(x)
y=0
dbCreateRect(cv "poly" list(x:y x:1.5 y:8.9))
dbCreateRect(cv "devwell" list(x:-4.8 y:2.4 x:6.3 y:6.5))
dbSave(cv)
input(x y)

)

procedure out(x y)
y=y+3
dbCreateRect(cv "dmet" list(x:7.7 x:3:y+1))
vial(x y:1.9)
dbSave(cv)
)

procedure met2_n(x y)
dbCreateRect(cv "dmet" list(x:y x:3:9.7))
dbSave(cv)
)

procedure out_met_ln(x y)
y=y+3
dbCreateRect(cv "dmet" list(x:6.5 x:3:y))
con(x y)
dbSave(cv)
)

procedure con(x y)
y=0
dbCreateRect(cv "metal" list(x:y+2.4 x:3.9 y:6.5))
dbCreateRect(cv "contact" list(x:1.2 y:3.7 x:2.7 y:5.2))
dbSave(cv)
vial(x y:7.9)
)

procedure met_lnn_(x y)
dbCreateRect(cv "metal" list(x:y x:3:9.7))
dbSave(cv)
)

procedure add_poly(x end_y)
dbCreateRect(cv "poly" list(x:8.7 x:1.5:end_y))
dbSave(cv)
)
procedure( via_out(x y)
  x=x+0.8
  y=y-0.8
  dbCreateRect(cv "metal" list(x-1:y-0.6 x+3.5:y+4))
  dbCreateRect(cv "via" list(x+.2:y+0.6 x+2.2:y+2.6))
  dbCreateRect(cv "dmet" list(x-1:y-0.6 x+3.5:y+4))
  dbSave(cv)
)

procedure( p_n_well(p, q, r, s)
  dbCreateRect(cv "nwell" list(p:q r+0.2:s))
  dbCreateRect(cv "pguard" list(p-3.5:q-3.5 r+3.7:s+3.5))
)

procedure( input(o p)
  o=o-1.2
  dbCreateRect(cv "poly" list(o:p o+3.9:p-3.9))
  dbCreateRect(cv "contact" list(o+1.2:p-1.2 o+2.7:p-2.7))
  dbCreateRect(cv "metal" list(o:p o+3.9:p-4.5))
  dbCreateRect(cv "metal" list(o-0.2:p-4.5 o+4.2:p-8.9))
  dbCreateRect(cv "dmet" list(o-0.2:p-4.5 o+4.2:p-8.9))
  dbCreateRect(cv "via" list(o+1:p-5.7 o+3:p-7.7))
)

procedure( vial(x y)
  x=x+0.8
  y=y-0.8
  dbCreateRect(cv "metal" list(x-1:y-0.6 x+3.5:y+4))
  dbCreateRect(cv "via" list(x+.2:y+0.6 x+2.2:y+2.6))
  dbCreateRect(cv "dmet" list(x-1:y-0.6 x+3.5:y+4))
  dbSave(cv)
)
procedure( met_ln(x y xl)
  dbCreateRect(cv "metal" list(x:y-0.3 xl+3:y+2.2))
  dbSave(cv)
)

procedure( met2_p(x y yl)
  yl=yl+7
  dbCreateRect(cv "dmet" list(x:y x+3:yl))
  dbSave(cv)
)

procedure( met_lnp(x y yl)
  dbCreateRect(cv "metal" list(x:y x+2.5:yl))
  dbSave(cv)
)

procedure( out_pl(x y yl)
  y=y+6.6
  dbCreateRect(cv "dmet" list(x:y x+3:yl))
  vial(x yl+1)
procedure(out_pl_met(x y y1)
y=y+6
y1=y1+1
dbCreateRect(cv "metal" list(x:y x+2.5:y1))
;vial(x y-1)
con_p(x y)
dbSave(cv)
)

procedure(con_p(x y)
y=y+6
dbCreateRect(cv "metal" list(x:y+2.4 x+3.9:y+6.5))
dbCreateRect(cv "contact" list(x+1.2:y+3.7 x+2.7:y+5.2))
dbSave(cv)
)

procedure(connect(x x1 y y1)
y=y+4
y1=y1+3
dbCreateRect(cv "metal" list(x:y x1:y1))
dbSave(cv)
)

procedure(power_rail(y dev_y)
yy=-3
dev_yy=dev_y-15.5
y1=y-2.7
dbCreateRect(cv "metal" list(-8.7:-5 -13.7:y))
dbCreateRect(cv "devwell" list(-8.7:-5 -13.7:dev_y))
dbCreateRect(cv "devwell" list(-8.7:dev_y+34.5 -13.7:y))
while(yy > dev_y+34.5)
dbCreateRect(cv "contact" list(-10.5:yy -12:yy+1.5))
yy=yy+5
while(yy < dev_y-2.7)
dbCreateRect(cv "contact" list(-10.5:yy -12:yy+1.5))
yy=yy+5
)

dbSave(cv)
)

procedure(ground_rail(x y dev_y)
yy=-3
dev_yy=dev_y-15.5
y1=y-2.7
dbCreateRect(cv "metal" list(x+3:-5 x+8:y))
dbCreateRect(cv "devwell" list(x+3:-5 x+8:dev_y))
dbCreateRect(cv "devwell" list(x+3:dev_y+34.5 x+8:y))
dbCreateRect(cv "p dope" list(x+1.4:-6.6 x+9.6:dev_y+1.6))
dbCreateRect(cv "ndope" list(x+1.4:-6.6 x+9.6:dev_y+1.6))

Appendix C

The SKILL Codes

169
dbCreateRect(cv "p dope" list(x+1.4:dev_y+32.9 x+9.6:y+1.7))
dbCreateRect(cv "n dope" list(x+1.4:dev_y+32.9 x+9.6:y+1.7))
while( yyl > dev_y+34.5
dbCreateRect(cv "contact" list(x+4.7:yy1 x+6.2:yy1+1.5))
yy1=yy1-5
)
while( yy < dev_y-3.5
dbCreateRect(cv "contact" list(x+4.7:yy x+6.2:yy+1.5))
yy=yy+5
)
dbSave(cv)
)
)

/* P-TRANSISTORS */

procedure( p_tran_mainl(add_y connect_y connect_x hor_x) increase_of_y=add_y first_layer=1 vert_y=2.7 tempx1=0 k=1 b='((2 4)) pass_p=copy(a) same_track=copy(b) println(same_track) temp=copy(a) diff_track_a=copy(a)

while( b !='()

tempx1=0
initx=-4.8
a=copy(temp)
ac=copy(a)
temp_list1=()
temp_list2=()
metal_temp=()
metal_list=copy(cddr(a))

while( a !='()

if( car(cdr(a)) != 'Q then
initx1=7.2
else
initx1=11.3
)

k=1
if( (member(car(a) metal_list) && car(a) != 'X' && member(car(a) car(b)))
then
while( metal_list != '()')

if( car(a) -- car(metal_list) then
inity=vert_y*5

if(first_layer != 1 then
if( k == 1 then
met2_ln(initx inity+add_y initx1+initx)
met2_p(initx inity+add_y+2 add_y)
vial(initx inity+add_y)
met2_p(initx1+initx inity+add_y+2 add_y)
vial(initx1+initx inity+add_y)
else
met2_ln(tempx1 inity+add_y initx1+initx)
met2_p(tempx1 inity+2+add_y add_y)
vial(tempx1 inity+add_y)
met2_p(initx1+initx inity+2+add_y add_y)
vial(initx1+initx inity+add_y)
)
)

if( first_layer == 1 then
if( k == 1 then
met2_ln(initx inity+add_y initx1+initx)
met2_p(initx inity+add_y+2 add_y+2.4)
met2_p(initx1+initx inity+add_y+2 add_y+2.4)
else
met2_ln(tempx1 inity+add_y initx1+initx)
met2_p(tempx1 inity+2+add_y add_y+2.4)
met2_p(initx1+initx inity+2+add_y add_y+2.4)
)
)

k=k+1

k=k+1

if(cadr(cdr(metal_list)))
if( cadr(metal_list) != 'Q then
inity1=inity1+7.2
else
inity1=inity1+11.3
)
)

metal_temp=cons(car(metal_list) metal_temp)
metal_list=cddr(metal_list)
temp_list2=cons(car(a) temp_list2)
a=subst('X car(a) a)
a=cdr(a)
metal_list=copy(cdr(a))
metal_temp='()

if( car(cdr(ac)) != 'Q then
initx=initx+7.2
else
initx=initx+11.3
)
temp_list1=cons(car(ac) temp_list1)
ac=cdr(ac)

vert_y=vert_y+1.13
first_layer=first_layer+1
b=cdr(b)
)
same_track=flatten2(same_track)
printn(same_track)
a=copy(diff_track_a)
ac=copy(a)
temp_list1='()
temp_list2='()
metal_list=copy(cdr(a))
metal_temp='()
tempx1=0
initx=-4.8

while( a != '()

if( car(cdr(a)) != 'Q then
initx1=7.2
else
initx1=11.3
)

k=1

if( (member(car(a) metal_list) && car(a) != 'X && (! member(car(a) same_track)) then
while( metal_list != '() 

if( car(a) == car(metal_list) then
inity=vert_y*5

if(first_layer != 1 then
if( k -- 1 then
met ln(initx inity+add_y initx1+initx)
met2 p(initx inity+add_y+2 add_y)
vial(initx inity+add_y)
met2 p(initx1+initx inity+add_y+2 add_y)
vial(initx1+initx inity+add_y)
else
met ln(tempx1 inity+add_y initx1+initx)
met2 p(tempx1 inity+2+add_y add_y)
vial(tempx1 inity+add_y)
met2 p(initx1+initx inity+2+add_y add_y)
vial(initx1+initx inity+add_y)
)
)

if( first_layer -- 1 then
if( k -- 1 then
met ln(initx inity+add_y initx1+initx)
met lp(initx inity+add_y+2 add_y+2.4)
met lp(initx1+initx inity+add_y+2 add_y+2.4)
else
met ln(tempx1 inity+add_y initx1+initx)
met lp(tempx1 inity+2+add_y add_y+2.4)
met lp(initx1+initx inity+2+add_y add_y+2.4)
)
)

k=k+1
tempx1=initx1+initx
)

if(cadr(cdr(metal_list)))
if( cadr(metal_list) != 'Q then
initx1=initx1+7.2
else
initx1=initx1+11.3
)
)

metal_temp=cons(car(metal_list) metal_temp)
metal_list=cdr(metal_list)
)
if( (car(a) != 'X && member(car(a) cdr(a)) ) )
vert_y=vert_y+1.13
)

first_layer=first_layer+1
)

temp_list2=cons(car(a) temp_list2)
a=subst('X car(a) a)
a=cddr(a)
metal_list=copy(cddr(a))
metal_temp='()

if( car(cdr(ac)) != 'Q then
initx=initx+7.2
else
initx=initx+11.3
)
temp_list1=cons(car(ac) temp_list1)
ac=cddr(ac)

power_gnd_y=inity+add_y+4
power_rail(power_gnd_y add_y)
p_tran(pass_p increase_of_y connect_y connect_x hor_n power_gnd_y)
)

procedure( p_tran(a add_yy connect_yy connect_xx hor_n power_gnd_y)
outx=-4.8
out_first=-4.8
temp_out='()
k=0
out='4
out_list=subst(Z out a)
out_list_one=copy(out_list)

while( out_list_one != '())
if( (car(out_list_one) == 'Z && (! member(car(out_list_one)
cdr(out_list_one)))) && (! member(car(out_list_one) temp_out))) then
k=1
out_real=out_first
out_pl_met(out_real add_yy add_yy-8.3)
println("SKS")
else
out_first=out_first+7.2
)
temp_out=cons(car(out_list_one) temp_out)
out_list_one=cddr(out_list_one)
)

if( k != 1 then
while( out_list != ')()
if( member(Z cdr(out_list)))
if( cdr(out_list) != 'Q then
outx=outx+7.2
else
outx=outx+11.3
)
)
out_list=cddr(out_list)
)

out_pl(outx add_yy add_yy-8)
connect(connect_xx outx connect_yy add_yy-8 )
else
if( out_real-connect_xx > 1
connect(connect_xx out_real connect_yy+.7 add_yy-7.8)
via_out(connect_xx add_yy-7.1)
)
)

main_list=copy(a)
temp_list='()
hor=0
while( main_list != '() )

if( (member(car(main_list) cdr(main_list)) || member(car(main_list)
temp_list))
if( (cadr(cdr(main_list)) && (! member(cadr(cdr(main_list))
cddr(main_list)))) &&
(! member(cadr(cdr(main_list)) temp_list)))
if( cadr(main_list) != 'Q then
left_con_tran2_p(hor add_yy)
add_poly(hor add_yy)
p=cadr(main_list)
dbCreateLabel(cv "text" hor:-2 get_pname(p) "centerLeft" "R0" "gothic"
3)
;hor=hor+3.5
hor=hor+7.2
else
;hor=hor+3.5
hor=hor+11.3
)
)
)

if( (member(car(main_list) cdr(main_list)) || member(car(main_list)
temp_list))
if( (cadr(cdr(main_list)) && (member(cadr(cdr(main_list))
cddr(main_list)) ||
member(cadr(cdr(main_list)); temp_list)))
if( cadr(main_list) != 'Q then
two_con_tran2_p(hor add_yy)
add_poly(hor add_yy)
p=cadr(main_list)
dbCreateLabel(cv "text" hor:-2 get_pname(p) "centerLeft" "R0" "gothic"
3)
hor=hor+7.2
else

hor=hor+11.3
)
)
)

if( (! member(car(main_list) cdr(main_list))) && (!
member(car(main_list) temp_list)))
if( (cadr(cdr(main_list))) && (! member(cadr(cdr(main_list))
 cddr(main_list))) &&
 (! member(cadr(cdr(main_list)) temp_list)))
if( cdadr(main_list) != 'Q then
no_con_tran2_p(hor add_yy)
add_poly(hor add_yy)
p=cadr(main_list)

dbcCreateLabel(cv "text" hor:-2 get_pname(p) "centerLeft" "R0" "gothic"
3);
;hor=hor+3.5
hor=hor+7.2
else
;hor=hor+3.5
hor=hor+11.3
)
)

if( (! member(car(main_list) cdr(main_list))) && (!
member(car(main_list) temp_list)))
if( (cadr(cdr(main_list))) && (member(cadr(cdr(main_list))
 cddr(main_list))) ||
 member(cadr(cdr(main_list)) temp_list)))
if( cdadr(main_list) != 'Q then
right_con_tran2_p(hor add_yy)
add_poly(hor add_yy)
p=cadr(main_list)

dbcCreateLabel(cv "text" hor:-2 get_pname(p) "centerLeft" "R0" "gothic"
3);
hor=hor+7.2
else
hor=hor+11.3
)
)
)

temp_list=cons(car(main_list) temp_list)
main_list=cddr(main_list)
)

if( hor_n > hor then
ground_rail(hor_n power_gnd_y)
else
ground_rail(hor power_gnd_y add_yy)
)
)
Shakil Siddiq was born on August 26, 1967 in Dhaka, Bangladesh. He completed his high school at Dhaka College in Dhaka. He graduated with a Bachelor of Engineering from Technical University of Nova Scotia (TUNS), Halifax in 1992. He completed his Master's of Applied Science in Electrical Engineering at the University of Windsor in November, 1994.