Multiple-Valued Logic Design
in Current-Mode CMOS

A Thesis
Submitted to the College of Graduate Studies and Research
in Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy
in the
Department of Electrical Engineering
University of Saskatchewan
Saskatoon, Saskatchewan

by

Atul K. Jain

Spring, 1994

The author claims copyright. Use shall not be made of the material contained herein without proper acknowledgement, as indicated on the copyright page.
to my grandparents

and parents
COPYRIGHT

The author has agreed that the Library, University of Saskatchewan, may make this thesis freely available for inspection. Moreover, the author has agreed that permission for extensive copying of this thesis for scholarly purpose may be granted by the professor or professors who supervised the thesis work recorded herein or, in their absence, by the Head of the Department or the Dean of the College in which the thesis work was done. It is understood that due recognition will be given to the author of this thesis and to the University of Saskatchewan in any use of the material in this thesis. Copying or publication or any other use of the thesis for financial gain without approval of the University of Saskatchewan and the author’s written permission is prohibited.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Electrical Engineering
University of Saskatchewan
Saskatoon, Saskatchewan, Canada
S7N 0W0
ABSTRACT

Over the last two decades, design using Multiple-Valued Logic (MVL) has been receiving considerable attention. With recent advancements in processing technologies it is now possible to implement higher-radix functions. MVL circuits have shown the potential of improving the use of chip area through increased functional density and reduced requirements for interconnection wiring. Currently, it is advantageous if MVL circuits are used along with binary circuits in the design of Very Large Scale Integration (VLSI) circuits. In order to minimize the fabrication process related overheads, it is desirable to design MVL circuits that can coexist on the same chip with binary logic circuits.

This thesis considers MVL design in current-mode CMOS compatible with existing binary logic CMOS fabrication processes. A given MVL function can be realized in the sum-of-product (SOP) form. The realization cost, in terms of the number of required product terms (PTs) and hence the circuit area, depends on the set of operators used. The choice of the set of operators is technology dependent. A new set of operators, consisting of literal, cycle, complement of literal, complement of cycle, min, and tsum operators is proposed in this thesis.

A general structure has been identified for realizing MVL circuits in current-mode CMOS. The structure consists of input, control, and output blocks. The input and the output blocks deal primarily with multiple-valued current signals. The control block signals are binary voltage signals providing flexibility by allowing the use of arbitrarily complex binary circuits. MVL circuits have been designed for the new set of operators using the general structure. The cost, in terms of number of transistors, of realizing the new set of operators and that of each of the existing sets of operators is comparable. The functionality of the designed circuits has been verified for 4-valued logic using HSPICE transient analysis simulations.
MVL function realizations using the new set of operators requires fewer PTs compared to realizations based on the existing sets of operators. A comprehensive comparison has been conducted for all 19683 3-valued 2-variable functions. The maximum number of PTs is reduced from 6, using the existing set of operators, to 3, using the new set of operators. The average number of PTs is decreased from 3.61 to 2.61. Sample 4-valued 2-variable function realizations using the new set of operators have also shown similar improvements.

In the past, several heuristic-based programs have been reported to obtain an efficient SOP expression for a given MVL function. HAMLET is one of these programs. It accepts user specified function expression and minimizes this expression. It includes implementation of many earlier reported heuristic-based algorithms. The Gold heuristic chooses the best realization after applying all other heuristics implemented in HAMLET. A new heuristic-based synthesis program has been developed to obtain a SOP expression for a given MVL function using the proposed set of operators. For a random sample of 4-valued 2-variable functions, the realizations have been compared for the developed program and the existing program, HAMLET (Gold). It is observed that for 69% of the functions, the developed program performs better and for 4% of the functions the HAMLET (Gold) realizations are better. The average number of PTs required by the developed program and HAMLET (Gold) are 5.53 and 6.62, respectively.

In order to further reduce the number of PTs required for MVL function realizations two new approaches have also been identified. The new approaches are based on difference-of-sum-of-products (DOSOP) and sum-of-terms (SOT) realization of MVL functions. It has been shown that the required number of terms can be reduced in the realization of some MVL functions, using the new approaches, as compared to those obtained using the SOP realization.
ACKNOWLEDGEMENTS

The author expresses his appreciation and gratitude to Dr. R. J. Bolton and Dr. M. H. Abd-El-Barr for their supervision of this work. Their advice and assistance in preparation of this thesis is thankfully acknowledged. The author acknowledges the valuable comments and criticisms provided by the members of the advisory committee, Dr. R. Gander, Dr. J. Greer, Prof. A. Krause, and Dr. H. Wood. It was an honour to have Dr. Z. Vranesic from the University of Toronto as the external examiner. His comments are highly appreciated. The author is also thankful to Dr. C. E. Soteros of Department of Mathematics at University of Saskatchewan for many valuable discussions.

The author wishes to express his deepest gratitude to his wife, Neera, and son, Pranay, for their constant encouragement and support. Special thanks are extended to his parents, all other relatives, and friends for their support in making this project a reality.

The author is also thankful to Mr. A. Jalnapurkar and Mr. I. MacPhedran for their help in the preparation of this thesis. Thanks are also due to all the colleagues at Kelsey Institute for their support and special thanks go to Mr. M. Strankay for his encouragement and understanding.

Financial assistance provided by the University of Saskatchewan in form of Graduate Scholarship and by the NSERC is thankfully acknowledged. Finally, the author would like to express his gratitude to Canadian Microelectronics Corporation for providing facilities to undertake this work.
Table of Contents

COPYRIGHT .......................................................... i
ABSTRACT ............................................................ ii
ACKNOWLEDGEMENTS ................................................ iv
Table of Contents .................................................... v
List of Figures ........................................................ viii
List of Tables ........................................................ xiii
List of Abbreviations ................................................ xv

1 Introduction ......................................................... 1
  1.1 Multiple-Valued Logic .......................................... 1
  1.1.1 Design Considerations ...................................... 3
  1.1.2 Full versus Partial Multiple-Valued Logic ................ 7
  1.2 Research Motivation ........................................... 11
  1.3 Research Objectives ........................................... 15
  1.4 Thesis Outline ................................................ 15

2 Notation and Definitions ........................................... 17
  2.1 Operators ....................................................... 17
  2.2 Function Realization ........................................... 20
  2.3 Function Synthesis ............................................. 23

3 Basic Circuit Elements ............................................. 29
  3.1 Current-mode CMOS logic ....................................... 29
  3.2 Sum ............................................................. 32
  3.3 Constant ......................................................... 32
  3.4 Current Mirrors ............................................... 35
  3.5 Threshold ....................................................... 36
  3.6 Switch .......................................................... 42
  3.7 Summary ......................................................... 46
8 Conclusions and Future Work

8.1 Conclusions .................................................. 137
8.2 Future Directions ............................................ 140

References ......................................................... 142

Appendix A Typical Process Parameters for CMOS3DLM ....... 148
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.1</td>
<td>Map representation of example 4-valued 2-variable function for which the truth table is given in Table 1.1.</td>
<td>3</td>
</tr>
<tr>
<td>Figure 1.2</td>
<td>Block diagram of a full adder.</td>
<td>4</td>
</tr>
<tr>
<td>Figure 1.3</td>
<td>A binary full adder realization using threshold element [10].</td>
<td>5</td>
</tr>
<tr>
<td>Figure 1.4</td>
<td>A quaternary full adder realization using threshold element [10].</td>
<td>5</td>
</tr>
<tr>
<td>Figure 1.5</td>
<td>Two-valued bus structure [9].</td>
<td>8</td>
</tr>
<tr>
<td>Figure 1.6</td>
<td>Multiple-valued bus structure [9].</td>
<td>8</td>
</tr>
<tr>
<td>Figure 1.7</td>
<td>Prime implicants of example 4-valued 2-variable function shown in Figure 1.1.</td>
<td>14</td>
</tr>
<tr>
<td>Figure 1.8</td>
<td>Minimal SOP form realization of example 4-valued 2-variable function shown in Figure 1.1.</td>
<td>15</td>
</tr>
<tr>
<td>Figure 2.1</td>
<td>Map representation for some example product terms for 4-valued 2-variable functions.</td>
<td>22</td>
</tr>
<tr>
<td>Figure 2.2</td>
<td>An example 4-valued 2-variable MVL function.</td>
<td>24</td>
</tr>
<tr>
<td>Figure 2.3</td>
<td>For example function shown in Figure 2.2, candidate product term for minterm (3[1' x_1]) (3[1' x_2]).</td>
<td>27</td>
</tr>
<tr>
<td>Figure 2.4</td>
<td>For example function shown in Figure 2.2, some level 2 CPTs for minterm (3[1' x_1]) (3[1' x_2]).</td>
<td>28</td>
</tr>
<tr>
<td>Figure 2.5</td>
<td>For example function shown in Figure 2.2, some level 3 CPTs for minterm (3[1' x_1]) (3[1' x_2]).</td>
<td>28</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>CMCL binary inverter circuit realization.</td>
<td>30</td>
</tr>
<tr>
<td>Figure 3.2</td>
<td>Sum circuit element. (a) Circuit realization (b) Symbol.</td>
<td>33</td>
</tr>
<tr>
<td>Figure 3.3</td>
<td>N-type constant circuit element. (a) Circuit realization (b) Symbol.</td>
<td>33</td>
</tr>
<tr>
<td>Figure 3.4</td>
<td>P-type constant circuit element. (a) Circuit realization (b) Symbol.</td>
<td>34</td>
</tr>
<tr>
<td>Figure 3.5</td>
<td>Circuit for realizing (N_{\text{ref}}) and (P_{\text{ref}}) voltages.</td>
<td>35</td>
</tr>
<tr>
<td>Figure 3.6</td>
<td>N-type current mirror circuit element. (a) Circuit realization (b) Symbol.</td>
<td>35</td>
</tr>
</tbody>
</table>
Figure 3.7 P-type current mirror circuit element. (a) Circuit realization (b) Symbol. .......................... 36
Figure 3.8 N-type threshold circuit element. (a) Circuit realization (b) Symbol. .......................... 37
Figure 3.9 P-type threshold circuit element. (a) Circuit realization (b) Symbol. .......................... 38
Figure 3.10 Circuit used to study the input current and the output voltage relationship of a threshold circuit element. ......................... 39
Figure 3.11 DC analysis simulation results for the circuit shown in Figure 3.10. .......................... 40
Figure 3.12 Transient analysis simulation results for the circuit shown in Figure 3.10. ......................... 41
Figure 3.13 Switch circuit element. (a) N-type circuit realization. (b) P-type circuit realization. (c) Symbol. .......................... 43
Figure 3.14 Circuit for evaluating performance of switches (a) N-type switch (b) P-type switch. .......................... 43
Figure 3.15 Performance of n-type switch using a p-type constant. ................. 44
Figure 3.16 Performance of p-type switch using a p-type constant. ................. 45

Figure 4.1 Block diagram of the structure used for realizing MVL operators. 49
Figure 4.2 Typical circuit realizations of the input and the output blocks. 50
Figure 4.3 Circuits realizing min operator. (a) Input Currents are sinking. (b) Input currents are sourcing [66]. .......................... 51
Figure 4.4 Circuit realization of tsum using the min circuit. ......................... 51
Figure 4.5 Circuit realization of tsum using the structure shown in Figure 4.1. .......................... 52
Figure 4.6 Circuit for realizing a literal, $k[\overline{x}]^y$, operator. .......................... 53
Figure 4.7 Circuit for realizing a complement of literal, $k[\overline{\overline{x}}]^y$, operator. .......................... 55
Figure 4.8 Circuit for realizing a cycle operator, $x^m$. .......................... 57
Figure 4.9 Realization of a level restorer circuit. .......................... 59
Figure 4.10 Another realization of a level restorer circuit. .......................... 61
Figure 4.11 HSPICE output for the transient analysis of the circuit realization of min$(x_1, x_2)$. .......................... 63
Figure 4.12 Expansion of HSPICE output for the transient analysis of the circuit realization of $\min(x_1, x_2)$ for time 900ns to 1100ns... 64

Figure 4.13 Expansion of HSPICE output for the transient analysis of the circuit realization of $\min(x_1, x_2)$ for time 1.4 $\mu$s to 1.6 $\mu$s... 65

Figure 4.14 Expansion of HSPICE output for the transient analysis of the circuit realization of $\min(x_1, x_2)$ for time 2.9 $\mu$s to 3.9 $\mu$s... 66

Figure 4.15 HSPICE output for the transient analysis of the circuit realizations of $tsum(x_1, x_2)$... 68

Figure 4.16 Expansion of HSPICE output for the transient analysis of the circuit realization of $tsum(x_1, x_2)$ for time 1.9 $\mu$s to 2.3 $\mu$s... 69

Figure 4.17 HSPICE output for the transient analysis of the circuit realization of $3[^1x_2]$ and $2[^1x_1]$. ... 70

Figure 4.18 HSPICE output for the transient analysis of the circuit realization of cycle, $x_2^2$, operator... 72

Figure 4.19 Maximum propagation delay variation for $3[^1x_1]$ circuit... 74

Figure 5.1 An example MVL function to show use of different combining operators in SOP form realization... 77

Figure 5.2 Product terms for realization of the function, $f_1(x_1, x_2)$, in SOP form using max operator... 79

Figure 5.3 Product terms for realization of the function, $f_1(x_1, x_2)$, in SOP form using tsum operator... 79

Figure 5.4 Product terms for realization of the function, $f_1(x_1, x_2)$, in SOP form using modsum operator... 80

Figure 5.5 An example 4-valued 2-variable MVL function... 81

Figure 5.6 Map of PT required for realizing the function $f_2(x_1, x_2)$ using Set 1... 82

Figure 5.7 Circuit realization of function, $f_2(x_1, x_2)$, using Set 1... 82

Figure 5.8 PTs required for realizing the function $f_2(x_1, x_2)$ using Set 2... 83

Figure 5.9 Circuit realization of function, $f_2(x_1, x_2)$, using Set 2... 84

Figure 5.10 PTs required for realizing the function $f_2(x_1, x_2)$ using Set 3... 85

Figure 5.11 Circuit realization of function, $f_2(x_1, x_2)$, using Set 3... 86

Figure 5.12 HSPICE output for the transient analysis of the circuit realization of $f_2(x_1, x_2)$ using Set 1... 87
Figure 5.13 HSPICE output for the transient analysis of the circuit realization of $f_2(x_1, x_2)$ using Set 1 and a *level restorer* circuit.

Figure 5.14 An example 4-valued 2-variable MVL function, which can be realized as the *complement* function using less number of PTs.

Figure 5.15 Pseudo-code of the program for finding the number of PTs required to realize $r$-valued $n$-variable functions for a given set of operators.

Figure 6.1 An example MVL function to illustrate realization based on the DOSOP form.

Figure 6.2 Partition of the example function, shown in Figure 6.1, for realization in the DOSOP form.

Figure 6.3 Structure $S_1$ for MVL function realization in the DOSOP form.

Figure 6.4 Structure $S_2$ for MVL function realization in the DOSOP form.

Figure 6.5 An example MVL function to illustrate realization based on the DOSOP form using different structures for implementation.

Figure 6.6 PTs for the SOP form realization of example function shown in Figure 6.5.

Figure 6.7 PTs for realization based on the DOSOP form using structure $S_1$, for example function shown in Figure 6.5.

Figure 6.8 PTs for realization based on the DOSOP form using structure $S_2$, for example function shown in Figure 6.5.

Figure 6.9 An example 5-valued 2-variable function for illustrating the DOSOP form realization.

Figure 6.10 Circuit realization of the function shown in Figure 6.1 using the DOSOP form and structure $S_1$.

Figure 6.11 HSPICE transient analysis simulation results of the circuit realization of the MVL function shown in Figure 6.10.

Figure 6.12 An example MVL function to illustrate realization based on the SOT form.

Figure 6.13 PTs for the SOP form realization of example function shown in Figure 6.12.

Figure 6.14 Partition of the example function, shown in Figure 6.12, for realization in the SOT form.

Figure 6.15 For $n = 2$ the circuit realization of a *logic term* (a) the *input* block (b) the *output* block.
Figure 6.16 For $n = 2$ and $r = 4$, the map representation and the control logic block circuit realization for the logic terms possible for $a_1 = 1$, $b_1 = 2$, $a_2 = 1$, and $b_2 = 2$.

Figure 6.16 (cont'd.)

Figure 6.17 Circuit realization for the example function (shown in Figure 6.12) using the general structure (shown in Figure 4.1).

Figure 6.18 HSPICE transient analysis simulation results of the circuit realization of the MVL function shown in Figure 6.17.

Figure 7.1 Minterm labels for 4-valued 2-variable functions.

Figure 7.2 An example 4-valued 2-variable function to illustrate the steps of the algorithm.

Figure 7.3 An example 4-valued 2-variable function to illustrate the use of Set 2 operators with the new algorithm.

Figure 7.4 An example function for which the HAMLET (Gold) requires fewer PTs as compared to the algorithm when level bound is 1.

Figure 7.5 An example function for which the HAMLET (Gold) requires fewer PTs as compared to the algorithm.

Figure 7.6 Average number of PTs required for the developed program using Set 1 and Set 2 operators, and HAMLET (Gold) for different categories of functions.
List of Tables

Table 1.1  Truth table for an example 4-valued 2-variable function.  
Table 1.2  Binary full adder specifications.   
Table 1.3  Quaternary full adder specifications.   
Table 3.1  Truth table for binary inverter   
Table 3.2  Truth table for a 4-valued inverter   
Table 4.1  Realization of literals and size of transistors $T_1$, $T_2$, and $T_3$ in 4-valued logic.   
Table 4.2  Size of transistors $T_1$, $T_2$, and $T_3$ for the realization of cycle operator in 4-valued logic.   
Table 4.3  Average and typical propagation delays for example circuits.   
Table 4.4  Different cases for simulation based on variation in threshold voltage for n-type and p-type transistors.   
Table 4.5  Limits for maximum propagation delay for process parameter variations.   
Table 5.1  Number of PDs in each category for 3-valued 2-variable functions.  
Table 5.2  Number of PDs for different set of operators in 3-valued 2-variable functions.   
Table 5.3  Distribution of number of PDs and number of functions, for 3-valued 2-variable functions, using three set of operators when the complement of function is not allowed.   
Table 5.4  Distribution of number of PDs and number of functions, for 3-valued 2-variable functions, using three set of operators when the complement of function is allowed.   
Table 5.5  Average number of PDs, for 3-valued 2-variable function, using three set of operators when the complement of function is not allowed and when it is allowed.   
Table 7.1  For 4-valued 2-variable functions, the distribution of number of functions along with the percentage of total number of functions based on number of non-zero minterms.   

xiii
Table 7.2  Comparison of synthesis results, for 4-valued 2-variable functions, using the developed program (Set 1 operators) and the HAMLET (GOLD) (Set 3 operators) for 5500 randomly generated function. ....................................... 133

Table 7.3  Comparison of synthesis results, for 4-valued 2-variable functions, SOP form realization based on Set 1 and Set 2 operators using the developed program for 5500 randomly generated function. ................................................. 134
List of Abbreviations

BiCMOS  Bipolar Complementary Metal Oxide Semiconductor
CAD    Computer-Aided Design
CCD    Charge-Coupled Devices
CMC    Canadian Microelectronics Corporation
CMCL   Current-Mode CMOS Logic
CMOS   Complementary Metal Oxide Semiconductor
CPT    Candidate Product Term
DOSOP  Difference-of-Sum-of-Products
HAMLET Heuristic Analyzer for Multiple-valued Logic Expression
Translation
IC     Integrated Circuit
MOS    Metal Oxide Semiconductor
MVL    Multiple-Valued Logic
PD     Prime Decomposition
PE     Processing Element
PT     Product Term
SOP    Sum-of-Products
SOT    Sum-of-Terms
VLSI   Very Large Scale Integration
VMCL   Voltage-Mode CMOS Logic
1. Introduction

1.1 Multiple-Valued Logic

Today's digital computing world is dominated by two-valued (binary) logic. The status of binary logic design has reached its present level of complexity largely due to the availability of efficient two-state devices and computer-aided design (CAD) tools. With increasing complexity of present day binary systems, interconnection is the most important problem (both on-chip and between chips). It is widely accepted that routing of interconnections on a chip is a difficult problem. Researchers have reported that the silicon area used for interconnections is about 70% of the area in a very large scale integration (VLSI) chip [1]. Similarly, difficulties of bringing an increasing number of connections off-chip is promoting a new consideration to packaging concepts.

In the past, many solutions have been proposed to overcome the interconnection problem (both on-chip and between chips). In all the solutions an attempt has been made to increase the information content of each connection and/or chip pin either using time multiplexing or level multiplexing. In time multiplexing, a pin of an integrated circuit (IC) is used to connect different signals at different times whereas level multiplexing allows an interconnection to carry an r-valued signal ($r > 2$).

With microprocessor dual-in-line packages, time multiplexing is the solution most often used when pin limitations exist. This is effective in alleviating the pin-out problem for some chip architectures, like Intel microprocessors 8085, 8086. However, it leads to an overall speed performance degradation. Furthermore, multiplexing of chip input and output signals also require more off-chip and on-chip interfacing circuitry.
Another solution for pin-out problem comes as a result of advancements in packaging techniques whereby specific packages, with more than 200 pins (for example, a regular pin-grid-array package), have been developed to solve the pin-limitation problem [2]. Although these new packaging techniques solve the pin-out problem at the chip level, they cause an interconnection problem at the board level. Thus, in general, neither time-multiplexing nor new packaging techniques provide an absolute solution to the pin-out problem.

The use of circuits with signals having more than two-values, i.e., multiple-valued signals, has been offered as a viable solution to the interconnection problem [3, 4, 5, 6]. Circuits that allow signals to have more than two logic levels are called multiple-valued logic (MVL) circuits. Similarly, functions for which input and output are specified using more than two logic values are called MVL functions. Consider the truth table of an example 4-valued 2-variable function shown in Table 1.1. Each of the input variables (\(x_1\) and \(x_2\)) and the output \((f(x_1, x_2))\) can have 4 logic values.

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_1)</td>
<td>(x_2)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
When $x_1$ is 0 and $x_2$ is 0 then $f(x_1, x_2)$ is 1. Similarly, when $x_1$ is 1 and $x_2$ is 3 then $f(x_1, x_2)$ is 2. The truth table shown in Table 1.1 can also be represented in a map form, as shown in Figure 1.1. It should be noted that zero values for $f(x_1, x_2)$ are not shown in Figure 1.1 for clarity.

![Figure 1.1 Map representation of example 4-valued 2-variable function for which the truth table is given in Table 1.1.](image)

### 1.1.1 Design Considerations

Over the last two decades, designs using MVL have been receiving considerable attention [4, 5, 6, 7, 8]. There are two basic reasons for using MVL circuits in VLSI [9]:

1. The possibility of making better use of the chip area through increased functional density.
2. The reduced requirement for interconnection wiring, including the smaller number of pins needed on IC packages.

Consider the realization of a full adder (shown in Figure 1.2) using binary and multiple-valued signals. A full adder circuit calculates the arithmetic sum of all the inputs and generate the sum, $S$, and the output carry, $C_{out}$. For binary logic realization $x_1, x_2, C_{in}, S, C_{out} \in \{0, 1\}$ whereas for MVL $x_1, x_2, S \in \{0, 1, \ldots, r - 1\}$ and $C_{in}, C_{out} \in \{0, 1\}$. Adder specifications for binary and MVL ($r = 4$) full adders are given in Table 1.2 and Table 1.3, respectively.
For different radices, Stemerdink [10] has studied circuit realizations for full-adders based on threshold elements. He has reported that for realizations of circuits using threshold logic, the choice of the radix may have a minimum impact on the complexity of the circuit realization. The threshold based realizations for binary and quaternary logic full-adders are shown in Figure 1.3 and Figure 1.4, respectively. In both realizations, E is equal to the sum of all inputs, i.e., \( x_1 + x_2 + C_{in} \). This sum is used to generate \( C_{out} \) using a threshold element. For binary logic, if \( E \) is greater than 1.5 then \( C_{out} \) is equal to 1 else \( C_{out} \) is equal to 0. Similarly, for quaternary logic, if \( E \) is greater than 3.5 then \( C_{out} \) is equal to 1 otherwise \( C_{out} \) is equal to 0. The output \( S \) is obtained by subtracting 2\( C_{out} \) from \( E \) for binary logic and subtracting 4\( C_{out} \) from \( E \) for quaternary logic.

It should be noted that quaternary logic realization is equivalent to a 2-bit binary full adder realization, where 2-bit inputs are encoded as quaternary inputs. In addition, the chip area required for binary and quaternary logic realization using

\[
\begin{array}{c|c|c}
\hline
x_1, x_2, C_{in}, S, C_{out} \in \{0, 1\} \\
\hline
\text{Input} & x_1 + x_2 + C_{in} & S & C_{out} \\
\hline
0 & 0 & 0 & 0 \\
1 & 1 & 0 & \\
2 & 0 & 1 & \\
3 & 1 & 1 \\
\hline
\end{array}
\]

**Figure 1.2** Block diagram of a full adder.
Table 1.3 Quaternary full adder specifications.

<table>
<thead>
<tr>
<th>$x_1, x_2, S \in {0,1,\ldots,r-1}$ and $C_{in}, C_{out} \in {0,1}$</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1 + x_2 + C_{in}$</td>
<td>$S$</td>
<td>$C_{out}$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

$E = x_1 + x_2 + C_{in}$  
$S = E - 2C_{out}$  
$C_{out}$

Figure 1.3 A binary full adder realization using threshold element [10].

$E = x_1 + x_2 + C_{in}$  
$S = E - 4C_{out}$  
$C_{out}$

Figure 1.4 A quaternary full adder realization using threshold element [10].
threshold logic is almost the same. Thus, the use of quaternary logic provides more functional density as compared to binary logic. This result can be extended to higher-valued logic. Furthermore, for a given number of interconnections, it is possible to transmit more information using MVL signals as compared to binary logic. For example, a 4-valued signal on a single wire carries twice the information that can be carried using a binary signal.

Despite the above mentioned advantages, MVL circuits have failed to penetrate the binary dominated digital world. According to Vranesic [9], the significant reason for slow progress in implementing MVL circuits in VLSI is the lack of an established market for such components. Other reasons include tighter tolerances and reduced noise immunity offered by MVL circuits. The availability of a wide variety of binary devices also makes it difficult to convince a logic designer to explore MVL alternatives. However, with advancements in processing technologies many MVL circuits are available for logic designers [2]. Recently, Kameyama et al. [11] have reported a $32 \times 32$ bit signed digit (SD) multiplier implementation using MVL circuits realized in current-mode CMOS. They have reported that the chip area and the power dissipation of the MVL multiplier circuit is reduced to half of that of the fastest conventional binary realization of the same multiplier. The propagation delays for both the implementations were reported to be comparable.

It is advantageous to use MVL circuits in VLSI design, when the following two requirements are fulfilled [6, 9].

1. Ease of interfacing MVL sub-systems with binary sub-systems.

2. An advanced technology which allows economical implementation of MVL circuits.

Before proceeding with the discussion on the interfacing requirement, it is necessary to understand what radix is to be used for MVL circuits. The choice of radix is influenced by the technology used to realize the circuits. It has been shown by researchers [5, 12, 13] that theoretically, $r$ should be equal to $e = 2.718$. However,
in practice the value of \( r \) must be an integer, this suggests that \( r = 3 \) (i.e., ternary) should be the obvious choice. In the past, CMOS circuits have been used to realize ternary valued logic circuits where two source voltages (Vdd and Vss) along with ground have been used to represent three MVL values \((1,0,-1)\) [14]. These circuits are useful for realizing arithmetic operations using signed arithmetic. Ternary logic circuits can be interfaced with binary logic circuits by using two binary bits to represent a ternary MVL value. Using this scheme, one combination of two binary bits is not assigned to any logic value. Thus, to be able to utilize the complete available functional density and interface MVL circuits with binary, it is necessary that radix, \( r \), be a power of 2. One would desire to have higher values of \( r \) to be able to encode more information in MVL as compared to binary. However, the acceptable tolerance values put a limit on the value of \( r \). The most practical choice, with currently available technologies, is \( r = 4 \). This allows the designer to explore complete available functional density.

1.1.2 Full versus Partial Multiple-Valued Logic

Some researchers [2, 6, 9] are of the opinion that MVL cannot survive on its own, and has to coexist with binary logic whereas some others [14, 15] support the view that future chips will use only MVL signals. For co-existence of binary and MVL circuits, it is essential to allow communications between binary and MVL elements. Vranesic [9] has proposed two structures to handle MVL and Binary circuits interconnection:

1. Use binary signalling on the bus (see Figure 1.5).

2. Use multivalued signalling on the bus (see Figure 1.6).

For both structures, it is essential to have conversion circuitry, i.e., encoder and decoder circuits. Conversion circuits are required to allow MVL elements to communicate with binary counterparts and vice versa. The conversion circuits may either be a part of the MVL chip (see dashed rectangle in Figure 1.5), or on separate chips. In order to minimize the usage of conversion circuitry, for binary dominated systems
Figure 1.5 Two-valued bus structure [9].

Figure 1.6 Multiple-valued bus structure [9].
the binary bus structure will be more useful as compared to MVL bus structure and
vice versa. However, the MVL bus structure has poor noise immunity for large values
of r.

For mixed-radix design (i.e., binary and MVL circuits on the same chip or in
the same system) to be cost effective it is necessary to satisfy the following two
criteria [16, 17]:

1. The encoders and decoders must be as small as possible to maximize chip saving.

2. The MVL circuits should be implementable by the same fabrication process
   used for binary logic circuits, thus minimizing process-related overhead.

In the past, several designs for 4-valued encoder/decoder circuits have been
proposed using different technologies. In 1984, Freitas and Current [18] reported
implementation of quaternary encoding and decoding CMOS circuits, where MVL
logic levels are represented as current levels. They used SPICE simulation to prove the
functionality of these circuits. In 1987, Current et al. [19] reported similar results for
the above circuits after fabrication. Similar encoder/decoder circuits have been used
by Butler and Kerkhoff [20] in charge-coupled devices (CCD) technology for realizing
circuits, where MVL signals are represented as amount of charge. In 1985, Abd-El-
Barr and Vranesic [21, 22] reported a different approach for the realization of 4-valued
encoder/decoder circuits in CMOS. In their implementation, they have combined
amplitude encoding with time division as two attributes to represent MVL values.
For defining four logic values, they have used two non-zero voltage levels and two
pulse durations. In 1987, Singh [16] proposed circuit realizations for encoder/decoder
circuits using voltage signals to represent MVL logic levels. In 1990, Bhattacharya [17]
proposed encoder/decoder circuit realizations using weak buffer in clocked CMOS
circuits. These circuits require fewer gates as compared to the circuits proposed by
Singh [16].

The above discussion indicates that it is possible to design mixed-radix circuits.
But, one has to compromise on speed, power dissipation, and area depending on the
technology used. However, sometimes designers does not have any other alternative other than to use mixed-radix design. Recently, Etiemble et al. [23] reported a transmission system of a massively parallel computer which consists of one million processing elements (PEs). From one PE to another PE, it is required to transmit 10 bits, and transmission is allowed in both directions. For binary signalling, it would require 20 wires for transmission between 2 PEs. However, there are 6 neighbours for each PE, thus from each PE there would be 120 wires using binary signalling. Instead of just using binary signalling they opted to integrate binary and MVL parts. They have used 4-valued voltage signals to provide interconnection between two processing elements (PE). They have used BiCMOS (Bipolar CMOS) circuitry to generate MVL signals. The BiCMOS realization of an encoder circuit is approximately half the size of CMOS realization and improves the propagation delay to 3.8 ns compared to 13.2 ns, if CMOS is used. Thus, in order to make mixed-radix or partial MVL circuits a reality, it is necessary to be able to select the right technology for implementation.

Partial MVL logic suffers from certain drawbacks. Firstly, encoder/decoder circuitry cost extra chip area and secondly, the circuit performance in terms of speed is degraded because of the necessity to do the conversion.

Full MVL circuits attempt to utilize the advantages of increased functional density provided by MVL and the reduced requirements for interconnection wiring, to its full extent. Furthermore, these circuits do not need any radix conversion circuitry on-chip. Heung and Mouftah [14, 15] reported implementations using 3-valued logic with depletion and enhancement type CMOS transistors. In 1987, Ueno et al. [24] reported designs using CMOS quaternary threshold logic circuits. In these circuits, MVL logic levels are realized by modifying the fabrication process to allow multiple ion implants. In addition, Abd-El-Barr et al. [25] and Dueck and Miller [26] have used a totally MVL approach for synthesis of MVL circuits using CCD technology.

Until now, it has not been possible to determine which of the two approaches, i.e., full and partial MVL circuit realization, is better than the other. Currently, circuits are being designed using both approaches. It is expected that the partial
MVL approach will continue to be used most often because of the availability of binary components and because most logic designers are more conversant with binary logic. However, eventually, the full MVL approach may evolve when complete MVL logic families and necessary CAD tools are available commercially, as is the case for binary logic. This will depend upon the development of new technologies for MVL design (which are acceptable to logic designers) and the success in dealing with the problem of tighter noise margins for high-radix MVL circuits.

1.2 Research Motivation

Despite the potential for reducing the interconnection problem and increasing the functional density, MVL circuits are still in their infancy. This is because of their complexity and the lack of well established design techniques, as are available for binary logic. The design of VLSI circuits can be divided into two phases. Phase I deals with the design of circuits in terms of basic gates or simple building blocks. In Phase II, geometric layouts are generated that are suitable for fabrication in order to realize the actual circuit. Logic minimization is an integral part of Phase I as it allows the designer to reduce the chip area required for circuit implementation. It is also necessary to have CAD tools that allow reduction of design time.

In the case of binary logic, the advancements have resulted in state-of-the-art CAD tools that automate as many of the design tasks as possible while allowing the designer the ability of controlling the synthesis process. A series of programs that translate an algorithm (specified in a high level language) into a chip layout (geometries suitable for fabrication) are now available [27, 28, 29]. These tools allow designers to specify circuit constraints that must be satisfied by the design.

In the case of MVL circuits design, some CAD tools have been reported for circuit design in CCD and current-mode CMOS. In 1988, Onneweer et al. [30] reported the first CAD tool for MVL circuits. Single variable MVL functions were synthesized and geometric layouts are automatically generated using current-mode CMOS technology. In 1989, Kerkhoff and Butler [31] reported another CAD tool
that is suitable for CCD programmable logic array (PLA) circuits. The input to this tool is an algebraic expression of the function to be realized. This expression is minimized and the geometric layout is generated as the output. In 1990, Yurchak and Butler [32] reported HAMLET (Heuristic Analyzer for Multiple-valued Logic Expression Translation), a CAD tool for MVL function realization. HAMLET accepts user specified function expression and minimizes this expression in terms of number of product terms required to realize the function. This minimization is suitable for both current-mode CMOS and CCD technologies. However, using HAMLET, it is possible to generate geometric layout for the circuit realization of the minimized MVL function in current-mode CMOS.

This thesis concentrates on Phase I of the design of MVL functions using current-mode CMOS circuits. It is known that for MVL circuit design, the selection of basic building blocks (also called set of operators) affects the cost of circuit realization in terms of chip area. This selection is technology dependent. In the past, many sets of operators have been proposed using different technologies [33, 34, 35, 36, 37, 38]. Using these operators, different logic minimization and synthesis tools have been reported [25, 32, 33, 35, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49]. Two approaches have been used for MVL function realization, namely using a cost-table approach [39, 40, 41, 42, 43, 44, 45], or a direct-cover approach using sum-of-products (SOP) form realization [32, 33, 35, 38, 46, 47, 48, 49].

In a cost-table approach, a given MVL function is realized as the sum of sub-functions which are stored together with their respective cost in a tabular form. The cost of realization of a given MVL function is the sum of the costs of sub-functions plus the cost of combining them. Usually, the number of transistors required for the realization is referred to as the cost of realization. It is difficult to design optimum cost-tables for a given technology. This represents a major disadvantage to the cost-table approach.

A direct cover approach using SOP form realization consists of two steps: select a minterm and select an implicant. These realizations are structured and are similar
to binary PLA realizations. Using a PLA, a given binary logic function is realized as a sum of product terms (PTs). A PLA consists of two major subsections or planes: an AND plane and an OR plane. The AND plane is used to realize PTs and the OR plane combines the generated PTs in the AND plane. An individual PT is realized as a single column in the AND plane, thus the number of PTs is referred to as the cost of realization. For classical binary realizations of PLAs, minimization is achieved by generating all the prime implicants followed by the selection of a minimum number of prime implicants which cover the function [50, 51, 52]. Only AND, OR, and NOT (inverter) gates are necessary to realize these circuits.

The problem of minimizing a sum-of-products realization of a given MVL function is much more complex. There are two reasons for this. Firstly, depending on the set of operators used, there are a number of ways of realizing a PT and the sum operation (i.e., combining PTs) [26, 37, 53, 54, 55]. The choice of operators is technology dependent. It is known that in current-mode CMOS realizations, the use of the truncated sum (tsum) operator for combining PTs is on average more efficient, as compared to the use of max or modsum operators. Secondly, it has been shown by researchers [53, 54, 55] that for tsum based realization of MVL functions, it is necessary to consider all the implicants and not just the prime implicants. Consider the function shown in Figure 1.1. There are 11 prime implicants which are shown in Figure 1.7. These implicants are prime implicants because they are not covered by any other implicant of the function. Using tsum operator for combining PTs, there is no combination of these prime implicants that yields the function \( f(x_1, x_2) \) (shown in Figure 1.1). The minimum realization consists of 7 PTs as shown in Figure 1.8. Out of these, 4 PTs are prime implicants and 3 PTs are not prime implicants. It should be noted that inclusion of one or more prime implicants is not a required condition for obtaining minimal realization. Tirumalai and Butler [55] have shown some examples where the minimal sum-of-product form of realization does not consist of any prime implicant.
The number of implicants for a MVL function could be prohibitively large. Consider a simple 4-valued 2-variable function having a value of 2 for all possible combinations of the input variables, the total number of implicants is 200 and there is only one prime implicant. The minimal cover consists of this single prime implicant. Finding an absolute minimal solution using brute force techniques is undesirable because it is computationally very expensive. For example, the function shown in Figure 1.1 has 47 implicants and requires 7 PTs in a minimal cover. In order to obtain a minimal cover, all combinations of six or fewer implicants have to be examined. There are 12,467,220 such combinations; searching all these is prohibitively time consuming. Therefore, heuristic techniques generating near minimal solutions are acceptable. Currently, all minimization tools reported for sum-of-products representation of MVL functions follow heuristics to achieve near minimal covers [32, 38, 47, 48, 49]. In 1991, Tirumalai and Butler [56] conducted an analysis of existing algorithms. Their analysis showed that it is not possible to classify one of these as the best because none always achieves the best cover. The proposed work is to investigate these algorithms along with existing sets of operators with an objective that further improvements could be achieved.

**Figure 1.7** Prime implicants of example 4-valued 2-variable function shown in Figure 1.1.
1.3 Research Objectives

The objectives of the research work reported in this thesis are the following:

1. To analyse the existing sets of operators and identify new set(s) of operators that can be realized using existing binary logic CMOS fabrication processes.

2. To develop a heuristic-based technique for the sum-of-product realization of MVL functions using the new set(s) of operators, with an objective that further improvements could be achieved as compared to the realizations based on the existing set of operators.

3. To investigate new techniques for MVL function realization that allow further reduction in the chip area required.

1.4 Thesis Outline

This thesis consists of eight chapters. Background material and definitions of terms used in this thesis are given in Chapter 2. Chapter 3 deals with discussion on current-mode CMOS technology. Circuit realization and analysis of basic circuit elements of current-mode CMOS are also considered in Chapter 3. Existing sets of operators have been studied with the objective of identifying a new set of operators for efficient SOP realization of MVL functions. The new set of operators along with their circuit realizations in current-mode CMOS are discussed in Chapter 4. In Chapter 5, SOP realization of MVL functions based on the new set of operators is considered. For MVL functions, the circuit realization based on the new set of operators is compared
to that obtained using the existing sets of operators in Chapter 5. In order to further improve the realization of MVL functions, two new approaches for the realization of MVL functions have been identified. These are discussed in Chapter 6 along with example function realizations for each approach. Chapter 7 deals with synthesis of MVL functions based on the new set of operators using SOP form. The algorithm used for the synthesis program is discussed in Chapter 7, along with some illustrative example function realizations. In addition, for a sample of 5500 randomly generated functions, the realizations based on the new synthesis program have been compared with that obtained using HAMLET [32] program. Thesis conclusions and future research directions are given in Chapter 8.
2. Notation and Definitions

In this chapter, definition of some important terms used in this thesis and background material for MVL are given.

This chapter consists of three sections. In Section 2.1, definitions related to the proposed and the existing sets of operators are given. Definitions of different terms used for MVL function realizations are given in Section 2.2. Terms related to MVL function synthesis are explained in Section 2.3.

2.1 Operators

Definition 2.1:

Consider an r-valued n-variable function \( f(X) \), where \( X = \{x_1, x_2, ..., x_n\} \) and \( x_i \) takes on values from \( R = \{0, 1, 2, ..., r - 1\} \). The function \( f(X) \) is a mapping

\[
 f : R^n \rightarrow R.
\]  

(2.1)

There are \( r^{(r^n)} \) different possible functions. For example, if \( r = 3 \) and \( n = 2 \), then there are \( 3^{(3^2)} \), i.e., 19683, possible functions.

Definition 2.2:

A complement of a logic level, \( l \) is defined as [57]:

\[
 \bar{l} = (r - 1) - l.
\]  

(2.2)

For 4-valued logic, if \( l = 0, \bar{l} = 3 \), and if \( l = 2, \bar{l} = 1 \).
Definition 2.3:

A min (minimum) operator is defined as [33]:

\[
\min(a_1, a_2, \ldots, a_n) = a_1 \cdot a_2 \cdot \cdots \cdot a_n
\] (2.3)

where \( a_i \in R \). For \( a_1 = 5, a_2 = 2, \) and \( a_3 = 3, \) \( \min(a_1, a_2, a_3) = \min(5, 2, 3) = 2. \)

Definition 2.4:

A max (maximum) operator is defined as [33]:

\[
\max(a_1, a_2, \ldots, a_n) = a_1 \odot a_2 \odot \cdots \odot a_n
\] (2.4)

where \( a_i \in R \). Considering \( r = 4 \) and \( n = 3 \), if \( a_1 = 2, a_2 = 3, \) and \( a_3 = 1 \) then \( \max(2, 3, 1) = 3. \)

Definition 2.5:

A tsum (truncated sum) operator is defined as [37, 38]:

\[
tsum(a_1, a_2, \ldots, a_n) = a_1 \boxplus a_2 \boxplus \cdots \boxplus a_n = \min(a_1 + a_2 + \cdots + a_n, r - 1)
\] (2.5)

where \( a_i \in R \). Considering \( r = 4 \) and \( n = 3 \), if \( a_1 = 2, a_2 = 3, \) and \( a_3 = 1 \) then \( tsum(2, 3, 1) = \min(2 + 3 + 1, 4 - 1) = \min(6, 3) = 3. \)

Definition 2.6:

A modsum (modulus sum) operator is defined as [26]:

\[
\text{modsum}(a_1, a_2, \ldots, a_n) = a_1 \oplus a_2 \oplus \cdots \oplus a_n
\]

\[
= (a_1 + a_2 + \cdots + a_n) \mod r
\] (2.6)

where \( a_i \in R \). Considering \( r = 4 \) and \( n = 3 \), if \( a_1 = 2, a_2 = 3, \) and \( a_3 = 1 \) then \( \text{modsum}(2, 3, 1) = (2 + 3 + 1) \mod 4 = 2. \)
Definition 2.7:

A *window literal* of an MVL variable, \( x \) is defined as [33]:

\[
^{a \leq x \leq b} (r - 1) \quad \text{if } a \leq x \leq b \\
0 \quad \text{otherwise,}
\]

(2.7)

where \( a, b \in R \) and \( a \leq b \). Considering \( r = 4 \), logical values of \( x, ^1x^3, ^1x^2, ^0x^0, \) and ^0x^3 are shown below:

<table>
<thead>
<tr>
<th>( x )</th>
<th>^1x^3</th>
<th>^1x^2</th>
<th>^0x^0</th>
<th>^0x^3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

The information in the above table can be summarized using the following compact notation \( x =<0123>, ^1x^3 =<0333>, ^1x^2 =<0330>, ^0x^0 =<3000>, \) and ^0x^3 =<3333>.

Definition 2.8:

A *literal* of an MVL variable, \( x \) is defined as [62]:

\[
k[^{a \leq x \leq b} x] = \begin{cases} 
  k & \text{if } ^{a \leq x \leq b} x = \text{binary high} \\
  0 & \text{otherwise,}
\end{cases}
\]

(2.8)

where

\[
^{a \leq x \leq b} x = \begin{cases} 
  \text{binary high} & \text{if } a \leq x \leq b \\
  \text{binary low} & \text{otherwise,}
\end{cases}
\]

(2.9)

\( a, b \in R, a \leq b, \) and \( k \in \{1, 2, \cdots, r - 1\} \); \( k \) is called the value of the literal.
A complement of literal is defined as:

\[
 k[\overline{a}(x)]^b = \begin{cases} 
 k & \text{if } (\overline{a}(x)^b = \text{binary high}) \\
 0 & \text{otherwise.} 
\end{cases} 
\]  

(2.10)

where

\[
 \overline{a}(x)^b = \begin{cases} 
 \text{binary low} & \text{if } a \leq x \leq b \\
 \text{binary high} & \text{otherwise,} 
\end{cases} 
\]  

(2.11)

Considering \( r = 4 \), for \( x =<0123> \), \( 3[\overline{1}(x)^3] = <0330> \), \( 2[\overline{3}(x)^2] = <0002> \), \( 3[\overline{1}(x)^3] = <3000> \), and \( 1[\overline{3}(x)^1] = <1001> \).

**Definition 2.9:**

A clockwise cycle operator is defined as \[35\]:

\[
x^m = (x + m) \mod r
\]  

(2.12)

where \( m \in R \).

A counter-clockwise cycle operator is defined as:

\[
x^m = (x - m) \mod r
\]  

\[
= x^{-m} = (x - m + r) \mod r.
\]  

(2.13)

Considering \( r = 4 \), for \( x =<0123> \), \( x^0 = <0123> \), \( x^1 = <1230> \), \( x^2 = <2301> \), \( x^3 = <3012> \), \( x^4 = <3012> \), \( x^5 = <2301> \), and \( x^6 = <1230> \).

2.2 Function Realization

**Definition 2.10:**

A product term (PT) is defined as a min operation of a set of literals, a set of complement of literals, and a set of cycle operators. In a PT each variable \( x_i \) appears
at most once. A PT, \( P(x_1, x_2, \ldots, x_n) \) is expressed as:

\[
P(x_1, x_2, \ldots, x_n) = X_1 \cdot X_2 \cdot \ldots \cdot X_n = \min(X_1, X_2, \ldots, X_n)
\] (2.14)

where \( X_i \in \{k_i[x_i^{\text{hi}}, x_i^{\text{lo}}], x_i^{\text{hi}}, x_i^{\text{lo}}\} \), \( a_i, b_i, d_i \in \mathbb{R} \) and \( k_i \in \{1, 2, \ldots, r - 1\} \) are the values for the \( i \)th variable. Some example PTs for 4-valued 2-variable functions are:

\[
1[x_1^{\text{hi}}, 1][x_2^{\text{lo}}],
1[x_1^{\text{hi}}, 2][x_2^{\text{lo}}],
1[x_1^{\text{hi}}, 1][x_2^{\text{hi}}],
1[x_1^{\text{hi}}, 2][x_2^{\text{hi}}],
2[x_1^{\text{hi}}, 2][x_2^{\text{lo}}],
2[x_1^{\text{hi}}, 2][x_2^{\text{hi}}],
2[x_1^{\text{hi}}, 1][x_2^{\text{lo}}],
2[x_1^{\text{hi}}, 1][x_2^{\text{hi}}],
x_1.
\]

Map representation for each of these PTs is shown in Figure 2.1. Please notice that zero values are omitted from the map representation for clarity. It should also be noted that there are many different ways of representing a PT. In the above examples, the following PTs are the same:

\[
(1[x_1^{\text{hi}}, 1]) \cdot (1[x_2^{\text{lo}}]) = (1[x_1^{\text{hi}}, 1]) \cdot (2[x_2^{\text{lo}}]) = (2[x_1^{\text{hi}}, 1]) \cdot (1[x_2^{\text{lo}}]),
\]

the PT, \( (2[x_1^{\text{hi}}, 1]) \) is independent of variable \( x_2 \), and the PT, \( (x_1) \) is independent of variable \( x_1 \).

Earlier realizations of MVL functions in CMOS that have been reported in the literature [53, 32], have used window literal \((a \cdot x^b)\) and \( \min(\cdot) \) operators to realize a PT. Using these operators, a product term, \( P'(x_1, x_2, \ldots, x_n) \) is expressed as:

\[
P'(x_1, x_2, \ldots, x_n) = c \cdot a_1 x_1^{b_1} \cdot a_2 x_2^{b_2} \cdot \ldots \cdot a_n x_n^{b_n} = \min(c, a_1 x_1^{b_1}, a_2 x_2^{b_2}, \ldots, a_n x_n^{b_n}),
\] (2.15)

where \( a_i x_i^{b_i} = (r - 1)[x_i^{\text{hi}}, x_i^{\text{lo}}], a_i, b_i \in \mathbb{R}, a_i \leq b_i \), and \( c \in \{1, 2, \ldots, r - 1\} \). It should be noted that \( c \) is the value of the \( P' \). One way of representing \( P' \) in terms of the product term definition adopted in this thesis is to absorb the constant \( c \) in all the
Figure 2.1 Map representation for some example product terms for 4-valued 2-variable functions.

\[ P'(x_1, x_2, \cdots, x_n) = (X_1) \cdot (X_2) \cdot \cdots \cdot (X_n) \]  \hfill (2.16)

where \( X_i = c[a_{i} x_i x_i] \). Considering \( r = 4 \) and \( n = 2 \), \( 1 \cdot 1 x_1^2 \cdot 1 x_2^3 \) can be rewritten as \( (1[1(x_1)]^2) \cdot (1[1(x_2)]^3) \). Thus, the PT realization based on literal and min operators, corresponds to the operators (i.e., window literal and min) used in the earlier realizations.
Definition 2.11:

A logic term, $T$ is defined as:

$$T = k[Y] = \begin{cases} k & \text{if } (Y = \text{binary high}) \\ 0 & \text{otherwise,} \end{cases} \quad (2.17)$$

where $Y = L_1 \langle \text{op} \rangle L_2 \langle \text{op} \rangle \cdots \langle \text{op} \rangle L_m$, $\langle \text{op} \rangle$ is a binary operator, $L_i \in \{^a_i x_i, \overline{^a_i x_i} \}$, $a_i, b_i \in R$, $a_i \leq b_i$, and $k \in \{1, 2, \cdots, r - 1\}$ determines the value of the logic term. In a logic term each variable $x_i$ appears at most once. There are three forms for a logic term depending on which of the three binary operators $\text{AND}$ ($\cap$), $\text{OR}$ ($\cup$), or $\text{XOR}$ ($\oplus$) is substituted for $\langle \text{op} \rangle$, i.e., $T \in \{P, Q, R\}$, where

$$P = k[L_1 \cap L_2 \cap \cdots \cap L_m];$$
$$Q = k[L_1 \cup L_2 \cup \cdots \cup L_m];$$
$$R = k[L_1 \oplus L_2 \oplus \cdots \oplus L_m].$$

Some 4-valued 2-variable terms are: $1[\overline{^1 x_1} \cap \overline{^2 x_2}];$ $2[\overline{^1 x_1} \cup \overline{^2 x_2}];$ and $2[\overline{^1 x_1} \oplus \overline{^2 x_2}].$

A literal of an MVL variable, $x$ is a special case of logic term, where $Y$ depends only on single variable. So, there are two types of literals: $k[^a x_i]$ and $k[\overline{^a x_i}].$

2.3 Function Synthesis

Definition 2.12:

For a MVL function, $f(x_1, x_2, \cdots, x_n)$ an assignment of values to variables $x_1 = a_1, x_2 = a_2, \cdots$, and $x_n = a_n$ is called a minterm, if

$$f(a_1, a_2, \cdots, a_n) \neq 0, \quad (2.18)$$

where $a_i \in \{0, 1, \cdots, r - 1\}$ for ith variable.
A minterm is a special case of product term consisting of literal and min operators, where the PT is dependent on all the variables and \( a_1 = b_1, a_2 = b_2, \ldots, \) and \( a_n = b_n. \) The value of PT is referred to as the value of the minterm. If the value of a minterm is equal to \( r, \) then it is called as a don’t care and is represented as \( d. \) In general, there can be a maximum of \( r^n \) number of minterms (i.e., function having non-zero value for all possible combinations of the input variables). For 4-valued 2-variable functions, this number is \( 4^2 = 16. \) Consider the example shown in Figure 2.2, some of the minterms are \((1[x_1]) \cdot (1[x_2]), (2[0,x_1]) \cdot (2[1,x_2]),\) and \((3[x_1]) \cdot (3[x_2]).\) The function consists of 10 minterms. Out of these, for 5 minterms the function value is 1, for 2 minterms the function value is 2, for 2 minterms function value is 3, and for 1 minterm function value is \( d \) (i.e., don’t care). In Figure 2.2, blank spaces in the map represent input combinations for which function value is 0.

**Definition 2.13:**

Two minterms are adjacent if they differ in one input variable \( x_i \) and for all smaller values of \( j \) and \( k \) satisfy the following

\[
f(a_1, a_2, \ldots, (a_i - j) \mod r, \ldots, a_n) \neq 0 \tag{2.19}
\]

and

\[
f(a_1, a_2, \ldots, (a_i + k) \mod r, \ldots, a_n) \neq 0, \tag{2.20}
\]
where for odd value of $r$, $(0 < j \leq \frac{r-1}{2}$ and $0 < k \leq \frac{r-1}{2}$) and for even value of $r$, $(0 < j \leq \frac{r}{2}$ and $0 < k \leq \frac{r}{2}$). In Figure 2.2, minterms $(1[0{x_1}]) \cdot (2[1{x_2}])$, $(3[1{x_1}]) \cdot (3[1{x_2}])$, and $(1[2{x_1}]) \cdot (1[1{x_2}])$ are adjacent to minterm $(1[3{x_1}]) \cdot (1[3{x_2}])$. However, minterm $(1[2{x_1}]) \cdot (1[1{x_2}])$ is not adjacent to $(1[3{x_1}]) \cdot (1[3{x_2}])$ because Equation 2.19 is not satisfied for $j = 1$ even though it is satisfied for $j = 2$.

**Definition 2.14:**

For a minterm, number of adjacencies for a variable $x_i$ is equal to

$$na_i = j + k,$$

(2.21)

where $j$ and $k$ are the maximum values that satisfy Equation 2.19 and Equation 2.20, respectively. There can be a maximum of $r - 1$ adjacencies for a variable. In Figure 2.2, minterm $(3[1{x_1}]) \cdot (3[1{x_2}])$ has 3 adjacencies ($j = 2$, $k = 1$) for variable $x_1$, and 2 adjacencies ($j = 2$, $k = 0$) for variable $x_2$.

**Definition 2.15:**

A minterm having value $m$ is covered by a PT, if $m$ is a don’t care or the value of PT, $k$ for that minterm is equal to $m$, where $m \in \{1, 2, \ldots, r - 1\}$. In Figure 2.2, PT $(2[0{x_1}]) \cdot (2[0{x_2}])$ covers minterms $(2[0{x_1}]) \cdot (2[0{x_2}])$, $(2[0{x_1}]) \cdot (2[1{x_2}])$, and $(4[1{x_1}]) \cdot (4[0{x_2}])$.

**Definition 2.16:**

A product term (PT1 consisting of a set of minterms $M_1$) is covered by another product term (PT2 consisting of a set of minterms $M_2$), if $M_1$ is a subset of $M_2$ and the values of all the minterms in $M_1$ are less than or equal to the corresponding minterms in $M_2$. In Figure 2.2, $(x_1^2) \cdot (3[1{x_2}])$ is a PT which covers other PTs. Some of the covered PTs are: $(2[1{x_2}]) \cdot (2[1{x_2}])$, $(1[2{x_1}]) \cdot (1[1{x_2}])$, and $(3[1{x_1}]) \cdot (3[1{x_2}])$. Similarly, PT $(1[1{x_2}]) \cdot (1[1{x_2}])$ covers $(1[1{x_2}]) \cdot (1[1{x_2}])$. 


**Definition 2.17:**

An **implicant** of a function \( f(x_1, x_2, \cdots, x_n) \) is a product term \( I(x_1, x_2, \cdots, x_n) \), such that

\[
f(x_1, x_2, \cdots, x_n) \geq I(x_1, x_2, \cdots, x_n)
\]  

(2.22)

for all assignments of \( x_i \)'s. In Figure 2.2, \((2[0(x_1')]) \cdot (2[1(x_2')]), (1[0(x_1')]) \cdot (1[1(x_2')]), (1[0(x_1')]) \cdot (1[1(x_2')]),\) and \((1[1(x_2')])\) are some of the implicants.

**Definition 2.18:**

A **prime implicant** of a function \( f(x_1, x_2, \cdots, x_n) \) is an implicant \( I(x_1, x_2, \cdots, x_n) \), such that no other implicant \( I'(x_1, x_2, \cdots, x_n) \) satisfies

\[
I'(x_1, x_2, \cdots, x_n) \geq I(x_1, x_2, \cdots, x_n)
\]  

(2.23)

for all assignments of \( x_i \)'s. In the above example, \((2[0(x_1')]) \cdot (2[1(x_2')]), (1[0(x_1')]) \cdot (1[1(x_2')]),\) and \((1[1(x_2')])\) are all prime implicants.

**Definition 2.19:**

If a PT consists of only **literals** and complement of **literals** having a value of \( r \) (i.e., don't care), then **candidate product term** (CPT) for a minterm is defined as the largest PT which covers all minterms adjacent to it. For example function shown in Figure 2.2, CPT for minterm \((3[1(x_1')]) \cdot (3[1(x_2')])\) is shown in Figure 2.3. It should be noted that all possible PTs that cover minterm \((3[1(x_1')]) \cdot (3[1(x_2')])\) in Figure 2.2 are covered by the above CPT.

**Definition 2.20:**

**Size of candidate product term** is defined as

\[
\prod_{i=1}^{n} (na_i + 1),
\]

(2.24)

where \( na_i \) is the number of adjacencies for \( i \)th variable. In Figure 2.3, the size of CPT is 12.
Definition 2.21:

*Size of product term* is defined as the number of non-zero minterms in the PT. In Figure 2.1, size of PTs \((1[1(x_1)]) \cdot (1[1(x_2)])\), \((2[2(x_1)]) \cdot (x_2)\), \((x_1^{-1}) \cdot (3[3(x_2)])\), \((x_1^{2}) \cdot (x_2^{2})\), \((2[2(x_1)])\), and \((x_2^{1})\) are 2, 9, 6, 9, 12, and 12, respectively.

Definition 2.22:

A *level* of a CPT is defined as 1, if a minterm is expanded in all its adjacencies for all the variables. The *level* of the CPT shown in Figure 2.3 is 1. For a minterm, sometimes it is required to consider lower size CPTs. The smaller CPTs are generated by reducing the size of previous level CPT in one variable at a time. They are labelled as one level higher and so on. Some of the *level 2* CPTs for minterm \((3[3(x_2)])\) in Figure 2.2 are shown in Figure 2.4. In Figure 2.5, some *level 3* CPTs are shown that have been obtained from CPT1 shown in Figure 2.4.
Figure 2.4  For example function shown in Figure 2.2, some level 2 CPTs for minterm \((3^{1 \bar{x}_1^{1'}} \bullet 3^{1 \bar{x}_2^{1'}})\).

Figure 2.5  For example function shown in Figure 2.2, some level 3 CPTs for minterm \((3^{1 \bar{x}_1^{1'}} \bullet 3^{1 \bar{x}_2^{1'}})\).
3. Basic Circuit Elements

Information in an integrated circuit (IC) is conveyed using electrical signals. Depending on the technology used for circuit realization, these signals are represented by voltage, current, or charge. For example, charge is used for information transfer and storage in charge-coupled devices (CCD) technology. In digital logic circuits, these signals are quantized and various logic levels are represented in terms of amount of charge, voltage signal value, or current signal values. For example, in CMOS the voltage signals $V_{dd}$ (the power supply) and $Gnd$ (the ground) are used to represent binary high and binary low, respectively.

Implementation of MVL circuits using I²L, charge-coupled devices (CCD), voltage-mode CMOS (VMCL), and current-mode CMOS (CMCL) technologies have been reported in the literature [2, 11, 20]. Among these, MVL implementation using CMOS technology is becoming attractive since it can make use of existing binary logic CMOS fabrication processes.

A CMOS realization of MVL circuits utilizing both CMCL and VMCL circuits is reported in this thesis. The VMCL basic circuit elements consist of binary logic circuits, such as inverter, NAND, NOR, and complex CMOS gates, which are discussed in any text book on CMOS VLSI [59, 60]. However, in order to understand the operation of designed MVL circuits, it is necessary to understand CMCL. In the following section, current-mode CMOS logic technology will be discussed briefly.

3.1 Current-mode CMOS logic

As the name implies, current is used for information transfer and storage in this technology. The measure of current represents logic values and the direction of current flow determines the sign of the logic value. There are many ways of assigning
current values to logic levels. One such assignment is: logic 0 = 0 (no current), logic 1 = $I_0$, logic 2 = $2I_0$, and so on. This thesis adopts the above assignment with $I_0 = 20 \ \mu A \ [58]$. For sign of logic values, consider a node $n_1$ in a circuit. Current can flow in two directions, either into the node $n_1$ or out of the node $n_1$. One direction of flow can be used to represent a positive sign whereas the other can represent a negative sign. For example, if current flowing into the node $n_1$ is termed as a positive current, then $20 \ \mu A$ current flowing into the node $n_1$ is represented as logic 1 and the same amount of current flowing out of the node $n_1$ represents a logic -1.

In general, for all current-mode circuits addition and subtraction of signals are simple to perform. This is because all the current signals connected at a node satisfy Kirchoff's current law of electrical networks. Consider the operation of a binary current-mode CMOS inverter circuit, shown in Figure 3.1. The circuit consists of a source that can supply $I_0$ amount of current and two sinks, input and output. The amount of current flowing from node $n_2$ to $n_1$ and node $n_2$ to $n_3$ depends on the current sinking capacity of the input and output, respectively. Assume that the required two logic levels are represented as logic 0 = 0 (no current) and logic 1 = $I_0$. The necessary states to implement a logical inverter are outlined in Table 3.1. When there is a “0” on the input (i.e., it cannot sink any current), there is no current flowing between nodes $n_1$ and $n_2$. All of the $I_0$ current flows to the output, through nodes $n_2$ and $n_3$. Thus, the output is at logic 1 in accordance with the table. Similarly, when there is a “1” on the input, i.e., $I_0$ current flowing from node $n_2$ to node $n_1$, all of the

![Figure 3.1 CMCL binary inverter circuit realization.](image)
Table 3.1  Truth table for binary inverter

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

source current $I_0$ flows to the input and there is no current flowing between nodes $n_2$ and $n_3$. Thus, in accordance with the table, the output is at logic 0. It should be noted that it is also possible to realize a binary inverter when input and output are current sources rather than current sinks. For this the $I_0$ current source will have to be replaced by a $I_0$ current sink in Figure 3.1.

The above realization can be extended to multiple-valued logic inverter. For 4-valued logic inverter, the necessary states are outlined in Table 3.2. In order to realize a circuit satisfying this table, it is necessary to change the $I_0$ current source in Figure 3.1 to a $3I_0$ current source (i.e., $(r - 1)I_0$). When the input is at logic 0, all of the $3I_0$ current flows through nodes $n_2$ to $n_3$. Thus, the output is at logic 3. However, when the input is at logic 1, the $3I_0$ current splits into two, $I_0$ current flows from node $n_2$ to $n_1$ and $2I_0$ current flows from node $n_2$ to $n_3$. Thus, in accordance with the table, the output is at logic 2. Similarly, other states can be verified.

Current-mode CMOS circuits suffer a number of drawbacks as compared to voltage-mode CMOS circuits. Firstly, there is a static power dissipation. Secondly, usually the fanout of a gate or circuit element is restricted to one, as will be seen in later sections. Lastly, for a given range of current values used to represent various logic levels, noise margin is poorer for high radix current-mode circuits. However, the
advantage in implementing summation and subtraction operations, and the ease of specifying multiple-valued logic signals allow to overcome the above disadvantages at chip level. For example, Kameyama et al. [11] have reported a 32 x 32 bit signed digit (SD) multiplier implementation using MVL circuits realized in current-mode CMOS. They have reported that the chip area and power dissipation of MVL multiplier circuits is reduced to half of that of the fastest conventional binary realization of the same multiplier.

There are five basic circuit elements in CMCL: sum, constant, current-mirror, threshold, and switch. The basic CMCL circuit elements along with VMCL circuits have been used to realize various MVL circuits in CMOS. In order to understand the operation of realized MVL circuits (as discussed in the following chapters), it is necessary to understand the operation of each CMCL circuit element. In the following sections, the logic operation, symbol, and circuit realization for each of the above mentioned element will be discussed.

3.2 Sum

The logic operation associated with the sum circuit element is

\[ y = x_1 + x_2 + \ldots + x_n \]  \hspace{1cm} (3.1)

where \( y \) is the output and \( x_1, x_2, \ldots, x_n \) represent the \( n \) inputs. The circuit element is realized by connecting all inputs together. The circuit realization and symbol used to represent it are shown in Figure 3.2. A sum circuit obeys Kirchhoff's current law (i.e., summation of all currents into and out of a node must equal zero).

3.3 Constant

The logic operation for the constant circuit element is

\[ y = k \]  \hspace{1cm} (3.2)
where $y$ is the output and $k \in \{0, 1, 2, \ldots, r - 1\}$ for $r$-valued logic. The circuit can be realized either using enhancement mode n-type or p-type transistors depending on whether sinking or sourcing action is desired. Figure 3.3 and Figure 3.4 show the circuit realization for n-type and p-type constant circuit element, respectively. It should be noted that the symbols used to represent n-type and p-type constant circuits are also shown in Figure 3.3 and Figure 3.4, respectively.

The circuit operation is based on the assumption that the n-type or p-type transistor is in saturation region, i.e., $(V_{DS} > V_{GS} - V_T)$ and $V_{GS} > V_T$ where $V_{DS}$ is the voltage between drain and source, $V_{GS}$ is the voltage between gate and source, and $V_T$ represents the threshold voltage of the transistor. Typical values for the
threshold voltage for n-type \( V_{Tn} \) and p-type \( V_{Tp} \) transistors are 0.7 V and -0.8 V, respectively. Under saturation the drain current for the transistor is given by [59, 60]

\[
I_D = \frac{K' W}{2L} (V_{GS} - V_T)^2
\]  

(3.3)

where \( I_D \) is the drain current, \( K' \) is the transconductance parameter, \( W \) is the width and \( L \) is the length of the transistor. In the above equation, if \( V_{GS} \) is kept constant then different current values can be obtained by varying the size of transistor (i.e., \( W \) and \( L \)).

In order to generate \( I_0 = 20 \, \mu A \) using minimum sized n-type and p-type transistors (i.e., \( W : L = 1 : 1 \)), \( V_{GS} \) voltage for the transistor must be equal to \( N_{ref} \) and \( P_{ref} \), respectively. Using typical Northern Telecom CMOS3DLM process parameters, HSPICE simulations were carried out for different \( V_{GS} \) voltages for both n-type and p-type constants. It was found that it is necessary to have \( N_{ref} = 1.61 \, V \) and \( P_{ref} = 2.43 \, V \). It is possible to generate these reference voltages using an inverter (designed with appropriately sized n-type and p-type transistors) and with the output connected to the input. Figure 3.5 shows a possible realization for generating \( N_{ref} \) and \( P_{ref} \) voltages. Higher logic values can be generated by increasing the width of transistor shown in Figure 3.3 or Figure 3.4. For example, to generate \( 2I_0 \) the size of the transistor (ratio of \( W \) and \( L \)) in Figure 3.3 or Figure 3.4 must be \( W/L = 2 \). In
Figure 3.5 Circuit for realizing $N_{ref}$ and $P_{ref}$ voltages.

general, to obtain $kI_0$ current the size of the transistor must be $W/L = k$.

3.4 Current Mirrors

Current mirrors allow replication of input current. They can also be used to provide a multiplication factor at the output. There are two types of current mirrors, n-type and p-type, as shown in Figure 3.6 and Figure 3.7, respectively.

Figure 3.6 N-type current mirror circuit element. (a) Circuit realization (b) Symbol.
In these circuits, $T_0$ represents the input transistor and $T_1, T_2, \ldots, T_n$ are the output transistors. If the size of all the transistors in a current mirror circuit is equal (i.e., $W_0 : L_0 = W_1 : L_1 = \ldots = W_n : L_n$) then the multiplication factor is unity. In general, the logic operation associated with the current mirror is

$$y_i = a_ix$$

where $y_i$ is the $i$th output, $a_i$ is called the scale factor for the $i$th output, and $x$ represents the input current. For the above relationship to be true, it is necessary that the size of $i$th output transistor is $W_i/L_i = a_iW_0/L_0$.

Current mirrors are used to provide fanouts of greater than one, since CMCL only allows a fanout of 1. In addition, the current direction is reversed using a current mirror, regardless of the mirror type. This is useful while realizing a subtraction operation, as will be seen later.

### 3.5 Threshold

The threshold circuit element is a current to voltage converter. There are two types of this circuit: n-type and p-type, as shown in Figure 3.8 and Figure 3.9,
respectively. When the input current, \( x \), is less than some specified current level for an n-type threshold element, \( k \), then the voltage signal, \( y \), is binary low; otherwise it is binary high. The function of the n-type circuit is defined as

\[
\text{thresh}(x,k) = \begin{cases} 
\text{binary high} & \text{if } x \geq k \\
\text{binary low} & \text{otherwise.}
\end{cases}
\] (3.5)

Similarly, the logic operation of the p-type circuit is

\[
\text{thresh}(x,k) = \begin{cases} 
\text{binary low} & \text{if } x \geq k \\
\text{binary high} & \text{otherwise.}
\end{cases}
\] (3.6)

It should be noted that these circuits are similar to the constant circuit. In the circuits (see Figure 3.8 and Figure 3.9), \( k \) indicates the maximum amount of current that can flow through the transistor. The value of \( k \) depends on the size of n-type and p-type transistors and is equal to \((W/L)I_0\), where \( W \) is the width and \( L \) is the length of the transistor. As the operation of p-type circuit is complementary to that of n-type circuit, the following discussion would be restricted to n-type circuit.

![N-type threshold circuit element](image)

**Figure 3.8** N-type threshold circuit element. (a) Circuit realization (b) Symbol.
The circuit shown in Figure 3.10 is used to study the relationship between the input current and the output voltage signals for a threshold circuit element. The circuit consists of a p-type current mirror, three n-type threshold elements ($T_a$, $T_b$, and $T_c$), and three pairs of inverters. The sizes of $T_a$, $T_b$, and $T_c$ are $W_a/L_a = 0.5$, $W_b/L_b = 1.5$, and $W_c/L_c = 2.5$ to detect logic 1, logic 2, and logic 3 levels. Typically the threshold element output is used to control a switch, as will be seen later in Chapter 4. For proper operation of switch, the pairs of inverters are required to obtain binary high as $V_{dd}$ and binary low as $Gnd$ at the output nodes of threshold circuit element. Both the DC analysis and transient analysis were carried out for the circuit shown in Figure 3.10.

For DC analysis, the input $x$ was varied from 0 to 100 $\mu$A. The simulation results (see Figure 3.11) show the corresponding variations in the currents $x$, $I_a$, $I_b$, and $I_c$ (Panel 1) and the voltages at nodes $y_a$, $y_b$, $y_c$, $A$, $B$, and $C$ (Panel 2) as a function of the input current, $x$. It can be seen from the figure that as the input current increases, the current $I_a$ saturates at 8 $\mu$A (approximately $0.5I_0$). Similarly, the currents $I_b$ and $I_c$ saturate at 29 $\mu$A (approximately $1.5I_0$) and 49 $\mu$A (approximately $2.5I_0$), respectively. The voltages at nodes $y_a$, $y_b$, and $y_c$ rise when the currents saturate, but do not reach $V_{dd}$. However, the voltage swing for nodes $A$, $B$, and $C$ is between
Figure 3.10 Circuit used to study the input current and the output voltage relationship of a threshold circuit element.

$Gnd$ and $V_{dd}$. It can also be seen from the figure that the transition from $Gnd$ to $V_{dd}$ for node voltages $A$, $B$, and $C$ occur at $x = 7 \mu A$, $x = 28 \mu A$, and $x = 50 \mu A$, respectively.

For transient analysis, the input $x$ was varied from logic 0 through to logic 3. The simulation results (two panels) are shown in Figure 3.12. Panel 1 shows the variation of current $x$, $I_a$, $I_b$, and $I_c$ and Panel 2 shows voltage variation at nodes $y_a$, $y_b$, $y_c$, $A$, $B$, and $C$, as a function of time. It can be seen from the figure that the current $I_a$ saturates at $8 \mu A$. Similarly, the currents $I_b$ and $I_c$ saturate at $29 \mu A$ and $49 \mu A$, respectively. The voltages at nodes $y_a$, $y_b$, and $y_c$ rise when the currents saturate. When the input switches from logic 0 to logic 1, the voltage at node $A$ changes from $Gnd$ to $V_{dd}$. Similarly, when the input changes from logic 1 to logic 2 and logic 2 to logic 3, the voltage at nodes $B$ and $C$ also changes from $Gnd$ to $V_{dd}$, respectively.
Figure 3.11 DC analysis simulation results for the circuit shown in Figure 3.10.
Figure 3.12 Transient analysis simulation results for the circuit shown in Figure 3.10.
3.6 Switch

A switch is an n-type or a p-type transistor controlled by a voltage signal at the gate, $V_{in}$ (see Figure 3.13). When the switch is ON, the current can flow; otherwise no current flows. The logic operation of a switch is defined as

$$y = \begin{cases} x & \text{if switch is ON} \\ 0 & \text{(no current)} \end{cases}$$

(3.7)

It is necessary to use the proper switch type that is compatible to other circuit elements in a given circuit. For example, if the constant or threshold element in the circuit is an n-type, then use of an n-type switch allows a much wider current range for operation as compared to the one obtained by using a p-type switch. Similarly, for a p-type constant or threshold element the use of a p-type switch is better than the use of an n-type switch.

The circuit for both types of switches using a p-type constant, transistor $T_1$, are shown in Figure 3.14. Simulation was carried out for various sizes of n-type and p-type switches (transistor $T_2$) and for different current values, that were obtained by varying the size of transistor $T_1$. Simulation results are summarized in Figure 3.15 and Figure 3.16 for n-type and p-type switches, respectively, where current $I_{d1}$ is plotted versus transistor size of $T_1$ for different type of switches and for various sizes. From Figure 3.15, it can be seen that the current $I_{d1}$ saturates for n-type switches as the size of p-type constant is increased. For p-type switches, however, there is an increase in the current $I_{d1}$ as the size of transistor $T_1$ in increased (see Figure 3.16). For the transistor sizes of $T_2$, ($W_2/L_2 = 6/3$ and $W_2/L_2 = 9/3$), the increase is almost linear over a wide range. It is seen that for a p-type constant source, the use of a p-type switch is better than the use of an n-type switch. Similar analysis can be carried out for an n-type constant sink.
Figure 3.13 *Switch* circuit element. (a) N-type circuit realization. (b) P-type circuit realization. (c) Symbol.

Figure 3.14 Circuit for evaluating performance of *switches* (a) N-type switch (b) P-type switch.
Figure 3.15 Performance of n-type switch using a p-type constant.
Figure 3.16 Performance of p-type switch using a p-type constant.
3.7 Summary

In this chapter, basic circuit elements required for CMOS realization of MVL circuits have been considered. They consist of CMCL and VMCL circuits. The VMCL basic circuit elements consist of binary logic circuits, such as inverter, NAND, NOR, and complex CMOS gates. There are five basic circuit elements in CMCL, namely sum, constant, current-mirror, threshold, and switch. The logic operation, symbol, and circuit realization for each CMCL circuit element have been discussed. These circuits along with VMCL elements have been used to realize MVL circuits, as discussed in the forthcoming chapters.
4. Multiple-Valued Logic Operator Circuits

A set of operators $S$ is said to be functionally complete if it is possible to represent any function using only operators from $S$. Functional completeness can be achieved in more than one way. In 1921, Post [61] had shown that $cycle$, $min$, and $max$ form a functionally complete set of operators. In 1968, Allen and Givone [33] had proposed an algebra based on a set of operators consisting of $window$ literal, $min$, and $max$ operators. This set of operators was enhanced by Su and Cheung [34]. They added $complement$ of $window$ literal operator, which was defined by Su and Sarris [57]. In 1970, Vranesic et al. [35] had proposed an algebra using $cycle$, $min$, $max$, and $unary inverter$ operations. In addition, it has been shown that $window$ literal ($a^b$), $min$ ($\bullet$), and $tsum$ ($\oplus$) are a functionally complete set of operators [11, 30]. The choice of a functionally complete set reflects the efficiency of the underlying implementation of functions, which is dependent on the technology used for fabrication.

In this chapter, a functionally complete set of operators is proposed for the realization of MVL functions [62]. These operators have been selected by studying the existing set of operators, in order to identify efficient building blocks for circuit realization in CMOS. The set includes: literal, $cycle$, $complement$ of literal, $complement$ of cycle, $min$, and $tsum$ operators. In addition, level restorer circuits are also discussed. All reported circuit realizations use both current-mode and voltage-mode CMOS circuit elements. Current-mode signals are used to represent MVL levels whereas voltage-mode signals are binary and are used to control switches. The voltage-mode signals are referred to as $binary$ high and $binary$ low in the further discussion. It is shown that the cost (in terms of the number of transistors) of realizing a literal operator is the same as that of realizing the complement of literal operator.
Similarly, the cost of realizing a cycle operator or the complement of cycle operator is also the same. In addition, the costs of realizing the literal and the cycle operators are comparable. All circuits reported for the realization of the literal, the cycle operator, and their complements are programmable at the metal level in fabrication. Typical HSPICE transient analysis simulation results to verify the functionality of the designed circuits are included. For all simulations, Northern Telecom CMOS3DLM parameters provided by Canadian Microelectronics Corporation (CMC) have been used and are performed using the HSPICE MOS level 3 model.

This chapter consists of four sections. In Section 4.1, a new structure is proposed that is used for the realization of MVL operators. Circuit realizations for the proposed set of operators is discussed in Section 4.2. The circuit realization of all the operators of the proposed set of operators, except min, is based on the structure discussed in Section 4.1. Example simulation for each circuit discussed in Section 4.2 along with the effect of variations in process parameters are included in Section 4.3. The chapter summary is given in Section 4.4.

4.1 General Structure

The proposed general structure is shown in Figure 4.1 [63]. The structure makes use of both CMCL and VMCL. It consists of three blocks: input, control, and output. The input and the output blocks handle MVL current signals whereas the control logic block deals with binary voltage signals. The input block circuit is primarily made up of current mirrors and threshold circuit elements. The output block circuit consists of switches, constants, and current mirrors. The function of the input block is to generate voltage signals for input current signals by using threshold circuits. In the control block, the voltage signals generated by the input block are combined using different binary operators, such as inverter, NAND, NOR, and complex CMOS gates, to obtain control signals. These control signals are used to control the gates of the switches in the output block.
Typical realizations of the *input* and the *output* blocks (for sinking current) are shown in Figure 4.2. The *control* block consists of binary circuits. In the following section, the realization of MVL operators using the general structure (shown in Figure 4.1) will be discussed.

![Figure 4.1 Block diagram of the structure used for realizing MVL operators.](image)

### 4.2 Circuit Implementation

#### 4.2.1 Min Operator

The realization of min operator is based on the truncated difference operator which is defined as [64, 65]:

\[
x \triangle y = \begin{cases} 
x - y & \text{if } x \geq y \\
0 & \text{otherwise.}
\end{cases}
\]

(4.1)

The min operation can be defined as:

\[
\text{min}(x, y) = x - (x \triangle y) = y - (y \triangle x).
\]
In the above definition, the subtraction operation is realized by reversing the direction of the subtrahend using a current mirror and then adding the reversed current to the minuend.

There are two realizations for the min operator depending on whether the input currents are sourcing or sinking. These are shown in Figure 4.3, which are based on CMCL [66]. In these circuits, $I_3 = (x \oplus y)I_0$ and $I_4 = xI_0 - I_3$. To understand the operation of the circuit shown in Figure 4.3, it is necessary to consider two cases: $x < y$ and $x \geq y$. When $x < y$ then $I_1 = xI_0$ and $I_2 = 0$. So, $I_3 = 0$ and $I_4 = xI_0 - I_3 = xI_0$, which means $\min(x, y) = x$. However, when $x \geq y$ then $I_1 = xI_0$ and $I_2 = xI_0 - yI_0$. Thus, $I_3 = (x - y)I_0$ and $I_4 = xI_0 - I_3 = xI_0 - (x - y)I_0 = yI_0$, which means $\min(x, y) = y$.

### 4.2.2 Tsum Operator

The function of the tsum operator is to restrict the output current level to $(r - 1)I_0$. For $r$-valued $n$-variable functions, the tsum operator can be realized as
min(x_1 + x_2 + ... + x_n, r - 1), as shown in Figure 4.4. In this realization the output levels may deteriorate because of three stage mirror action and the subtraction operation in the min operator circuit implementation. This problem can be solved by using the realization shown in Figure 4.5. The realization is based on the structure shown in Figure 4.1. In this circuit, the size of transistor T_1 is set equal to \( W_1/L_1 = (r - 1 - 0.5) \) to detect the logic levels greater than \( r - 2 \) and the size of transistor T_2 is \( W_2/L_2 = (r - 1) \) to source \( I_2 = (r - 1)I_0 \) current when switch, \( S_2 \) is ON. Consider the following
two cases in order to understand the operation of the circuit: $0 < Y < (r - 1)$ and $(r - 1) \leq Y$.

**Case 1:** If $0 < Y < (r - 1)$ then $I_1 = YI_0$ and the voltage at node A is *binary low*. The voltage at node B is *binary high*. The switch, $S_1$ is ON and the switch, $S_2$ is OFF. The current $I_3 = YI_0$ and $I_4 = 0$, and the output current $I_5$ is equal to $I_3 + I_4$, which is $YI_0$. The output logic level is $Y$ (i.e., $x_1 + x_2 + \ldots + x_n$).

**Case 2:** If $(r - 1) \leq Y$ then $I_1 = (r - 1 - 0.5)I_0$ and the voltage at node A is *binary high*. The voltage at the node B is *binary low*. The switch, $S_1$ is OFF and the switch, $S_2$ is ON. The current $I_3 = 0$ and $I_4 = (r - 1)I_0$, and the output current $I_5$ is equal to $(r - 1)I_0$. The output logic level is truncated to $(r - 1)$ (i.e., $\min(x_1 + x_2 + \ldots + x_n, r - 1)$).

Figure 4.5  Circuit realization of $tsum$ using the structure shown in Figure 4.1.
4.2.3 Literal and Complement of Literal Operator

For any 4-valued variable there are ten possible literals, namely \( k[0(x)^0] \), \( k[1(x)^1] \), \( k[2(x)^2] \), \( k[3(x)^3] \), \( k[1(x)^1] \), \( k[2(x)^2] \), \( k[3(x)^3] \), and \( k[0(x)^0] \). It should be noted that \( k[0(x)^0] \) represents a constant \( k \). A circuit for realizing a literal, \( k[0(x)^0] \) is shown in Figure 4.6, where the size of transistor \( T_3 \) must be equal to \( W_3/L_3 = k \). If

\[
k = r - 1,
\]

then the window literal, \( a x^b \), is realized. Thus, the realization of \( a x^b \) is a special case of \( k[0(x)^0] \). In the literal operator circuit, transistor \( T_1 \) acts as a threshold element for current level \( (a - 0.5)I_0 \) and \( T_2 \) acts as a threshold element for current level \( (b + 0.5)I_0 \), where \( a > 0 \). Thus, current \( I_a \) is restricted to \( (a - 0.5)I_0 \) and \( I_b \) is restricted to \( (b + 0.5)I_0 \). To understand the operation of the circuit, it is necessary to consider the following three cases: \( 0 < x < a \); \( a \leq x \leq b \); and \( b < x < r \).

**Case 1:** If \( 0 < x < a \) then \( I_a = I_b = xI_0 \). The voltages at nodes A and B are binary low. The voltage at node C is also binary low and the
switch controlled by C is OFF. The output current is zero, therefore \( k[^{a}x^{b}] = 0 \).

Case 2: If \( a \leq x \leq b \) then \( I_a = (a - 0.5)I_0 \) and \( I_b = xI_0 \). The voltages at nodes A and B would be binary high and binary low, respectively. The voltage at node C will be binary high and the switch controlled by C will be ON. The output current will depend on the size of transistor \( T_3 \). If the size of transistor \( T_3, W_3/L_3 = k \) then the output current is \( kI_0 \), therefore \( k[^{a}x^{b}] = k \).

Case 3: If \( b < x < r \) then \( I_a = (a - 0.5)I_0 \) and \( I_b = (b + 0.5)I_0 \). Both the nodes A and B are binary high. The voltage at node C is binary low and the switch controlled by C is OFF. The output current is zero, therefore \( k[^{a}x^{b}] = 0 \).

Various logic levels for \( k \) (i.e., 1, 2, ..., \( r - 1 \)) can be realized by specifying the size of transistor \( T_3 \) equal to \( k \) (i.e., \( W_3/L_3 = k \)). For example, to realize \( 2[^{a}x^{b}] \) the size of transistors \( T_1, T_2, \) and \( T_3 \) should be \( W_1 : L_1 = 1 : 2, W_2 : L_2 = 5 : 2, \) and \( W_3 : L_3 = 2 : 1 \), respectively. It should be noted that the size of transistors is specified as the ratio of width and length of transistors; odd size transistors can be realized as one large transistor or a few small transistors in parallel. For example, transistor \( T_2 \) can be realized as one large transistor having an actual size \( W_2 : L_2 = 15 \mu m : 6 \mu m \) using CMOS3DLM technology, where the smallest transistor size is \( 3 \mu m : 3 \mu m \). The other possible realization is to connect two transistors of sizes \( 6 \mu m : 3 \mu m \) and \( 3 \mu m : 6 \mu m \) in parallel, with their gates connected together. It should be noted that in the following discussion, only \( W/L \) ratios have been specified for transistor sizes.

The circuit in Figure 4.6 can be modified to realize a complement of literal, which is defined as:

\[
k[^{a}x^{b}] = \begin{cases} 
  k & \text{if } ^{a}x^{b} = \text{binary low} \\
  0 & \text{otherwise,}
\end{cases} \tag{4.2}
\]
where $a, b \in R$ and $k \in \{1, 2, \ldots, r - 1\}$. The modified circuit is shown in Figure 4.7. The operation of this circuit is similar to the operation of the circuit shown in Figure 4.6, the only difference is that the voltage at node C in Figure 4.6 is inverted for the circuit shown in Figure 4.7. A complement of literal is used to realize certain literals that cannot be realized by the circuit shown in Figure 4.6. In the circuit shown in Figure 4.6, if $a = 0$ then it is not possible to realize $k[\overline{\overline{a}(x)^k}]$. However, this literal can be realized by the circuit shown in Figure 4.7 as $k[\overline{a}^b(x)^{1-k}]$. For example, $k[\overline{a}(x)^1]$ is realized as $k[\overline{a}(x)^2]$. It should be noted that both the literal operator and the complement of literal operator circuits need the same number of transistors. Thus, the use of the complement of literal operator could be as advantageous as the use of the literal operator. Different values for $a$, $b$, and $k$ can be realized by varying the size of transistors $T_1$, $T_2$, and $T_3$. Table 4.1 summarizes realization of 10 literals along with sizes of transistors.
Table 4.1  Realization of literals and size of transistors $T_1$, $T_2$, and $T_3$ in 4-valued logic.

<table>
<thead>
<tr>
<th>Literal Operator</th>
<th>Circuit Realization</th>
<th>Size of Transistors ($W/L$ Ratios)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k[0(x)^0]$</td>
<td>$k[1(x)^3]$</td>
<td>$T_1(W_1:L_1)$ 7:2 $T_2(W_2:L_2)$ 2:1 $T_3(W_3:L_3)$ 4:1</td>
</tr>
<tr>
<td>$k[1(x)^1]$</td>
<td>$k[1(x)^1]$</td>
<td>$T_1(W_1:L_1)$ 7:2 $T_2(W_2:L_2)$ 2:1 $T_3(W_3:L_3)$ 4:1</td>
</tr>
<tr>
<td>$k[2(x)^2]$</td>
<td>$k[2(x)^2]$</td>
<td>$T_1(W_1:L_1)$ 7:2 $T_2(W_2:L_2)$ 2:1 $T_3(W_3:L_3)$ 4:1</td>
</tr>
<tr>
<td>$k[3(x)^3]$</td>
<td>$k[3(x)^3]$</td>
<td>$T_1(W_1:L_1)$ 7:2 $T_2(W_2:L_2)$ 2:1 $T_3(W_3:L_3)$ 4:1</td>
</tr>
<tr>
<td>$k[0(x)^1]$</td>
<td>$k[2(x)^2]$</td>
<td>$T_1(W_1:L_1)$ 7:2 $T_2(W_2:L_2)$ 2:1 $T_3(W_3:L_3)$ 4:1</td>
</tr>
<tr>
<td>$k[1(x)^1]$</td>
<td>$k[1(x)^1]$</td>
<td>$T_1(W_1:L_1)$ 7:2 $T_2(W_2:L_2)$ 2:1 $T_3(W_3:L_3)$ 4:1</td>
</tr>
<tr>
<td>$k[2(x)^3]$</td>
<td>$k[2(x)^3]$</td>
<td>$T_1(W_1:L_1)$ 7:2 $T_2(W_2:L_2)$ 2:1 $T_3(W_3:L_3)$ 4:1</td>
</tr>
<tr>
<td>$k[3(x)^3]$</td>
<td>$k[3(x)^3]$</td>
<td>$T_1(W_1:L_1)$ 7:2 $T_2(W_2:L_2)$ 2:1 $T_3(W_3:L_3)$ 4:1</td>
</tr>
<tr>
<td>$k[1(x)^1]$</td>
<td>$k[1(x)^1]$</td>
<td>$T_1(W_1:L_1)$ 7:2 $T_2(W_2:L_2)$ 2:1 $T_3(W_3:L_3)$ 4:1</td>
</tr>
<tr>
<td>$k[2(x)^2]$</td>
<td>$k[2(x)^2]$</td>
<td>$T_1(W_1:L_1)$ 7:2 $T_2(W_2:L_2)$ 2:1 $T_3(W_3:L_3)$ 4:1</td>
</tr>
<tr>
<td>$k[3(x)^3]$</td>
<td>$k[3(x)^3]$</td>
<td>$T_1(W_1:L_1)$ 7:2 $T_2(W_2:L_2)$ 2:1 $T_3(W_3:L_3)$ 4:1</td>
</tr>
<tr>
<td>$k[0(x)^1]$</td>
<td>constant $k$</td>
<td>$T_1(W_1:L_1)$ 7:2 $T_2(W_2:L_2)$ 2:1 $T_3(W_3:L_3)$ 4:1</td>
</tr>
</tbody>
</table>

$T_1$, $T_2$, and $T_3$ for 4-valued logic.

With the addition of the complement of literal the number of available literals increases. For 4-valued logic, there are three more literals: $k[\overline{1(x)^1}]$; $k[\overline{2(x)^2}]$; and $k[\overline{3(x)^3}]$. For $x =<0123>$, $k[\overline{1(x)^1}] =<k0k>$; $k[\overline{2(x)^2}] =<k0k>$; and $k[\overline{3(x)^3}] =<k0k>$. These literals are realized using the complement of literal circuit (see Figure 4.7) and the transistor sizes are the same as those of $k[1(x)^1]$, $k[2(x)^2]$, and $k[3(x)^3]$, respectively (from Table 4.1).

4.2.4 Cycle and Complement of Cycle Operator

As defined previously (see Section 2.1), there are two types of cycle operators: clockwise and counter-clockwise. These two types are related by [35]:

$$x^m = x^{-m}.$$  \hspace{1cm} (4.3)

In this discussion, only one kind of operator will be considered, the generic cycle operator. A circuit implementation for realizing the cycle operator ($x^m$) is shown in Figure 4.8. In this circuit, transistor $T_1$ functions as a threshold element for $(m-0.5)I_0$
Figure 4.8 Circuit for realizing a cycle operator, $x^m$.

$(m > 0)$ current level. The sizes of transistors $T_2$ and $T_3$ are set equal to $W_2/L_2 = r - m$ and $W_3/L_3 = m$, respectively. The pair of inverters are required to obtain binary high as $V_{dd}$ and binary low as Gnd at node B for proper operation of switches.

It should be noted that when $m = 0$, the cycle operator is not required as $x^0 = x$. To understand the operation of the circuit, consider the following two cases: $0 < x < m$ and $m < x < r$.

**Case 1:** If $0 \leq x < m$ then $I_1 = xI_0$ and the voltages at nodes A and B are binary low. The switch, $S_1$ is OFF and the switch, $S_2$ is ON. Under this condition, $I_2 = (r - m)I_0$ and $I_3 = 0$. The output current $I_4$ is equal to $xI_0 + I_2 - I_3$, which is $(r - m)I_0$ more than the input, $xI_0$. A logic level of $r - m$ is added to the input to obtain the output, $x^m = x - m + r$. 
Case 2: If \( m < x < r \) then \( I_1 = (m - 0.5)I_0 \) and the voltages at nodes A and B are binary high. The switch, \( S_1 \) is ON and the switch, \( S_2 \) is OFF. Under this condition, \( I_2 = 0 \) and \( I_3 = mI_0 \). The output current \( I_4 \) is less than the input, \( xI_0 \) by amount \( mI_0 \), and \( x^m = x - m \).

Circuit implementations can be obtained for various values of \( m \) by varying the size of transistors \( T_1 \), \( T_2 \), and \( T_3 \). Table 4.2 shows the size of all three transistors for values \( m = 1 \), \( m = 2 \), and \( m = 3 \) for 4-valued logic.

**Table 4.2** Size of transistors \( T_1 \), \( T_2 \), and \( T_3 \) for the realization of cycle operator in 4-valued logic.

<table>
<thead>
<tr>
<th>Cycle Operator</th>
<th>Size of Transistors (W/L Ratios)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( T_1(W_1:L_1) )</td>
</tr>
<tr>
<td>( x^1 )</td>
<td>1:2</td>
</tr>
<tr>
<td>( x^2 )</td>
<td>3:2</td>
</tr>
<tr>
<td>( x^5 )</td>
<td>5:2</td>
</tr>
</tbody>
</table>

The *complement of cycle* operator is defined as:

\[
\overline{x^m} = (r - 1) - x^m.
\]

Considering \( r = 4 \), for \( x = \langle 0123 \rangle \), \( x^1 = \langle 1230 \rangle \) and \( x^1 = \langle 2103 \rangle \). The *complement of cycle* operation can be realized using the following relationship:

\[
\overline{x^m} = \overline{x} \overline{r-m} = \overline{x^m}.
\]

This implementation is advantageous if input variables and their complements are available. Considering \( r = 4 \), for \( x = \langle 0123 \rangle \), \( \overline{x} = \langle 3210 \rangle \) and \( \overline{x^1} = \langle 2103 \rangle \) which is equal to \( \overline{x^3} \) and \( \overline{x^1} \). If it is assumed that the complement of an input variable is available then the cost of realizing the cycle operator and the complement of cycle operator is the same.
4.2.5 Level Restorer Circuit

CMCL is not a self restoring logic and there is some deterioration of MVL current levels as signals propagate through various CMCL circuit elements. These MVL signals may be added or subtracted (such as in the realization of $t_{sum}$ operator), so the signal deterioration may be compounded and result in an incorrect logic level. Therefore, it may be necessary to restore the MVL current values to proper levels.

For 4-valued logic, a level restorer circuit implementation is shown in Figure 4.9. The circuit consists of an input CMCL block, an inverter, a complex CMOS gate, and an output CMCL block. The combination of the input block, the inverter, and the complex gate realize a decoder circuit to convert current-mode MVL signal to voltage-mode binary signals. The output block is used to realize an encoder circuit that converts voltage-mode binary signals to current-mode MVL signal. The input block uses a p-type current mirror and 3 threshold circuit elements ($T_1$, $T_2$, and $T_3$) for detecting logic levels 1, 2, and 3, respectively. The output block consists of 2 p-type switches ($S_1$ and $S_2$) and 2 p-type constants ($T_4$ and $T_5$) and is used to generate proper MVL signals at the output. The size of transistors $T_1$, $T_2$, $T_3$, $T_4$, and $T_5$ are $W_1/L_1 = 0.5$, $W_2/L_2 = 1.5$, $W_3/L_3 = 2.5$, $W_4/L_4 = 1$, and $W_5/L_5 = 2$, respectively. It should be noted that in this circuit $x$ represents the amount of input current and the output $y$ represents the logic level. To understand the operation of the circuit,
consider the following four cases: $0 \leq x < 0.5I_0$, $0.5I_0 \leq x < 1.5I_0$, $1.5I_0 \leq x < 2.5I_0$, and $2.5I_0 \leq x$.

**Case 1:** When $0 \leq x < 0.5I_0$, then $I_1 = I_2 = I_3 = x$ and the voltage at all the three nodes A, B, and C are binary low. The voltages at nodes $L_1$ and $L_2$ are both binary high. Both the switches, $S_1$ and $S_2$, are OFF. The currents $I_4 = I_5 = 0$, and the output current $I_4 + I_5$ is equal to 0. The output, $y$ is at logic level 0.

**Case 2:** If $0.5I_0 \leq x < 1.5I_0$, then $I_1 = 0.5I_0$ and $I_2 = I_3 = x$. The voltage at the node A is binary high and the voltages at nodes B and C is binary low. The voltage at the node $L_1$ is binary low and at the node $L_2$ is binary high. The switch, $S_1$ is ON and the switch, $S_2$ is OFF. The current $I_4 = I_0$ whereas the current $I_5 = 0$. The output current $I_4 + I_5 = I_0$. The output, $y$ is at logic level 1.

**Case 3:** For $1.5I_0 \leq x < 2.5I_0$, $I_1 = 0.5I_0$, $I_2 = 1.5I_0$, and $I_3 = x$. The voltages at the nodes A and B are binary high and the voltage at the node C is binary low. The voltage at the node $L_1$ is binary high and at the node $L_2$ is binary low. The switch, $S_1$ is OFF and the switch $S_2$ is ON and the currents $I_4 = 0$ and $I_5 = 2I_0$. The output current $I_4 + I_5 = 2I_0$, so the output, $y$ is at logic level 2.

**Case 4:** When $2.5I_0 \leq x$, then $I_1 = 0.5I_0$, $I_2 = 1.5I_0$, and $I_3 = 2.5I_0$. The voltages at all the three nodes A, B, and C are binary high, which implies that the voltages at both the nodes $L_1$ and $L_2$ are binary low. Both the switches, $S_1$ and $S_2$, are ON and the currents $I_4 = I_0$ and $I_5 = 2I_0$. The output current $I_4 + I_5 = 3I_0$, so the output, $y$ is at logic level 3.

It is possible to design another level restorer circuit by removing the complex CMOS gate and modifying the output CMCL block in the circuit shown in Figure 4.9.
The modified circuit is shown in Figure 4.10. The output block consists of 3 p-type switches ($S_1$, $S_2$, and $S_3$) and 3 p-type constants ($T_4$, $T_5$, and $T_6$), and the control block consists of 3 inverters. The size of transistors $T_1$, $T_2$, $T_3$, $T_4$, $T_5$, and $T_6$ are $W_1/L_1 = 0.5$, $W_2/L_2 = 1.5$, $W_3/L_3 = 2.5$, $W_4/L_4 = 1$, $W_5/L_5 = 1$, and $W_6/L_6 = 1$, respectively. The switches $S_1$, $S_2$, and $S_3$ control the currents $I_4$, $I_5$, and $I_6$, respectively, to be either 0 or $I_0$. The circuit operation is similar to the above discussion.

![Figure 4.10 Another realization of a level restorer circuit.](image)

These circuits can be used to guarantee that the signals generated by MVL circuits are correctly interpreted by the subsequent circuits, as will be shown in the following chapter.
4.3 Simulation

HSPICE transient analysis simulation was done to verify the functionality of the circuits discussed in the previous section. All simulations use HSPICE MOS level 3 model and Northern Telecom CMOS3DLM parameters, provided by Canadian Microelectronics Corporation (CMC), see Appendix A. It should be noted that for all the simulations current-mirrors have been used as typical loads.

4.3.1 Example Circuits

A. Min Operator

Consider the realization of $\text{min}(x_1, x_2) = x_1 \cdot x_2$ for $r = 4$ as an example. As discussed previously, the $\text{min}$ operator can be realized by the circuit shown in Figure 4.3. The simulation results for the realization of $\text{min}$ are shown in Figure 4.11, which consists of 3 panels. All the panels show current level variation as a function of time. Panel 1 and Panel 2 show all possible combinations for the input variables, $x_1$ and $x_2$, respectively, while Panel 3 shows the variation of $\text{min}(x_1, x_2)$. As can be noticed that the $\text{min}(x_1, x_2)$ output gives the expected logic levels for all possible combinations of inputs $x_1$ and $x_2$. The simulation results shown in Figure 4.12 are an expansion of part (900ns to 1100ns) of the simulation results shown in Figure 4.11. In Figure 4.12, $x_1$ is switching from logic 3 (i.e., 60 $\mu$A) to logic 0 (i.e., 0 $\mu$A) and $x_2$ is changing from logic 0 to logic 1. According to the functionality of the circuit the output must remain at logic 0, however, there is a spike of about 8 $\mu$A at the time of switching. Figure 4.13 shows how different input values affect the output. The input $x_1$ is switching from logic 1 to logic 2 but input $x_2$ remains at logic 1, so the output should remain at logic 1 (i.e., 20 $\mu$A). However, it can be seen that there is a change in the output current level (by approximately 3 $\mu$A). As the new value is still less than 30 $\mu$A (i.e., $1.5I_0$), the threshold level for detecting logic 2 level, it will be recognized as logic 1 by subsequent circuits (if connected). Figure 4.14 shows the transient analysis from 2.9 $\mu$s to 3.9 $\mu$s. The input $x_1$ is changing through all the four logic levels and input $x_2$ is switching from logic 2 to logic 3. According to $\text{min}$
Figure 4.11 HSPICE output for the transient analysis of the circuit realization of $\min(x_1, x_2)$. 
Figure 4.12 Expansion of HSPICE output for the transient analysis of the circuit realization of \( \min(x_1, x_2) \) for time 900ns to 1100ns.
Figure 4.13 Expansion of HSPICE output for the transient analysis of the circuit realization of \( \min(x_1, x_2) \) for time 1.4 \( \mu \)s to 1.6 \( \mu \)s.
Figure 4.14 Expansion of HSPICE output for the transient analysis of the circuit realization of $\min(x_1, x_2)$ for time 2.9 µs to 3.9 µs.
operation the output follows the input $x_1$, which can be seen in Figure 4.14.

B. Tsum Operator

To check the functionality of the *tsum* circuit, shown in Figure 4.5, consider the realization of $tsum(x_1, x_2)$ for $r = 4$ as an example. The simulation results for the realization are shown in Figure 4.15, which consists of 3 panels. All the panels show current level variation as a function of time. Panel 1 and Panel 2 show all possible combinations for the input variables, $x_1$ and $x_2$, respectively. Panel 3 shows the variation of $tsum(x_1, x_2)$. It can be seen from Figure 4.15 that while $x_2$ is at logic 0, the output follows the input $x_1$. When $x_2$ is at logic 3, the output is at logic 3 and is independent of input $x_1$, in accordance to the *tsum* operator definition. Similarly, it can be verified that the simulation results are correct for other logic values of the inputs. The simulation results shown in Figure 4.16 are an expansion of part (1.9 $\mu$s to 2.3 $\mu$s) of the simulation results shown in Figure 4.15.

C. Literal and Complement of Literal Operator

Consider the realization of $3[1_{\overline{x}_1}]$ and $2[1_{\overline{x}_1}]$ for $r = 4$. The *literal* operator, $3[1_{\overline{x}_1}]$ can be realized by the circuit shown in Figure 4.6. The size of transistors $T_1$, $T_2$, and $T_3$ are $W_1/L_1 = 0.5$, $W_2/L_2 = 2.5$, and $W_3/L_3 = 3$, respectively. Similarly, the *complement of literal* operator, $2[1_{\overline{x}_1}]$ can be realized by the circuit shown in Figure 4.7. The size of transistors $T_1$, $T_2$, and $T_3$ are $W_1/L_1 = 0.5$, $W_2/L_2 = 1.5$, and $W_3/L_3 = 2$, respectively. The simulation results for the realization of $3[1_{\overline{x}_1}]$ and $2[1_{\overline{x}_1}]$ are shown in Figure 4.17 for the input $x$ changing through all the four logic levels. The simulation consists of 3 panels, where all the panels show current level variations as a function of time. Panel 1 shows all the four logic levels for the input variable, $x$. Panel 2 and Panel 3 show variations of $3[1_{\overline{x}_1}]$ and $2[1_{\overline{x}_1}]$, respectively. In accordance to the *literal* operator definition, it is seen that if $x$ is at logic 1 or logic 2, the output, $3[1_{\overline{x}_1}]$ is at logic 3, otherwise it is at logic 0. Similarly according to the definition of the *complement of literal* operator, when $x$ is at logic 1, the output, $2[1_{\overline{x}_1}]$ is at logic 0, otherwise it is at logic 2.
Figure 4.15 HSPICE output for the transient analysis of the circuit realizations of \( tsum(x_1, x_2) \).
Figure 4.16 Expansion of HSPICE output for the transient analysis of the circuit realization of \( tsum(x_1, x_2) \) for time 1.9 \( \mu \)s to 2.3 \( \mu \)s.
Figure 4.17 HSPICE output for the transient analysis of the circuit realization of $3[x^2]$ and $2[x^1]$. 
D. Cycle Operator

Consider the realization of $x^2 = \langle 2301 \rangle$ (i.e., $x^2$) for $r = 4$. The cycle operator, $x^2$ can be realized by the circuit shown in Figure 4.8. The size of transistors $T_1$, $T_2$, and $T_3$ must be $W_1/L_1 = 1.5$, $W_2/L_2 = 2$, and $W_3/L_3 = 2$, respectively. The simulation results for the realization of $x^2$ are shown in Figure 4.18. The simulation results consist of 2 panels and show variations in current levels as a function of time. Panel 1 shows all four logic levels of the input variable, $x$ and Panel 2 shows corresponding variation for $x^2$. According to the definition of cycle operator, if $x = \langle 0123 \rangle$ then the output must be $x^2 = \langle 2301 \rangle$, which is verified by the simulation results.

In CMCL circuits propagation delay depends on the current levels switched at the input and the output. Average and maximum propagation delays, measured using HSPICE, for all the example simulations are summarized in Table 4.3.

<table>
<thead>
<tr>
<th>Example Circuits</th>
<th>HSPICE Simulation</th>
<th>Propagation Delay (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\min(x_1, x_2)$</td>
<td>Figure 4.11</td>
<td>2.04</td>
</tr>
<tr>
<td>$tsum(x_1, x_2)$</td>
<td>Figure 4.15</td>
<td>6.57</td>
</tr>
<tr>
<td>$3[\langle x \rangle^2]$</td>
<td>Figure 4.17</td>
<td>10.54</td>
</tr>
<tr>
<td>$2[\langle x \rangle^1]$</td>
<td>Figure 4.17</td>
<td>10.23</td>
</tr>
<tr>
<td>$x^2$</td>
<td>Figure 4.18</td>
<td>7.05</td>
</tr>
</tbody>
</table>

4.3.2 Effect of Process Parameter Variations

The above simulation results are for typical CMOS3DLM process parameters. To see the effect of variations in process parameters, HSPICE simulations were carried out for various process parameters. The variations in transistor threshold voltage and gate oxide thickness were studied. For Northern Telecom CMOS3DLM process, the variation in threshold voltage for n-type is $V_{tn} \pm 0.2$ V and for p-type is $V_{tp} \pm 0.2$ V,
Figure 4.18 HSPICE output for the transient analysis of the circuit realization of cycle, $x^2$, operator.
where $V_{in}$ and $V_{tp}$ are the nominal threshold voltages for n-type and p-type transistors, respectively. The variation in gate oxide thickness ($T_{ox}$) is $T_{oxn} \pm 3$ nm, where $T_{oxn}$ is the nominal value. Based on threshold voltage variations, five cases were analyzed. These cases are summarized in Table 4.4. It should be noted that Case I to Case IV are the extreme variations and Case V represents the typical values, used in all the previous simulations.

In each case, the simulation was done by varying the gate oxide thickness between $T_{oxn} - 3$ nm and $T_{oxn} + 3$ nm in steps of 1 nm. For $3[x^2]$ circuit, maximum propagation delay variation for all the five cases as a function of gate oxide thickness is shown in Figure 4.19. It can be seen that the propagation delay decreases as the gate oxide thickness is increased. This can be attributed to the fact that gate capacitance reduces as the gate oxide thickness increases. For other circuits, the variation for propagation delay as a function of gate oxide thickness is similar to Figure 4.19. Graphs have not been included for all the circuits, however, the range of maximum propagation delay values for each circuit are given in Table 4.5.

### Table 4.4

Different cases for simulation based on variation in threshold voltage for n-type and p-type transistors.

<table>
<thead>
<tr>
<th>Different Cases</th>
<th>Threshold Voltage (V)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n-type</td>
<td>p-type</td>
</tr>
<tr>
<td>Case I</td>
<td>$V_{in} - 0.2$</td>
<td>$V_{tp} + 0.2$</td>
</tr>
<tr>
<td>Case II</td>
<td>$V_{in} - 0.2$</td>
<td>$V_{tp} - 0.2$</td>
</tr>
<tr>
<td>Case III</td>
<td>$V_{in} + 0.2$</td>
<td>$V_{tp} + 0.2$</td>
</tr>
<tr>
<td>Case IV</td>
<td>$V_{in} + 0.2$</td>
<td>$V_{tp} - 0.2$</td>
</tr>
<tr>
<td>Case V</td>
<td>$V_{in} - 0.05$</td>
<td>$V_{tp} - 0.05$</td>
</tr>
</tbody>
</table>

4.4 **Summary**

In this chapter, a set of operators for CMOS implementation of MVL operators has been proposed. The set of operators consists of literal, cycle, complement of literal, complement of cycle, min, and tsum operators. A circuit realization for each operator has been included. These circuits use both current-mode and voltage-mode CMOS
Figure 4.19 Maximum propagation delay variation for $3^1(x^2)$ circuit.
Table 4.5  Limits for maximum propagation delay for process parameter variations.

<table>
<thead>
<tr>
<th>Example Circuits</th>
<th>Maximum Propagation Delay (ns)</th>
<th>Lower limit</th>
<th>Upper Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\min(x_1, x_2))</td>
<td>3.55</td>
<td>4.73</td>
<td></td>
</tr>
<tr>
<td>(t\sum(x_1, x_2))</td>
<td>10.25</td>
<td>18.45</td>
<td></td>
</tr>
<tr>
<td>(3[\overline{x}])</td>
<td>11.43</td>
<td>17.26</td>
<td></td>
</tr>
<tr>
<td>(2[\overline{x}])</td>
<td>8.91</td>
<td>13.28</td>
<td></td>
</tr>
<tr>
<td>(x^{-})</td>
<td>8.79</td>
<td>13.04</td>
<td></td>
</tr>
</tbody>
</table>

circuits. In addition, circuit implementations have been discussed for level restorer circuits that can be used to restore the MVL current values to proper level. It has been shown, with the aid of HSPICE transient analysis simulations, that MVL circuits can be implemented using standard binary CMC Northern Telecom CMOS3DLM process technology and function as expected. In addition, simulation results have been included to show the effect of variation in process parameters. The variations in threshold voltages (for n-type and p-type transistors) and gate oxide thickness have been studied. It is noted that the maximum propagation delays for the example circuits, which depend on the current levels switched at the input and the output, are in the range of 3.55 ns to 18.45 ns for process parameter variations and the range is 3.77 ns to 13.97 ns for typical process parameters.
5. Multiple-Valued Logic Function Realization: Sum-of-Products

In the previous chapters, basic circuit elements and circuit realization for the proposed set of operators have been discussed. Emphasis will now be put on the realization of MVL functions. There have been two approaches used for MVL function realization, namely using cost-tables [39, 40, 41, 42, 43, 44, 45], and sum-of-product (SOP) form realization [32, 33, 35, 38, 47, 48, 49].

In this chapter, MVL function realizations using the SOP form are considered. The realizations based on the set of operators proposed in Chapter 4 (literal, cycle, complement of literal, complement of cycle, min, and tsum) are compared to realizations based on the existing sets of operators consisting of: literal, complement of literal, min, and tsum operators; and literal, min, and tsum operators [67]. For r-valued n-variable MVL functions, a program has been written to determine how the required number of product terms vary for the different sets of operators. As it is computationally expensive to test the program for all 4-valued 2-variable functions (i.e., \(4^{(4^2)} = 4,294,967,296\)), the program has been tested for 3-valued 2-variable MVL functions and the results are reported. Realizations of a 4-valued 2-variable MVL function based on the above three sets of operators are discussed as an example. The HSPICE transient analysis simulation results of the realization based on the proposed set of operators are included.

This chapter consists of four sections. In Section 5.1, realizations of MVL functions are discussed. In Section 5.2, realizations based on different set of operators for an example MVL function are discussed. For 3-valued 2-variable functions, results of realizations based on the above three sets of operators are summarized in Section 5.3. The chapter summary is given in Section 5.4.
5.1 Realization of MVL Functions

Multiple-valued logic (MVL) functions, like binary functions, can be represented in sum-of-product (SOP) form. However, unlike binary logic, there have been many different forms proposed for realizing individual product terms (PTs) and summing operators [5, 6, 38, 55]. It was shown in Chapter 2 that PTs can be realized in different ways depending on which of the unary operators (literal, cycle, complement of literal, and complement of cycle) are used. For a given set of unary operators, it is possible to have different realizations of PTs depending on which combining operation is used, namely max, tsum, or modsum.

In general, a MVL function \( f(x_1, x_2, \ldots, x_n) \) can be expressed in SOP form as:

\[
f(x_1, x_2, \ldots, x_n) = P_1 <\text{sumop}> P_2 <\text{sumop}> \cdots <\text{sumop}> P_l
\]

\[
= <\text{sumop}> (P_1, P_2, \ldots, P_l),
\]

(5.1)

where \( P_i \)'s represent PTs and \(<\text{sumop}>\) is either the max (\(\bigvee\)), the tsum (\(\bigoplus\)), or the modsum (\(\bigodot\)) operator. In order to understand different realizations, consider the realization of the 4-valued 2-variable function, \( f_1(x_1, x_2) \) shown in Figure 5.1.

![Figure 5.1](image)  

**Figure 5.1** An example MVL function to show use of different combining operators in SOP form realization.
Assuming that the literal operator is used to realize a PT, the three realizations based on \textit{max}, \textit{tsum}, and \textit{modsum} operators are:

\textbf{Max:}
\begin{equation}
    f_1(x_1, x_2) = (1[0](x_1^2)) \bullet (1[0](x_2^1)) \odot (1[1](x_1^3)) \bullet (1[0](x_2^2)) \\
    \odot (1[1](x_2^1)) \bullet (1[0](x_2^2)) \odot (2[0](x_1^2)) \bullet (2[1](x_2^1)) \\
    \odot (3[1](x_1^1)) \bullet (3[1](x_2^1)),
\end{equation}

\textbf{Tsum:}
\begin{equation}
    f_1(x_1, x_2) = (1[0](x_1^2)) \bullet (1[0](x_2^1)) \boxplus (1[1](x_1^3)) \bullet (1[1](x_2^1)) \boxplus \\
    (1[0](x_2^2)) \bullet (1[1](x_2^1)) \boxplus (1[1](x_1^2)) \bullet (1[3](x_2^3)) \boxplus \\
    (1[1](x_1^1)) \bullet (1[1](x_2^1)),
\end{equation}

\textbf{Modsum:}
\begin{equation}
    f_1(x_1, x_2) = (1[1](x_1^2)) \ominus (1[1](x_2^1)) \oplus (1[0](x_1^2)) \bullet (1[0](x_2^2)) \\
    \ominus (1[1](x_1^1)) \bullet (1[1](x_2^1)) \ominus (3[1](x_1^2)) \bullet (3[1](x_2^2)).
\end{equation}

The map representation for each PT for \textit{max}, \textit{tsum} and \textit{modsum} realizations are shown in Figure 5.2, Figure 5.3, and Figure 5.4, respectively.

From the above example, it is seen that the required number of PTs vary depending on which operator is used to combine the terms. A similar example has been reported by Dueck and Miller [26] to show realizations of a MVL function based on different operators used for realizing PTs and combining operator. There are many possible combinations of operations that can result in different set of operators. Allen and Givone [33] had proposed a set consisting of \textit{min}, \textit{max}, and \textit{window literal} operators. Vranesic et al. [35] discussed MVL function realizations based of \textit{min}, \textit{max}, \textit{cycle}, and \textit{unary inverter} operators. McCluskey [36] and Kerkhoff [37] used \textit{min}, \textit{tsum}, and \textit{window literal} operators for function realizations in PL and CCD, respectively. They used the \textit{tsum} operator instead of the \textit{max} operator because it is much simpler to realize the \textit{tsum} operator in these technologies as compared to the \textit{max} operator. This indicates that, the effective use of different set of operators is technology dependent.
Figure 5.2 Product terms for realization of the function, $f_1(x_1, x_2)$, in SOP form using max operator.

Figure 5.3 Product terms for realization of the function, $f_1(x_1, x_2)$, in SOP form using tsum operator.
Figure 5.4 Product terms for realization of the function, $f_1(x_1, x_2)$, in SOP form using modsum operator.

It was shown in Chapter 4, that the cost (in terms of number of transistors) of realizing the literal or the complement of literal is the same, and that this cost is comparable with that of the cycle operator. Thus, the use of the complement of literal and the cycle operator can be as advantageous as the use of the literal operator for PTs realization. Also, for current-mode CMOS, like $I^2L$ and CCD technologies, the use of the tsun operator is more effective as compared to max and modsum operators.

In this chapter, MVL function realization based on the set of operators proposed in Chapter 4 is discussed. In order to see the usefulness of the proposed set of operators, realization results have been compared to the realizations using two existing sets of operators. It will be shown how the choice of operators affects the number of PTs required to realize a function. The three sets of operators considered are:

1. **Set 1**: literal, cycle, complement of literal, complement of cycle, min, and tsum.
2. **Set 2**: literal, complement of literal, min, and tsum.
3. **Set 3**: literal, min, and tsum.
Set 1 represents the proposed set of operators. Set 2 represents the set of operators proposed by Su and Cheung [34] where max operator has been replaced by tsum operator. Similarly, Set 3 represents Allen and Givone's [33] set of operators with the max operator is replaced by tsum operator.

5.2 Example Realization

A given MVL function can be realized as:

\[ f(x_1, x_2, \ldots, x_n) = P_1 \sqcup P_2 \sqcup \ldots \sqcup P_l \]

\[ = \text{tsum}(P_1, P_2, \ldots, P_l) \quad (5.5) \]

where \( P_i \)'s represent PTs. Consider the realization of the 4-valued 2-variable MVL function, \( f_2(x_1, x_2) \) shown in Figure 5.5. Using Set 1, the function can be expressed as a single PT (see Figure 5.6) as follows:

\[ f_2(x_1, x_2) = (2^{[1, 2]}_1) \cdot (x_2^3). \quad (5.6) \]

The circuit realization in terms of the proposed set of operators and the basic circuit elements of CMCL is shown in Figure 5.7. For detailed realization of each block see.
Figure 5.6 Map of PT required for realizing the function \( f_2(x_1, x_2) \) using Set 1.

Section 4.2.

If Set 2 is used to realize \( f_2(x_1, x_2) \) then 2 PTs are required (see Figure 5.8), as follows:

\[
f_2(x_1, x_2) = (2[x_0 1]) \cdot (2[x_2 2]) \cdot (1[x_1 1]) \cdot (1[x_2 2]).
\]  

(5.7)

Four literals are needed in order to realize the above expression. However, the PT \((1[x_1 1]) \cdot (1[x_2 2])\) is equal to \((2[x_1 1]) \cdot (1[x_2 2])\). To reduce the required circuitry, the function \( f_2(x_1, x_2) \) can be rewritten as:

\[
f_2(x_1, x_2) = (2[x_1 1]) \cdot (2[x_2 2]) \cdot (2[x_1 1]) \cdot (1[x_2 2]).
\]  

(5.8)
In the above expression, two complement of literal and one literal operators are needed. These are $2^{[1][x_1]^0}$, $2^{[1][x_2]^3}$, and $1^{[2][x_2]^2}$. The circuit realization for the above expression is shown in Figure 5.9.

Using Set 3 it is necessary to realize 6 PTs for the function $f_2(x_1, x_2)$ as shown below:

$$f_2(x_1, x_2) = (2^{[0][x_1]^0}) \cdot (2^{[0][x_2]^0}) \oplus (2^{[2][x_1]^3}) \cdot (2^{[0][x_2]^0}) \oplus (1^{[0][x_1]^0}) \cdot (1^{[2][x_2]^2})$$

$$\oplus (1^{[2][x_1]^3}) \cdot (1^{[2][x_2]^2}) \oplus (2^{[0][x_1]^0}) \cdot (2^{[3][x_1]^3}) \oplus (2^{[2][x_1]^3}) \cdot (2^{[3][x_2]^3})$$

(5.9)

PTs in map form are shown in Figure 5.10. In order to realize the function, it is necessary to realize 7 literals, namely $2^{[0][x_1]^0}$, $2^{[2][x_1]^3}$, $1^{[0][x_1]^0}$, $1^{[2][x_1]^3}$, $2^{[0][x_2]^0}$, $1^{[2][x_2]^2}$, and $2^{[3][x_2]^3}$. As mentioned before, there are many equivalent representations for a PT. In the above example,

$$(1^{[0][x_1]^0}) \cdot (1^{[2][x_2]^2}) = (2^{[2][x_1]^3}) \cdot (1^{[2][x_2]^2})$$, and

$$(1^{[2][x_1]^3}) \cdot (1^{[2][x_2]^2}) = (2^{[2][x_1]^3}) \cdot (1^{[2][x_2]^2})$$

(5.10)
Thus, the SOP form for \( f_2(x_1, x_2) \) can be rewritten as:

\[
f_2(x_1, x_2) = (2^{[\neg x_1]^0}) \cdot (2^{[\neg x_2]^0}) \oplus (2^{[\neg x_1]^0}) \cdot (2^{[\neg x_2]^0}) \oplus (2^{[\neg x_1]^0}) \cdot (1^{[\neg x_2]^1}) \\
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \oplus (2^{[\neg x_1]^1}) \cdot (1^{[\neg x_2]^1}) \oplus (2^{[\neg x_1]^0}) \cdot (2^{[\neg x_2]^1}) \oplus (2^{[\neg x_1]^0}) \cdot (2^{[\neg x_2]^1}),
\]

(5.11)

which requires only 5 literals, namely \( 2^{[\neg x_1]^0}, 2^{[\neg x_2]^0}, 2^{[\neg x_1]^1}, 1^{[\neg x_2]^1}, \) and \( 2^{[\neg x_2]^1}. \)

The corresponding circuit realization is shown in Figure 5.11.

Simulation results for the circuit realization shown in Figure 5.7 has only been included. As discussed previously, in order to implement the circuit it is necessary to realize \( 2^{[\neg x_1]^0} \) and \( x_2^3 \) operators. The complement of literal, \( 2^{[\neg x_1]^0} \), can be realized using the circuit shown in Chapter 4 (see Figure 4.7). The size of transistors \( T_1, T_2, \) and \( T_3 \) being \( W_1/L_1 = 0.5, W_2/L_2 = 1.5, \) and \( W_3/L_3 = 2, \) respectively. For the cycle operator it is known that \( x_2^3 = x_2^1 \) (see Section 4.2.4), so \( x_2^3 \) can be realized as \( x_2^1 \) using the cycle operator circuit shown in Chapter 4, see Figure 4.8. The size of transistors \( T_1, T_2, \) and \( T_3 \) are \( W_1/L_1 = 0.5, W_2/L_2 = 3, \) and \( W_3/L_3 = 1, \) respectively. The simulation results for the realization of the MVL function \( f_2(x_1, x_2) \) are shown in Figure 5.12, which consists of 3 panels. All the panels show current level variations as
Figure 5.10 PTs required for realizing the function $f_2(x_1, x_2)$ using Set 3.

a function of time. Panel 1 and Panel 2 show all possible combinations of the input variables, $x_1$ and $x_2$, respectively, while Panel 3 shows the function $f_2(x_1, x_2)$. It can be easily verified from the figure that Panel 3 represents the expected behaviour of the function. Propagation delays, dependent on the current levels switched between, are between 10.03 ns and 19.52 ns with an average delay of 13.43 ns.

CMCL is not a self restoring logic and there is some deterioration of MVL current levels as signals propagate through various CMCL circuit elements. These MVL signals may be added or subtracted (such as in the realization of min operator), so the signal deterioration may be compounded and result in an incorrect logic level. Therefore, it may be necessary to restore the MVL current values to proper levels. It can be noticed that the output logic levels (in Figure 5.12) are not as distinct as the input logic levels. In a typical application the output will feed a threshold circuit to detect the correct logic level. In order to show that the outputs are properly evaluated by the next stage circuit, the level restorer circuit, discussed in Section 4.2.5, is used at the output. The output of the circuit shown in Figure 5.7 is the input to the level restorer circuit. The corresponding simulation results are shown in Figure 5.13. It can be seen that the output levels are restored to the proper levels.
For certain MVL functions, it is advantageous to realize the complement of the function instead of the original function. Consider the realization of the 4-valued 2-variable function, $f_3(x_1, x_2)$ shown in Figure 5.14, as an example. Using the first set of operators (Set 1), the function can be expressed in SOP form as:

$$f_3(x_1, x_2) = (x_2^3) \oplus (1[x_3^0]) \oplus (2[i_1]) \cdot (x_2^3).$$  \hspace{1cm} (5.12)

The expression consists of 3 PTs. In order to realize the function it is necessary to realize 2 cycle and 2 literal operators. However, the above function can also be represented as:

$$f_3(x_1, x_2) = f_2(x_1, x_2) = (2[i_1^1]) \cdot (x_2^3),$$  \hspace{1cm} (5.13)

where it is necessary to realize only one PT consisting of a cycle operator and a complement of literal operator. The realization of this function is the same as shown in Figure 5.7, except that at the output the logic values are subtracted from $(r - 1)$
Figure 5.12 HSPICE output for the transient analysis of the circuit realization of \( f_2(x_1, x_2) \) using Set 1.
Figure 5.13 HSPICE output for the transient analysis of the circuit realization of $f_2(x_1, x_2)$ using Set 1 and a level restorer circuit.
Figure 5.14 An example 4-valued 2-variable MVL function, which can be realized as the complement function using less number of PTs.

(i.e., 3 for the above example). It is seen that by realizing the complement of a MVL function, one can reduce the number of PTs required to realize the function. It should be noted that this is only applicable for some MVL functions. For example, the number of PTs required to realize the function $f_2(x_1, x_2)$ (see Figure 5.5) as a complement is more than those required to realize the original function.

5.3 Comparison between Different Sets of Operators

In the previous section, it has been shown with an example that the use of the set of operators consisting of literal, cycle, complement of literal, complement of cycle, min, and tsum operators can be useful in reducing the number of PTs for the realization of some functions. For the reported example, the reduction is from 6 PTs to 1 PT, i.e., 86% when compared to the realization based on the set consisting of literal, min, and tsum operators. In order to determine the usefulness of the new set of operators for a given set of $r$-valued $n$-variable functions, it is necessary to determine the number of functions for which the number of PTs required for realizing the function decreases using the Set 1 operators as compared to those required by realizations based on Set 2 and Set 3 operators.
As discussed previously, a given MVL function can be decomposed into product terms and realized as truncated sum of those PTs. In addition, it has been shown that there are a number of possible realizations for a PT, depending on the set of operators used for realizing the PTs. For \( r \) -valued \( n \) -variable functions, there is a fixed number of possible PTs for a given set of operators. Therefore, there is a restricted set of PTs into which a given MVL function can be decomposed. These product terms can be referred to as prime decompositions (PDs). A program has been written for \( r \) -valued \( n \) -variable functions to determine the minimum number of PTs required to realize MVL functions using a given set of operators. In addition, it is possible, using this program, to allow the realization of the function as a complement of some other MVL function by setting/resetting the complement flag. The program is incremental in nature. It starts by obtaining a list of functions that can be realized as a single PT, i.e., all the functions corresponding to the PDs. If the complement flag of function is set, then a list of functions is generated by taking the complement of the functions realized as a single PT. This is followed by the generation of a list of functions that can be realized in terms of 2 PDs. If the complement of function is allowed, then new functions, which can be realized as the complement of the functions realized as the tsum of 2 PDs, are added to the list. The procedure continues for higher number of PDs until all the possible functions (called total functions in the program) are realized. The pseudo-code of the program is given in Figure 5.15.

Considering 3-valued 2-variable functions, there are 19683 different possible functions. For the proposed set of operators, which consists of the literal, the cycle, the complement of literal, the complement of cycle, min, and tsum operators, there are 219 possible PDs. These PDs can be divided into 10 categories depending on how a PT is realized. Table 5.1 shows a typical example and the number of PDs in each category. The applicable categories and total number of PDs for individual set of operators (i.e., Set 1, Set 2, and Set 3) are shown in Table 5.2.

The program has been tested for all the three sets of operators with and without the complement of function allowed. The results are summarized in Table 5.3 when
Variables
PD_LIST is the list of all the prime decompositions.
TOTAL_PD is the number of PDs.
FUNC[i] is the list of functions realized as the tsum of i PDs.
    (note: FUNC[1]=PD_LIST)
NUM[i] is the number of functions realized as the tsum of i PDs.
FUNC_COUNT is the number of functions realized so far.
TOTAL_FUNCTIONS is the total number of functions possible.
COMP_FLAG is the flag which tells whether the complement of the function
    is to be tried or not.
num_of_pds is the count of number of PDs needed to realize functions.

Initialize
num_of_pds ← 1
FUNC_COUNT ← TOTAL_PD
mark all the functions corresponding to PDs as realized.
if(COMP_FLAG is set)
    for each PD in PD_LIST
        newfunc=complement of the PD
        if(newfunc not marked realized)
            add 1 to FUNC_COUNT;
            mark the newfunc as realized
while(FUNC_COUNT is less than TOTAL_FUNCTIONS)
    new_func_count ← 0
    for i=0 to i= NUM[num_of_pds]
        for j=0 to j= TOTAL_PD
            newfunc=tsum(FUNC[num_of_pds][i],FUNC[1][j])
            if(newfunc not marked realized)
                add newfunc to FUNC[num_of_pds+1][new_func_count]
                add 1 to new_func_count
                mark the newfunc as realized
        add 1 to num_of_pds
    NUM[num_of_pds] ← new_func_count
    add new_func_count to FUNC_COUNT
if(COMP_FLAG is set)
    for each function, f, in the list FUNC[num_of_pds]
        newfunc = complement of f
        if(newfunc not marked realized)
            add 1 to FUNC_COUNT
            mark the newfunc as realized

Figure 5.15 Pseudo-code of the program for finding the number of
PTs required to realize r-valued n-variable functions for
a given set of operators.
Table 5.1  Number of PDs in each category for 3-valued 2-variable functions.

<table>
<thead>
<tr>
<th>Category Number</th>
<th>Choices for X₁ and X₂ operators in a PD</th>
<th>Typical Example</th>
<th>Number of PDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>both literal</td>
<td>( (2\overline{x_1}) \cdot (1\overline{x_2}) )</td>
<td>73</td>
</tr>
<tr>
<td>II</td>
<td>literal and complement of literal</td>
<td>( (1\overline{x_1}) \cdot (1\overline{x_2}) )</td>
<td>24</td>
</tr>
<tr>
<td>III</td>
<td>both complement of literal</td>
<td>( (2\overline{x_1}) \cdot (2\overline{x_2}) )</td>
<td>2</td>
</tr>
<tr>
<td>IV</td>
<td>cycle and literal</td>
<td>( x_1 \cdot (3\overline{x_2}) )</td>
<td>36</td>
</tr>
<tr>
<td>V</td>
<td>cycle and complement of literal</td>
<td>( (2\overline{x_1}) \cdot (x_2^3) )</td>
<td>6</td>
</tr>
<tr>
<td>VI</td>
<td>both cycle</td>
<td>( x_1^2 \cdot (x_2^3) )</td>
<td>9</td>
</tr>
<tr>
<td>VII</td>
<td>complement of cycle and literal</td>
<td>( (3\overline{x_1}) \cdot (3\overline{x_2}) )</td>
<td>36</td>
</tr>
<tr>
<td>VIII</td>
<td>complement of literal and complement of cycle</td>
<td>( (3\overline{x_1}) \cdot (x_2^2) )</td>
<td>6</td>
</tr>
<tr>
<td>IX</td>
<td>cycle and complement of cycle</td>
<td>( (\overline{x_1}) \cdot (x_2^1) )</td>
<td>18</td>
</tr>
<tr>
<td>X</td>
<td>both complement of cycle</td>
<td>( (x_1^2) \cdot (x_2^3) )</td>
<td>9</td>
</tr>
</tbody>
</table>

The complement of function is not allowed, and in Table 5.4 when the complement of function is allowed. The average number of PDs required for all the three set of operators are summarized in Table 5.5.

From the above results, it can be seen that by using Set 1 operators, it is possible to reduce the maximum number of PTs needed to realize all 3-valued 2-variable functions from 6 PTs, using Set 3, and 5 PTs, using Set 2, to 3 PTs using Set 1 without using the complement of functions. When the complement of function is allowed, it was found that the maximum number of PTs required to realize all 3-

Table 5.2 Number of PDs for different set of operators in 3-valued 2-variable functions.

<table>
<thead>
<tr>
<th>Set of Operators</th>
<th>PD Categories</th>
<th>No. of PDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textbf{Set 1}: literal, cycle, complement of literal, complement of cycle, min, and tsum</td>
<td>all the ten categories (I to X)</td>
<td>219</td>
</tr>
<tr>
<td>\textbf{Set 2}: literal, complement of literal, min, and tsum</td>
<td>I II, and III</td>
<td>99</td>
</tr>
<tr>
<td>\textbf{Set 3}: literal min, and tsum</td>
<td>only I</td>
<td>73</td>
</tr>
</tbody>
</table>
Table 5.3  Distribution of number of PDs and number of functions, for 3-valued 2-variable functions, using three set of operators when the complement of function is not allowed.

<table>
<thead>
<tr>
<th>Number of PDs</th>
<th>Number of Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Set 1</td>
</tr>
<tr>
<td>1</td>
<td>219</td>
</tr>
<tr>
<td>2</td>
<td>7320</td>
</tr>
<tr>
<td>3</td>
<td>12144</td>
</tr>
<tr>
<td>4</td>
<td>6162</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>19683</td>
</tr>
</tbody>
</table>

valued 2-variable functions is reduced from 5 PTs using Set 3 or Set 2 to 3 PTs using Set 1. In addition, there is an approximately 27% reduction in the average number of PDs required to realize all the possible functions when Set 1 operators are used as compared to those required using Set 3 operators. The improvements achieved by using Set 1 operators is not restricted to 3-valued 2-variable functions, as shown earlier (see Section 5.2) and as will be seen in Chapter 7.

Table 5.4  Distribution of number of PDs and number of functions, for 3-valued 2-variable functions, using three set of operators when the complement of function is allowed.

<table>
<thead>
<tr>
<th>Number of PDs</th>
<th>Number of Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Set 1</td>
</tr>
<tr>
<td>1</td>
<td>219</td>
</tr>
<tr>
<td>2</td>
<td>7200</td>
</tr>
<tr>
<td>3</td>
<td>8238</td>
</tr>
<tr>
<td>4</td>
<td>2466</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>15657</td>
</tr>
</tbody>
</table>

\( f \): function realized in original form.

\( \overline{f} \): function realized as complement of some other function.
Table 5.5  Average number of PDs, for 3-valued 2-variable function, using three set of operators when the complement of function is not allowed and when it is allowed.

<table>
<thead>
<tr>
<th>Operators</th>
<th>Average number of PDs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Complement of function not allowed</td>
<td>Complement of function allowed</td>
</tr>
<tr>
<td>Set 1</td>
<td>2.61</td>
<td>2.39</td>
</tr>
<tr>
<td>Set 2</td>
<td>3.19</td>
<td>2.91</td>
</tr>
<tr>
<td>Set 3</td>
<td>3.61</td>
<td>3.27</td>
</tr>
</tbody>
</table>

5.4 Summary

The performance of the set of operators proposed in Chapter 4 has been compared with two existing set of operators for the realization of MVL functions. It has been shown in Chapter 4 that the costs of realizing the new set of operators and the existing set of operators are comparable. The realizations of a 4-valued 2-variable function using different set of operators have been included to show that by using the new set of operators it is possible to reduce the number of product terms required to realize a given MVL function in SOP form. It has also been shown that for some MVL functions it is advantageous to realize the complement of the function.

In order to see the performance of the different set of operators over a large sample of MVL functions, the number of PTs required in the realization of all 19683 3-valued 2-variable functions have been included. For the proposed set of operators (i.e., Set 1), the average number of PTs required to realize all functions is reduced as compared to realizations based on Set 2 and Set 3. In addition, there is a substantial reduction in the maximum number of PTs required to realize the functions using the proposed set of operators.
Like binary logic functions, MVL functions can be realized in SOP form using programmable logic arrays (PLA). In PLAs, each PT is realized as one column, thus the number of required PTs is referred to as the cost of realization. In order to design area efficient digital circuits, it is necessary to reduce the number of PTs in order to minimize the total cost of realization. In this chapter two new approaches are considered for efficient realization of MVL functions. These function realizations are based on the difference-of-sum-of-products (DOSOP) form and the sum-of-terms (SOT) form. Using the new approaches, the number of required terms can be reduced in the realization of some MVL functions as compared to those obtained using the SOP form.

This chapter consists of three sections. In Section 6.1, MVL function realization based on the difference-of-sum-of-products terms is discussed along with some examples. Section 6.2 deals with a discussion on the sum-of-terms realization for MVL functions. Some illustrative examples are also included in this section. The chapter summary is given in Section 6.3.

6.1 Difference-of-Sum-of-Products Realization

In Chapter 5, it was shown how the use of different sets of operators affects the number of required PTs using the SOP realization of MVL functions. For certain MVL functions, it is possible to further reduce the required number of PTs using the difference-of-sum-of-products form realization.
A MVL function $f(x_1, x_2, \ldots, x_n)$ can be expressed in a difference-of-sum-of-products form as [68]:

$$f(x_1, x_2, \ldots, x_n) = (P_1 \oplus P_2 \oplus \cdots \oplus P_l) - (Q_1 \oplus Q_2 \oplus \cdots \oplus Q_k) \quad (6.1)$$

where $P_i$'s and $Q_i$'s represent PTs and

$$(P_1 \oplus P_2 \oplus \cdots \oplus P_l) \geq (Q_1 \oplus Q_2 \oplus \cdots \oplus Q_k)$$

always hold true. It is advantageous to use DOSOP form iff $l + k < m$, where $m$ is the number of PTs required for realization based on SOP form using the same set of operators as used for DOSOP form realization.

Using the subtraction operation it is possible to partition the original function into two sub-functions, the difference of which forms the original function. First sub-function is realized in sum-of-product form which is minuend in the difference (subtraction) operation. The sum-of-product realization of the other function forms the subtrahend of the difference operation. Consider, for example, a 4-valued 2-variable function, $f_4(x_1, x_2)$ for which the map representation is shown in Figure 6.1. Using Set 1 operators, discussed in Chapter 5, 5 PTs are required for the SOP form realization.

SOP form:

$$f_4(x_1, x_2) = (2\{x_1^1x_1^3\}) \cdot (2\{x_2^1\}) \oplus (2\{x_1^1x_1^3\}) \cdot (2\{x_2^1x_2^3\}) \oplus \cdots$$

$$\quad (2\{x_1^1\}) \cdot (2\{x_2^1\}) \oplus (2\{x_1^1x_1^3\}) \cdot (2\{x_2^1x_2^3\}) \oplus (1\{x_1^3\}) \oplus (1\{x_2^3\}) \quad (6.2)$$

However, with the new approach (DOSOP), the function shown in Figure 6.1 can be partitioned as shown in Figure 6.2. Only 2 PTs are needed to realize the function using this approach.
DOSOP form:
\[
f_4(x_1, x_2) = f_p(x_1, x_2) - f_n(x_1, x_2) = (2^{1 \cdot x_1} \cdot 2^{1 \cdot x_2}) - (1^{2 \cdot x_1^2} \cdot 1^{2 \cdot x_2^2})
\] (6.3)

There are two structures that can be used for implementing MVL functions using the DOSOP approach. These structures are called as Structure S1 and Structure S2. The block diagram for these structures are shown in Figure 6.3 and Figure 6.4, respectively. In S1 both the sub-function outputs are truncated before subtraction. However, for S2 truncation is carried out after subtraction. The number of PTs required to realize a function using these structures could be different. For example, consider the 4-valued 2-variable example function shown in Figure 6.5. Using the existing approach (SOP), 8 PTs are needed (see Figure 6.6 for a possible realization). For the DOSOP form, using S1 needs only 3 PTs and with S2 at least 4 PTs are required, as shown in Figure 6.7 and Figure 6.8, respectively. In the following discussion, only implementations using the structure S1 for the realization of MVL functions in the DOSOP form are considered.

\[
\begin{array}{cccc}
  & x_1 & x_2 & f_4(x_1, x_2) \\
 0 & & & \\
 1 & & & \\
 2 & & & \\
 3 & & & \\
\end{array}
\]

**Figure 6.1** An example MVL function to illustrate realization based on the DOSOP form.
The suitability of DOSOP approach depends on identification of patterns for which there are some possible improvements in terms of the number of PTs needed. Many 4-valued 2-variable example functions have been tried to identify patterns for which the use of DOSOP would need fewer PTs as compared to SOP realization. Some of the patterns for which there is possible improvement using the DOSOP form of realization are considered in this section. For each pattern an example function in also discussed.

Pattern 1:

where \( a > b \), \( a \in \{2, 3\} \), and \( b \in \{1, 2\} \). Using Set 1 operators, discussed in Section 5.1 and Section 5.2, 3 PTs are required using the SOP form and only 2 PTs are needed using the DOSOP form realization. For example, consider a 4-valued 2-variable MVL function with \( a = 3 \) and \( b = 1 \). The realizations using Set 1 operators are:

![Figure 6.2 Partition of the example function, shown in Figure 6.1, for realization in the DOSOP form.](image)
Figure 6.3 Structure $S_1$ for MVL function realization in the DOSOP form.

Figure 6.4 Structure $S_2$ for MVL function realization in the DOSOP form.
Figure 6.5 An example MVL function to illustrate realization based on the DOSOP form using different structures for implementation.

\[ f_5(x_1, x_2) = (1^{[1]}(x_1)) \cdot (1^{[2]}(x_2)) \uplus (1^{[2]}(x_1)) \cdot (1^{[1]}(x_2)) \uplus \\ (1^{[1]}(x_1)) \cdot (1^{[2]}(x_2)) \uplus (1^{[2]}(x_1)) \cdot (1^{[1]}(x_2)) \uplus \\ (3^{[0]}(x_1)) \cdot (3^{[2]}(x_2)) \uplus (3^{[2]}(x_1)) \cdot (3^{[0]}(x_2)) \uplus \\ (3^{[0]}(x_1)) \cdot (3^{[2]}(x_2)) \uplus (3^{[2]}(x_1)) \cdot (3^{[0]}(x_2)) \]

Figure 6.6 PTs for the SOP form realization of example function shown in Figure 6.5.
\[
f_5(x_1, x_2) = f_p(x_1, x_2) - f_n(x_1, x_2)
\]
\[
= (3^0(x_1^2)) \cdot (3^0(x_2^2)) - (2^1(x_1^1)) \cdot (2^0(x_2^2))
\]

Figure 6.7 PTs for realization based on the DOSOP form using structure \( S_1 \), for example function shown in Figure 6.5.

\[
f_5(x_1, x_2) = f_p(x_1, x_2) - f_n(x_1, x_2)
\]
\[
= (3^0(x_1^2)) \cdot (3^0(x_2^2)) - (2^1(x_1^1)) \cdot (2^0(x_2^2))
\]

Figure 6.8 PTs for realization based on the DOSOP form using structure \( S_2 \), for example function shown in Figure 6.5.
SOP form:

\[ f(x_1, x_2) = (1[x_1^0]) \cdot (1[x_2^1]) \oplus (3[x_1^0]) \cdot (3[x_2^0]) \oplus (3[x_1^0]) \cdot (3[x_2^1]) \]

DOSOP form:

\[ f_p(x_1, x_2) = (3[x_1^0]) \cdot (3[x_2^1]) \]
\[ f_n(x_1, x_2) = (2[x_1^0]) \cdot (2[x_2^1]) \]
\[ f(x_1, x_2) = f_p(x_1, x_2) - f_n(x_1, x_2) = (3[x_1^0]) \cdot (3[x_2^1]) - (2[x_1^0]) \cdot (2[x_2^1]). \]

Pattern 2:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

where \( a > b, a \in \{2, 3\}, \text{ and } b \in \{1, 2\} \). Using Set 1 operators, 5 PTs are required using the SOP form and only 4 PTs are needed using the DOSOP form realization. For example, consider a 4-valued 2-variable MVL function with \( a = 3 \) and \( b = 2 \). The realizations using Set 1 operators are:

SOP form:

\[ f(x_1, x_2) = (2[x_1^0]) \cdot (2[x_2^0]) \oplus (2[x_1^1]) \cdot (2[x_2^1]) \oplus (3[x_1^0]) \cdot (3[x_2^0]) \oplus (3[x_1^1]) \cdot (3[x_2^1]) \oplus (3[x_1^0]) \cdot (3[x_2^1]), \]

(6.6)
DOSOP form:

\[
\begin{align*}
 f_p(x_1, x_2) &= (3^{[0](x_1^p)}) \cdot (3^{[0](x_2^p)}) \\
 f_n(x_1, x_2) &= (1^{[1](x_1^p)}) \cdot (1^{[0](x_2^p)}) \oplus (1^{[2](x_1^p)}) \cdot (1^{[2](x_2^p)}) \oplus \\
 f(x_1, x_2) &= f_p(x_1, x_2) - f_n(x_1, x_2) \\
 &= (3^{[0](x_1^p)}) \cdot (3^{[0](x_2^p)}) - ((1^{[1](x_1^p)}) \cdot (1^{[0](x_2^p)})) \oplus \\
 &\quad (1^{[2](x_1^p)}) \cdot (1^{[2](x_2^p)}) \oplus (3^{[1](x_1^p)}) \cdot (3^{[1](x_2^p)})).
\end{align*}
\] (6.7)

Pattern 3:

\[
\begin{array}{c|cccc}
 x_2 & 0 & 1 & 2 & 3 \\
 \hline
 x_1 \ & 0 & a & a & a \\
 & 1 & a & b & a \\
 & 2 & a & a & a \\
 & 3 & & & \\
\end{array}
\]

where \( a > b \), \( a \in \{2, 3\} \), and \( b \in \{1, 2\} \). Using Set 1 operators, 5 PTs are required using the SOP form and only 2 PTs are needed using the DOSOP form realization. As an illustrative example, consider the realization of MVL function shown in Figure 6.1. The figure represents the above pattern but for different values of \( x_1 \) and \( x_2 \). The SOP and DOSOP form realizations are given by Equation 6.2 and Equation 6.3, respectively.

Pattern 4:

\[
\begin{array}{c|cccc}
 x_2 & 0 & 1 & 2 & 3 \\
 \hline
 x_1 \ & 0 & a & b & a \\
 & 1 & b & & \\
 & 2 & a & b & a \\
 & 3 & & & \\
\end{array}
\]
where \( a > b, a \in \{2, 3\}, \) and \( b \in \{1, 2\} \). Using Set 1 operators, 8 PTs are required for the SOP form and for the DOSOP form realization 3 or 4 PTs are needed, depending on the values of \( a \) and \( b \). For example, for \( a = 3 \) and \( b = 1 \), the SOP form realization needs 8 PTs and for the DOSOP form realization 3 PTs are required, as shown in Figure 6.6 and Figure 6.7, respectively. However, for \( a = 3 \) and \( b = 2 \), realization based on the DOSOP form requires 4 PTs.

It should be noted that some of these patterns could have different orientations in a map representation. Similarly, it is possible to combine some patterns to obtain new patterns. The list of patterns reported in this thesis is by no means exhaustive. There are some more patterns that have been observed and there may be many more. The study was restricted to 4-valued 2-variable functions. However, the new approach is suitable for more number of variables and higher valued logic functions. For example, consider the 5-valued 2-variable MVL function shown in Figure 6.9. Realization based on the SOP form requires a minimum of 11 PTs whereas only 4 PTs are required using the DOSOP form realization, as shown below:

Figure 6.9 An example 5-valued 2-variable function for illustrating the DOSOP form realization.
SOP form:

\[
f_6(x_1, x_2) = (1^0[x_1]) \bullet (1^1[x_2]) \boxplus (1^2[x_1]) \bullet (1^3[x_2]) \boxplus \\
(1^1[x_1]) \bullet (1^2[x_2]) \boxplus (1^3[x_1]) \bullet (1^4[x_2]) \boxplus \\
(2^0[x_1]) \bullet (2^1[x_2]) \boxplus (2^2[x_1]) \bullet (2^3[x_2]) \boxplus \\
(2^1[x_1]) \bullet (2^2[x_2]) \boxplus (2^3[x_1]) \bullet (2^4[x_2]) \boxplus \\
(2^2[x_1]) \bullet (2^3[x_2]) \boxplus (2^4[x_1]) \bullet (2^5[x_2]),
\]

DOSOP form:

\[
f_p(x_1, x_2) = (3^0[x_2]) \\
f_n(x_1, x_2) = (2^1[x_1]) \bullet (2^2[x_2]) \boxplus (2^3[x_1]) \bullet (2^4[x_2]) \boxplus \\
(2^2[x_1]) \\
f_6(x_1, x_2) = f_p(x_1, x_2) - f_n(x_1, x_2) \\
= (3^0[x_2]) - ((2^1[x_1]) \bullet (2^2[x_2]) \boxplus (2^3[x_1]) \bullet (2^4[x_2])).
\]

6.1.1 An Example

Consider the realization of function shown in Figure 6.1 in the DOSOP form. Only two PTs are required to realize this function as shown in Equation 6.3. The realization based on structure S_1 is shown in Figure 6.10. The circuit was simulated using HSPICE transient analysis. Simulations results are summarized in Figure 6.11, which consists of 5 panels. All the panels show current level variation as a function of time. Panel 1 and Panel 2 show all the possible combinations for the input variables, \(x_1\) and \(x_2\). The partitioned functions \(f_p(x_1, x_2)\) and \(f_n(x_1, x_2)\) are shown in Panel 3 and Panel 4, respectively. Panel 5 shows the output, \(f_6(x_1, x_2)\) (i.e., \(f_p(x_1, x_2) - f_n(x_1, x_2)\)). It can be verified from simulation results, that the output is in correspondence with the expected logic levels.
6.2 Sum-of-Terms Realization

In the previous section, it was shown that for some MVL functions the DOSOP form realization requires fewer PTs as compared to the SOP form realization. In this section, a new approach, sum-of-terms form is considered. This realization is based on the general structure discussed in Section 4.1 (see Figure 4.1).

A MVL function \( f(x_1, x_2, \ldots, x_n) \) can be expressed in a sum-of-terms (SOT) form as [63]:

\[
    f(x_1, x_2, \ldots, x_n) = T_1 \ominus T_2 \ominus \cdots \ominus T_l
\]

(6.10)

where \( T_i \in \{ P_i, Q_i, R_i \} \) (see Section 2.2). Consider, for example, the 4-valued 2-variable function, \( f_7(x_1, x_2) \) shown in Figure 6.12.

Using the SOP form realization (see Figure 6.13), a total of 3 PTs are required.

**SOP form:**

\[
    f_7(x_1, x_2) = (2[1_1^{t_1}]) \oplus (2[1_2^{t_2}]) \oplus (2[1_3^{t_3}]) \cdot (2[1_4^{t_4}])
\]

(6.11)

However, with the new approach (SOT), the function shown in Figure 6.12 can be...
Figure 6.11 HSPICE transient analysis simulation results of the circuit realization of the MVL function shown in Figure 6.10.
A given MVL function can be realized as a truncated sum of logic terms \((P, Q, R)\) in the SOT form. By definition (see Section 2.2), a logic term is defined as \(AND (\cap)\), \(OR (\cup)\), or \(XOP (\oplus)\) operation \(L_i\) unary operators, which are binary voltage signals.

\[
f_7(x_1, x_2) = T_1 \oplus T_2
\]

\[
= 2[(\uparrow x_1) \cup (\uparrow x_2)] \oplus 2[(\uparrow x_1)^2 \cap (\uparrow x_2)^2]
\]

\[\text{(6.12)}\]
Using DeMorgan’s law,

\[ P = k[L_1 \cap L_2 \cap \ldots \cap L_n] \]  \hspace{1cm} (6.13)

can be written as

\[ P = k[\overline{L_1} \cup \overline{L_2} \cup \ldots \cup \overline{L_n}]. \]  \hspace{1cm} (6.14)

Similarly,

\[ Q = k[L_1 \cup L_2 \cup \ldots \cup L_n] \]  \hspace{1cm} (6.15)

can be expressed as

\[ Q = k[\overline{L_1} \cap \overline{L_2} \cap \ldots \cap \overline{L_n}]. \]  \hspace{1cm} (6.16)

As far as SOT realization based on general structure (Figure 4.1) is concerned, for all the three logic terms (i.e., \( P, Q, R \)), the input and the output blocks are the same. However, the realization of the control logic block differs for \( P, Q, \) or \( R \) and it also depends on whether \( \overline{x}^a \) or \( \overline{x}^b \) is used in the logic term. Considering \( r = 4 \) and \( n = 2 \), the circuit realizations for the input and the output blocks are shown in Figure 6.15.

Various values of \( a_i \) and \( b_i \) can be realized by varying the size of the transistors \( T_{11} \) and \( T_{12} \) in the input block circuit. Different \( k \) values can be obtained for a term by varying the size of transistor \( T_3 \) in the output block circuit. For obtaining \( a_1 = 1 \), \( b_1 = 2 \), \( a_2 = 2 \), and \( b_2 = 2 \), the size of transistors \( T_{11}, T_{12}, T_{21}, \) and \( T_{22} \) are \( W_{11}/L_{11} = \)
Figure 6.15 For \( n = 2 \) the circuit realization of a logic term (a) the input block (b) the output block.

\[
\begin{align*}
1/2, \ W_{12}/L_{12} &= 5/2, \ W_{21}/L_{21} = 3/2, \text{ and } W_{22}/L_{22} = 5/2, \text{ respectively. As mentioned before, there are many possible logic terms depending on what binary operation is used.}
\end{align*}
\]

Using the three binary operators, AND, OR, and XOR and different

\[
L_i \in \{i(x_1)^2, i(x_2)^2, i(x_1)i(x_2), i(x_2)^3\},
\]

there are ten different possible terms. The control logic and the map for these terms are shown in Figure 6.16. It should be noted that the cost of realization for the logic terms, \( P \) and \( Q \) is the same, as both circuits utilize the same number of transistors. However, the realization of the logic term \( R \) uses more transistors as compared to the realization of the logic terms \( P \) and \( Q \). It is, however, possible to realize two \( P \) logic terms as one \( R \) logic term. This is clear from the examples shown in Figure 6.16. If the sum of the logic terms \( P_2 \) and \( P_3 \) has to be realized in a function, then it can be
Figure 6.16 For $n = 2$ and $r = 4$, the map representation and the control logic block circuit realization for the logic terms possible for $a_1 = 1$, $b_1 = 2$, $a_2 = 1$, and $b_2 = 2$. 
realized as a one logic term, $R_1$. Similarly, logic term $R_2$ can replace the sum of logic terms $P_1$ and $P_4$. So using the proposed structure, it is possible to realize powerful logic terms.

### 6.2.1 An Example

As mentioned before, a given MVL function can be decomposed into $P$, $Q$, and $R$ logic terms and realized as truncated sum of the decomposed logic terms. In addition, it has been shown that the logic terms ($P$, $Q$, and $R$) can be realized by using the general structure shown in Figure 4.1. Consider, as an example the realization
of the 4-valued 2-variable MVL function (shown in Figure 6.12). This function can be expressed as shown in Equation 6.12. The corresponding circuit realization using the general structure is shown in Figure 6.17. The circuit functionality is verified by conducting HSPICE transient analysis simulation. The simulation results are summarized in Figure 6.18, which consists of 5 panels. Each panel shows current level variation as a function of time. Panel 1 and Panel 2 show all the possible combinations for the input variables, \( x_1 \) and \( x_2 \). The variation for logic terms, \( T_1 \) and \( T_2 \) are shown in Panel 3 and Panel 4, respectively. Panel 5 shows the corresponding current level of the function \( f_6(x_1, x_2) \).
Figure 6.18 HSPICE transient analysis simulation results of the circuit realization of the MVL function shown in Figure 6.17.
6.3 Summary

In this chapter, two new forms, \textit{difference-of-sum-of-products} (DOSOP) and \textit{sum-of-terms} (SOT) for MVL function realizations have been discussed. For the DOSOP form it has been shown, with an illustrative example, that the use of subtraction operation reduces the number of PTs needed to realize some MVL functions. In addition, two structures, i.e., structure $S_1$ and structure $S_2$ have been considered for the realization of MVL functions in the DOSOP form. It has been shown that the number of PTs required for realization using these structures may be different. Some of the patterns have been reported for 4-valued 2-variable functions for which the DOSOP form realization using structure $S_1$ needs fewer PTs as compared to the realization in the SOP form.

The SOT form realization makes effective use of the general structure discussed in Section 4.1. It has been shown, with an illustrative example, that for some MVL function the SOT form realization requires less number of PTs as compared to the SOP form realization. Different realizations for a term have also been discussed in this chapter.
7. Multiple-Valued Logic Function Synthesis

Over the past several years a number of different algorithms have been proposed for MVL function synthesis. These approaches can be divided into two groups, namely cost-table approach and direct cover approach. In cost-table approach [39, 40, 41, 42, 43, 45], a given MVL function is realized as the sum of sub-functions which are stored in a table form. The table is called a cost-table because it stores the cost of realization for each sub-function. The cost of realization of a given MVL function is the sum of the costs of sub-functions plus the cost of combining them. Usually, the number of transistors required for realization is referred to as the cost of realization. It is difficult to obtain optimum cost-tables for a given technology. This represents a major disadvantage to the cost-table approach.

A direct cover approach realization consists of two steps:

1. Select a minterm.

2. Select an implicant to cover the selected minterm.

Steps 1 and 2 are iterated until all the minterms of the given function are covered. A direct cover approach realization results in structured circuits similar to binary PLA realizations. An individual PT is realized as a single column of a PLA, thus the number of PTs is referred to as the realization cost. In the previous two chapters (Chapter 5 and Chapter 6), three forms for MVL function realization have been discussed. These forms are suitable for structured implementation. In this chapter, synthesis of a given MVL function based on the SOP form will be considered.
For binary logic functions, it is known that from a given set of PTs the extraction of a minimal cover, represented in sum-of-products form, is a NP complete problem [56]. The problem of SOP extraction for MVL functions is much more complicated as compared to that of binary logic functions. For binary logic functions, it is known that the minimal SOP representation of a function consists of only prime implicants. Like binary logic functions, it has been shown by Tirumalai and Butler [55] that when the min/max operator combination is used, a minimal expression can be found using only prime implicants of the input function. However, for the min/\textit{tsum} operator combination it is necessary to consider all implicants of a given function to produce a minimal SOP expression.

As a result, heuristic techniques have been used to realize efficient circuits. Pomper and Armstrong [46] were the first to introduce a min/\textit{tsum} based direct cover approach that obtains a near minimal realization. Their approach is based on selecting a random minterm and then selecting a PT which covers the largest number of minterms. Later, Dueck and Miller [53, 38] and Besslich [47] have proposed methods where the most isolated minterm is selected rather than a random minterm. Dueck and Miller selected the product term which, when subtracted from the function, introduces the fewest discontinuities in the function. It should be noted that all the above algorithms consider all implicants that cover the selected minterm.

It is known that the Pomper and Armstrong, and the Dueck and Miller algorithms are appropriate for min/\textit{tsum} based realization whereas Besslich's algorithm is good for min/max operators [38]. Yurchak and Butler [32] have developed a CAD tool for min/\textit{tsum} based realizations. This tool is called HAMLET (Heuristic Analyzer for Multiple-valued Logic Expression Translation) and includes implementation for both the Pomper and Armstrong, and the Dueck and Miller methods. They observed that neither of the two algorithms is consistently better than the other for all functions. This resulted in another heuristic called \textit{Gold}. For a given function both the heuristics are applied, and the best realization is selected as Gold. In this chapter, a new direct cover algorithm is presented.
This chapter consists of four sections. In Section 7.1, the algorithm used for function synthesis is discussed. Synthesis of some example functions is considered in Section 7.2. Results of applying the new algorithm are compared to those results obtained using existing algorithms in Section 7.3. The chapter summary is given in Section 7.4.

7.1 The Algorithm

As mentioned previously, there have been different heuristics used for minterm and implicant selection. In this section, a new direct cover algorithm for MVL function synthesis based on the set of operators discussed in Chapter 4 and Chapter 5 is presented. Terms used in this section are defined in Section 2.3.

The algorithm is recursive in nature and consists of five functions: `main()`, `select_cover()`, `select_minterm()`, `select_pd()`, and `check_lower_level_cpts()`. The basic steps of algorithm are summarized below:

1. The MVL function to be realized is accepted in `main()` and `select_cover()` is called to obtain a cover of the function.

2. From `select_cover()`, the function `select_minterm()` is called to select a minterm using the heuristics that are discussed later in this section.

3. From `select_cover()`, the function `select_pd()` is called to select an implicant to cover the selected minterm. For the selection of an implicant, a restricted set of implicants is searched rather than considering all possible implicants. The heuristics used for selection of the implicant are discussed later in this section.

4. In `select_cover()`, after selecting an implicant, the function is simplified by subtracting the selected implicant to obtain a new function.

5. If all the minterms are covered or the new function consists of only don't care minterms, then proceed to step 6 else `select_cover()` is called for the new function (i.e., control goes to step 2).
6. After a cover or solution is obtained, the algorithm backtracks to find a better realization by exploring other implicants in the restricted set at each selection node.

The details of each function are given in the following discussion.

The purpose of the main() function is to initialize the global variables and accept the function to be realized along with candidate product term (CPT) level to be considered while selecting implicants. The pseudo-code for this function is given below.

```
Global Variables
curr_num_of_pts is the number of required PTs for current realization.
curr_realization is the list of PTs in current realization.
best_num_of_pts is the number of PTs required for so far best realization.
best_realization is the list of PTs in so far best realization.
f_map is the map of the function to be realized. It also stores selected minterm along with its CPT.
level_bound is the user's input to specify the depth of implicants to be searched for the selection of implicant.

main()
{
    // initialization
    curr_num_of_pts ← 0
    curr_realization is empty
    input f_map
    input level_bound
    best_real.num_of_pts ← number of minterms
    select.cover(f_map)
}
```

The main() function calls select.cover(). The purpose of select.cover() function is to obtain an efficient SOP form realization for a given function. The number of PTs required for the best realization so far is used as the bounding value for other realizations. The pseudo-code for this function is given below.
select_cover(f)
{
    // Variable
    ans is either True/False, where True indicates all minterms covered.
    minterm is the selected minterm.

    add 1 to curr_num_of_pts
    if (curr_num_of_pts is less than best_num_of_pts)
        minterm = select_minterm(f)
        ans ← select_pd(f, minterm, 0)
    else
        ans ← True
    subtract 1 from curr_num_of_pts
    return(ans)
}

The select_cover() function calls two functions, select_minterm() and select_pd(),
which are the core of the algorithm. As the name suggests, select_minterm() selects
a minterm and select_pd() finds an implicant (PT or PD) that covers the selected
minterm. The details of both of these are given in the following discussion.

The function select_minterm() considers all the minterms and calculates cover
ratio. The cover ratio represents the percentage of minterms covered by an implicant
having its size equal to the size of it’s CPT. When no implicant is found, cover ratio
is equal to 0. The function selects a minterm that has maximum value for cover ratio.
If there are more than one such minterms, then the minterm having minimum CPT
size among these minterms is selected. From equal CPT size minterms, the minterm
having lower value is selected. For simplicity the minterms are labelled as shown in
Figure 7.1 for 4-valued 2-variable functions. The pseudo-code for select_minterm() is
given below.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
</tbody>
</table>

Figure 7.1 Minterm labels for 4-valued 2-variable functions.
select_minterm(f)
{
    // Variables
    prev_cover_ratio is the best cover ratio so far.
    prev_cpt is the best CPT.
    prev_cpt_size is the size of best CPT.
    prev_minterm_val is the value of the best minterm.
    prev_sum_of_cpt_minterm is the sum of minterms covered by the CPT.
    prev_minterm is the best minterm.
    cpt_size is the size of current CPT.
    num_of_minterm_covered is the number of minterms covered by an implicant that
    covers the current minterm and is of size equal to cpt_size.
    cover_ratio is the cover ratio of current minterm.
    sum_of_cpt_minterm is the sum of minterms covered by the CPT.

    // Initialization
    prev_cover_ratio ← 0
    prev_cpt is constant r
    prev_minterm_val ← r
    prev_sum_of_cpt_minterm ← r^n multiplied by (r - 1)

    for each non-zero minterm, m
        calculate number of adjacencies, CPT, and cpt_size
        select maximum value for num_of_minterm_covered, for all the implicants
            covering the minterm and having size equal to cpt_size
        cover_ratio ← num_of_minterm_covered divided by cpt_size
        if (cover_ratio is less than prev_cover_ratio)
            check next minterm
        else if (cover_ratio is equal to prev_cover_ratio)
            if (cpt_size is greater than prev_cpt_size)
                check next minterm
            else if (cpt_size is equal to prev_cpt_size)
                if (value of minterm is greater than prev_minterm_val)
                    check next minterm
                else if (value of minterm is equal to prev_minterm_val)
                    calculate sum_of_cpt_minterm
                    if (sum_of_cpt_minterm is greater than or equal to
                        prev_sum_of_cpt_minterm)
                        check next minterm
        prev_minterm ← m
        prev_cover_ratio ← cover_ratio
        prev_cpt_size ← cpt_size
        prev_minterm_val ← value of minterm, m
    return (prev_minterm)
}
The function `select_pd()` considers all the implicants that cover the minterm selected by `select_minterm()` and have their size equal to the size of CPT. These implicants are searched in a decreasing order of the sum of the non-zero values of minterms covered by the implicant. If an implicant is found then a new function is obtained by subtracting the selected implicant from the input function. However, if no implicant of size equal to CPT's size is found or the user wants to search smaller size implicants, the algorithm allows to search implicants that cover lower order CPTs (see Section 2.3). The pseudo-code for this function is given below.

```plaintext
select_pd(f, minterm, level)
{
for each implicant, I, that covers the minterm and has size equal to the size of CPT
    add I to curr_realization list
for all possible minterms
    if ((minterm is don't care) or ((f(x) is less than or equal to I(x))
        and (f_map(x) is equal to (r - 1)))
        new_func ← r
    else
        new_func ← f(x) - I(x)
    if (all minterms are covered)
        best_realization ← curr_realization
        best_num_of_pts ← curr_num_of_pts
        return(True)
    else
        return(select_cover(new_func))
if ((no implicant found) or (level is less than level_bound))
    return(check_lower_level_cpt(f, minterm, level + 1))
}
```

The function `select_pd()` calls the `check_lower_level_cpt()` function to consider lower order implicants. The pseudo-code for this function is given below.
check_lower_level_cpt(f, minterm, level)
{
    for each lower level CPT, l_cpt, having non-zero value for the minterm
    ans ← select_pd(f, minterm, level)
    if (ans is True)
        return(True)
    return(False)
}

The algorithm is suitable for both Set 1 and Set 2 operators, discussed in Chapter 5. The algorithm has been implemented in C and has been tested for many 4-valued 2-variable functions. In Section 7.3, the results for these are compared with HAMLET's (Gold) results obtained using Set 3 operators for PT realization. In the following section, synthesis of an example function is considered to illustrate the steps of the algorithm along with some other examples.

7.2 Examples

Example 7.1:

Consider the 4-valued 2-variable function, $f_8(x_1, x_2)$ shown in Figure 7.2. Using Set 1 operators for the realization of a PT and implicants of size equal to the size of

```
\begin{array}{cccc}
  x_1 & 0 & 1 & 2 & 3 \\
  x_2 & 0 & 2 & 1 & 2 \\
       & 1 & 2 & 3 & 1  \\
       & 2 & 1 & 2 & 1  \\
       & 3 & 2 & 1 & 3  \\
\end{array}
```

Figure 7.2 An example 4-valued 2-variable function to illustrate the steps of the algorithm.
CPT (i.e., level bound is 1), the algorithm proceeds as follows:

1. best_num_of_pts ← 14 (i.e., number of minterms), best_realization is the list of minterms, curr_num_of_pts ← 0, and curr_realization list is empty.

2. The function select_cover() is called for the function \( f_8(x_1, x_2) \) shown in Figure 7.2. A one is added to curr_num_of_pts to make it equal to 1. As the curr_num_of_pts (i.e., 1) is less than the best_num_of_pts (i.e., 14), select_minterm() is called for the function \( f_8(x_1, x_2) \).

3. The function consists of 14 minterms. Starting with the first minterm number of adjacencies, CPT, cpt_size, cover_ratio, sum_of cpt_minterm are calculated for all the minterm. The values obtained for each minterm are summarized below in tabular form.

<table>
<thead>
<tr>
<th>Minterm label</th>
<th>Adjacencies x1 x2</th>
<th>CPT ( 4^2(x_1^2) )</th>
<th>cpt_size</th>
<th>cover_ratio</th>
<th>sum_ofcpt_minterm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2 2 2 3</td>
<td>12</td>
<td>0.00</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1 2 3 1</td>
<td>12</td>
<td>0.33</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2 2 2 3</td>
<td>12</td>
<td>0.00</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2 3 3 3</td>
<td>4</td>
<td>16</td>
<td>0.00</td>
<td>24</td>
</tr>
<tr>
<td>6</td>
<td>3 3 3 3</td>
<td>4</td>
<td>16</td>
<td>0.00</td>
<td>24</td>
</tr>
<tr>
<td>7</td>
<td>1 3 1 3</td>
<td>4</td>
<td>16</td>
<td>0.50</td>
<td>13</td>
</tr>
<tr>
<td>8</td>
<td>2 3 3 3</td>
<td>4</td>
<td>16</td>
<td>0.00</td>
<td>24</td>
</tr>
<tr>
<td>9</td>
<td>1 3 3 3</td>
<td>4</td>
<td>16</td>
<td>0.00</td>
<td>24</td>
</tr>
<tr>
<td>10</td>
<td>2 3 3 3</td>
<td>4</td>
<td>16</td>
<td>0.00</td>
<td>24</td>
</tr>
<tr>
<td>11</td>
<td>1 3 1 3</td>
<td>4</td>
<td>16</td>
<td>0.50</td>
<td>13</td>
</tr>
<tr>
<td>12</td>
<td>1 3 3 3</td>
<td>4</td>
<td>16</td>
<td>0.00</td>
<td>24</td>
</tr>
<tr>
<td>13</td>
<td>2 2 3 3</td>
<td>12</td>
<td>0.00</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>1 2 3 1</td>
<td>12</td>
<td>0.33</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>3 2 3 2</td>
<td>12</td>
<td>0.60</td>
<td>22</td>
<td></td>
</tr>
</tbody>
</table>
Among all the minterms, minterm 7 and 11 both have the maximum cover_ratio, i.e., 0.50. The size of CPT, minterm value, and the sum of CPT minterms are also the same. Thus, the minterm 7 is selected as it is the first to give maximum cover_ratio. Once a minterm is selected, the function returns to select_cover() and calls function select_pd().

4. There is only one implicant of size equal to cpt_size that covers the selected minterm. The implicant is $1[\bar{x}_1x_2^2]$. It is selected and the new function is shown below.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$f_{8a}(x_1, x_2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

For the new function $f_{8a}(x_1, x_2)$, the function select_cover() is called. The curr_num_of_pts is incremented by one. Again as the curr_num_of_pts (i.e., 2) is less than best_num_of_pts (i.e., 14), the function select_minterm() is called.

5. The function $f_{8a}(x_1, x_2)$ consists of 10 minterms. Again the number of adjacencies, CPT, cpt_size, cover_ratio, sum_of_cpt_minterm are calculated for all the minterms. It is found that there are 6 minterms (i.e., minterms 1, 4, 5, 8, 13, and 16) having the maximum cover_ratio, i.e., 0.88, and the same cpt_size (i.e., 9). According to the algorithm, the minterm having the lowest value is selected. However, there are two minterms (i.e., minterm 5 and 8) that have the same minimum value and their sum_of_cpt_minterm are also equal. Minterm 5 is selected as it was considered first. The function returns to select_cover() and calls the function select_pd().

6. There are two implicants that cover the selected minterm. They are: $(\bar{x}_1^3) \cdot (\bar{x}_2^3)$
and \((1^2(x_1^2)) \cdot (1^2(x_2^2))\). Among these, \((\overline{x_1}^3) \cdot (\overline{x_2}^3)\) is selected first because the sum of minterms covered by this implicant is 14 as compared to 9 if implicant \((1^2(x_1^2)) \cdot (1^2(x_2^2))\) is selected. On subtracting \((\overline{x_1}^3) \cdot (\overline{x_2}^3)\), the function obtained is shown below.

For the new function \(f_{s6}(x_1, x_2)\), the function \(select\_cover()\) is called. The first step is to increment \(curr\_num\_of\_pts\) by one. Still, the \(curr\_num\_of\_pts\) (i.e., 3) is less than \(best\_num\_of\_pts\) (i.e., 14), the function \(select\_minterm()\) is called.

7. The function \(f_{s6}(x_1, x_2)\) consists of 2 minterms (i.e., minterm 6 and 10) that need to be covered. For both the minterms, the number of adjacencies, CPT, \(cpt\_size\), \(cover\_ratio\), \(sum\_of\_cpt\_minterm\) are the same. Thus, minterm 6 is selected and the function \(select\_minterm()\) returns to \(select\_cover()\) function, and function \(select\_pd()\) is called.

8. There is only one implicant that covers the selected minterm. The implicant is \((1^1(x_1^1)) \cdot (1^1(x_2^1))\). This is selected and the new function is shown below.
The new function consists of only don't care minterms, thus a cover is obtained which is better than the initial solution. The best_realization list of PTs is replaced by curr_realization list and best_num_of_pts is set equal to curr_num_of_pts (i.e., 3). The function returns to select_cover() where one is subtracted from curr_num_of_pts and the algorithm backtracks to find a better solution.

9. The control goes back to step 6 and the second implicant, \((1[\overline{x_1}]) \cdot (1[\overline{x_2}])\) is selected for function \(f_{8a}(x_1, x_2)\). On subtracting \((1[\overline{x_1}]) \cdot (1[\overline{x_2}])\) from \(f_{8a}(x_1, x_2)\), the function obtained is shown below.

\[
\begin{array}{c|cccc}
  & 0 & 1 & 2 & 3 \\
\hline
0 & 1 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 2 \\
2 & 1 & 0 & 1 & 0 \\
3 & 1 & 0 & 0 & 2 \\
\end{array}
\]

The new function consists of six uncovered minterms, the function select_cover() is called again.

10. In select_cover() function the curr_num_of_pts is incremented by one and becomes equal to best_num_of_pts (i.e., 3). Thus, the function returns to select_pd() after subtracting one from curr_num_of_pts (i.e., 2). As in select_pd() there are no more implicants to be considered, the control returns to select_cover(). In select_cover(), the curr_num_of_pts is decremented by 1 before returning to previous select_pd() call. There also no more implicants have to be considered, the control returns to main() function via select_cover() function.

11. The best realization consists of 3 PTs

\[
f_8(x_1, x_2) = (1[1(x_2)^2]) \oplus (\overline{x_1}^3) \cdot (\overline{x_2}^3) \oplus (1[1(x_1)^1]) \cdot (1[1(x_2)^2]).
\]
For the above example, using HAMLET (Gold) approach 8 PTs are required as shown below.

\[ f_8(x_1, x_2) = (1[0(x_1)^1]) \oplus (1[0(x_1)^0]) \bullet (1[0(x_2)^1]) \oplus (1[0(x_2)^0]) \]
\[ (1[1(x_1)^1]) \bullet (1[1(x_1)^0]) \oplus (1[1(x_2)^1]) \bullet (1[1(x_2)^0]) \]
\[ (1[2(x_1)^1]) \bullet (2[0(x_2)^1]) \oplus (2[0(x_2)^0]) \]
\[ (2[1(x_2)^1]) \bullet (2[2(x_2)^0]) \oplus (3[0(x_1)^1]) \bullet (3[0(x_2)^0]). \] (7.2)

**Example 7.2:**

Consider the realization for the 4-valued 2-variable function shown in Figure 7.3, using the developed program. The PTs are realized using Set 2 operators. The function consists of 10 minterms. Using the new algorithm with level_bound as one, 5 PTs are required. The SOP form realization of function is

\[ f_9(x_1, x_2) = (1[0(x_1)^1]) \bullet (1[0(x_2)^1]) \oplus (2[0(x_1)^0]) \bullet (2[0(x_2)^0]) \]
\[ (2[1(x_1)^1]) \bullet (2[1(x_2)^1]) \oplus (2[1(x_2)^0]) \]
\[ (2[2(x_1)^1]) \bullet (2[2(x_2)^0]) \oplus (3[0(x_1)^0]) \bullet (3[0(x_2)^1]). \] (7.3)

For HAMLET (Gold) approach, 9 PTs are required as shown in Equation 7.4.
It should be noted that the algorithm does not always outperform HAMLET (Gold). Example 7.3 shows a 4-valued 2-variable function for which the HAMLET (Gold) approach requires fewer PTs as compared to the realization based on the new algorithm, restricting implicants to the size of CPT (i.e., `level_bound` equal to 1).

**Example 7.3:**

Consider the 4-valued 2-variable function shown in Figure 7.4. The function consists of 9 minterms. Using the new algorithm with `level_bound` as one and `Set 1` operators, 7 PTs are required. The SOP form realization of function is shown in Equation 7.5.

$$f_{10}(x_1, x_2) = (1[^0(x_2)^0] \boxplus (1[^0(x_1)^0]) \bullet (1[^0(x_2)^0])) \bullet (1[^0(x_2)^0]) \boxplus ((1[^0(x_2)^0]) \bullet (1[^0(x_2)^0])) \boxplus (2[^0(x_1)^0]) \bullet (1[^0(x_2)^0])) \bullet (1[^0(x_2)^0]) \boxplus (2[^0(x_1)^0]) \bullet (1[^0(x_2)^0])) \bullet (1[^0(x_2)^0]) \boxplus (1[^0(x_2)^0]) \bullet (1[^0(x_2)^0]) \boxplus (3[^0(x_1)^0]) \bullet (2[^0(x_1)^0]) \boxplus (3[^0(x_1)^0]) \bullet (3[^0(x_1)^0])) \bullet (3[^0(x_1)^0]). \quad (7.5)$$

**Figure 7.4** An example function for which the HAMLET (Gold) requires fewer PTs as compared to the algorithm when `level_bound` is 1.
For HAMLET (Gold) approach, only 6 PTs are required as shown in Equation 7.6.

\[
f_{10}(x_1, x_2) = (1[2(x_1)^2]) \cdot (1[0(x_2)^0]) \oplus (1[0(x_1)^0]) \cdot (1[2(x_2)^2]) \oplus \\
(1[1(x_1)^1]) \cdot (1[2(x_2)^2]) \oplus (2[0(x_1)^0]) \cdot (2[0(x_2)^0]) \oplus \\
(3[2(x_1)^2]) \cdot (3[0(x_2)^0]) \oplus (3[1(x_1)^1]) \cdot (3[2(x_2)^2]).
\] (7.6)

For the above example, if level bound is increased by 1 then the new algorithm also requires 6 PTs as shown in Equation 7.7.

\[
f_{10}(x_1, x_2) = (1[2(x_2)^2]) \oplus (1[1(x_1)^1]) \cdot (1[2(x_2)^2]) \oplus (3[2(x_1)^2]) \cdot (x_2^2) \\
\oplus (2[0(x_1)^0]) \cdot (2[0(x_2)^0]) \oplus (3[2(x_1)^2]) \cdot (3[0(x_2)^0]) \oplus \\
(1[2(x_1)^2]) \cdot (1[2(x_2)^2]).
\] (7.7)

There are some examples for which the new algorithm always requires more PTs. Example 7.4 shows a 4-valued 2-variable function for which the HAMLET (Gold) approach requires fewer PTs as compared to the realization based on the new algorithm.

**Example 7.4:**

Consider the 4-valued 2-variable function shown in Figure 7.5. The function

![Figure 7.5](image)

**Figure 7.5** An example function for which the HAMLET (Gold) requires fewer PTs as compared to the algorithm.
consists of 10 minterms. Using Set 1 operators the new algorithm’s realization always consists of 8 PTs, irrespective of level bound value. The SOP form realization of function is shown in Equation 7.8.

\[
f_{11}(x_1, x_2) = (2[1^1(x_1)^2]) \cdot (2[1^2(x_2)^2]) \oplus (3[1^1(x_1)^2]) \cdot (3[1^2(x_2)^2]) \oplus \\
(1[2^1(x_1)^2]) \cdot (1[2^2(x_2)^2]) \oplus (1[0^1(x_1)^2]) \cdot (1[0^2(x_2)^2]) \oplus \\
(x_1^0) \cdot (3[1^1(x_2)^2]) \oplus (3[2^1(x_1)^2]) \cdot (3[1^2(x_2)^2]) \oplus (3[0^1(x_2)^2]) \oplus \\
(1[0^1(x_1)^2]) \cdot (1[0^2(x_2)^2]) \oplus (1[1^1(x_1)^2]) \cdot (1[1^2(x_2)^2])
\]  

(7.8)

For HAMLET (Gold) approach, only 7 PTs are required as shown in Equation 7.9.

\[
f_{11}(x_1, x_2) = (1[3^1(x_1)^2]) \cdot (1[3^2(x_2)^2]) \oplus (1[1^1(x_1)^2]) \cdot (1[1^2(x_2)^2]) \oplus \\
(2[3^1(x_1)^2]) \cdot (2[3^2(x_2)^2]) \oplus (2[0^1(x_1)^2]) \cdot (2[0^2(x_2)^2]) \oplus \\
(2[1^1(x_1)^2]) \cdot (2[1^2(x_2)^2]) \oplus (3[1^1(x_1)^2]) \cdot (3[1^2(x_2)^2]) \oplus \\
(3[2^1(x_1)^2]) \cdot (3[1^2(x_2)^2])
\]  

(7.9)

From all the above examples, it is seen that for some functions the new algorithms is superior and for others the new algorithm is inferior to HAMLET (Gold). In order to observe the performance over a wider range of functions, both of the algorithms were tested using 5500 randomly generated functions. The results of this comparison are discussed in the following section.

7.3 Results

As mentioned previously, there are \(4^{4^2}\) (i.e., 4, 294, 967, 296) different possible 4-valued 2-variable functions. It is impossible to synthesize all of these. However, there are 16 minterms and the functions can be divided into seventeen categories depending on the number of non-zero minterms. Number of functions along with the percentage of total number of functions in each category are shown in Table 7.1.
Table 7.1  For 4-valued 2-variable functions, the distribution of number of functions along with the percentage of total number of functions based on number of non-zero minterms.

<table>
<thead>
<tr>
<th>Number of non-zero minterms</th>
<th>Number of functions</th>
<th>Percentage of total possible functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2.32 * 10^{-8}</td>
</tr>
<tr>
<td>1</td>
<td>48</td>
<td>1.12 * 10^{-6}</td>
</tr>
<tr>
<td>2</td>
<td>1,080</td>
<td>2.51 * 10^{-5}</td>
</tr>
<tr>
<td>3</td>
<td>15,120</td>
<td>3.52 * 10^{-4}</td>
</tr>
<tr>
<td>4</td>
<td>147,420</td>
<td>0.003</td>
</tr>
<tr>
<td>5</td>
<td>1,061,424</td>
<td>0.024</td>
</tr>
<tr>
<td>6</td>
<td>5,837,832</td>
<td>0.136</td>
</tr>
<tr>
<td>7</td>
<td>25,019,280</td>
<td>0.582</td>
</tr>
<tr>
<td>8</td>
<td>84,440,070</td>
<td>1.966</td>
</tr>
<tr>
<td>9</td>
<td>225,173,520</td>
<td>5.243</td>
</tr>
<tr>
<td>10</td>
<td>472,864,392</td>
<td>11.010</td>
</tr>
<tr>
<td>11</td>
<td>773,778,096</td>
<td>18.016</td>
</tr>
<tr>
<td>12</td>
<td>967,222,620</td>
<td>22.520</td>
</tr>
<tr>
<td>13</td>
<td>892,820,880</td>
<td>20.788</td>
</tr>
<tr>
<td>14</td>
<td>573,956,280</td>
<td>13.363</td>
</tr>
<tr>
<td>15</td>
<td>229,582,512</td>
<td>5.345</td>
</tr>
<tr>
<td>16</td>
<td>43,046,721</td>
<td>1.002</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>4,294,967,296</strong></td>
<td><strong>100</strong></td>
</tr>
</tbody>
</table>

Out of these 17 categories, the developed program has been tested for functions having more than 5 non-zero minterms. These functions are represented by the last 11 categories. These categories represent more than 99.9% of the functions. For each category, 500 functions have been generated randomly using the HAMLET program. Each function has been decomposed based on Set 1 and Set 2 operators using the program written. In addition, each function has been realized by using the HAMLET's (Gold) approach based on Set 3 operators. The comparison results for each category using Set 1 operators with the developed program and the HAMLET (Gold) program are summarized in Table 7.2. From the table it can be seen that when all the categories are combined, for 27% of the functions the number of PTs required are the same for both the approaches. For 69% of the functions, the developed program performs
Table 7.2 Comparison of synthesis results, for 4-valued 2-variable functions, using the developed program (Set 1 operators) and the HAMLET (GOLD) (Set 3 operators) for 5500 randomly generated function.

<table>
<thead>
<tr>
<th>Number of non-zero minterms</th>
<th>Number of PTs</th>
<th>Equal</th>
<th>Developed Program Better</th>
<th>HAMLET (GOLD) Better</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>284</td>
<td>205</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>216</td>
<td>260</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>181</td>
<td>302</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>130</td>
<td>350</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>129</td>
<td>344</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>114</td>
<td>364</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>83</td>
<td>397</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>66</td>
<td>419</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>81</td>
<td>406</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>77</td>
<td>412</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>132</td>
<td>351</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>1493</td>
<td>3810</td>
<td>197</td>
<td></td>
</tr>
</tbody>
</table>

better and for 4% of the functions HAMLET (Gold) approach is better. However, for categories having non-zero minterms in the range of 9 to 15, which represents over 96% of the possible functions, there are 19% functions for which the number of PTs are the same. For 77% of the functions the developed program performs better and for 4% of the functions HAMLET (Gold) gives superior results.

A similar analysis was done for the developed program using Set 1 and Set 2 operators for realizing PTs. The comparison results for each category using Set 1 and the Set 2 operators are summarized in Table 7.3. For all the categories combined, the number of PTs are the same for 60% of the functions for both the set of operators. For 39% of the functions, realizations based on Set 1 operator require fewer PTs as compared to Set 2 realizations. There are 1% of the functions for which Set 2 realizations are superior as compared to Set 1 realizations. Considering only the categories having non-zero minterms in the range of 9 to 15, for 52% of the functions equal number of PTs are required for both the set of operators. For 47% of the functions
Table 7.3 Comparison of synthesis results, for 4-valued 2-variable functions, SOP form realization based on Set 1 and Set 2 operators using the developed program for 5500 randomly generated function.

<table>
<thead>
<tr>
<th>Number of non-zero minterms</th>
<th>Number of PTs</th>
<th>Set 1 Realization Better</th>
<th>Set 2 Realization Better</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Equal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>409</td>
<td>90</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>373</td>
<td>125</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>324</td>
<td>173</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>281</td>
<td>213</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>279</td>
<td>213</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>242</td>
<td>250</td>
<td>8</td>
</tr>
<tr>
<td>12</td>
<td>221</td>
<td>272</td>
<td>7</td>
</tr>
<tr>
<td>13</td>
<td>216</td>
<td>280</td>
<td>4</td>
</tr>
<tr>
<td>14</td>
<td>264</td>
<td>229</td>
<td>7</td>
</tr>
<tr>
<td>15</td>
<td>302</td>
<td>119</td>
<td>7</td>
</tr>
<tr>
<td>16</td>
<td>374</td>
<td>118</td>
<td>8</td>
</tr>
<tr>
<td>Total</td>
<td>3285</td>
<td>2154</td>
<td>61</td>
</tr>
</tbody>
</table>

- The Set 1 realizations are superior and for 1% of the functions Set 1 realizations are inferior as compared to Set 2 realizations.

- In order to compare the performance based on all the three set of operators, the average number of PTs for each category have been plotted in Figure 7.6. In the figure, the labels New (Set 1) and New (Set 2) are the average PTs for different categories using the developed program for Set 1 and Set 2 operators, respectively. The label Gold (Set 3) indicates average PTs for the realizations based on Set 3 operators using HAMLET (Gold) program. The overall average number of PTs for 5500 functions using the developed program based on Set 1 and Set 2 operators, and HAMLET (Gold) using Set 3 operator are 5.53, 6.02, and 6.62, respectively.
A synthesis algorithm for the SOP form realization of MVL functions has been discussed in this chapter. The algorithm can be used for both the cases, when a PT is realized by using Set 1 or Set 2 operators. The algorithm is divided into five sub-functions: main(), select_cover(), select_minterm(), select_pd(), and check_lower_level_cpts(). The details for each of these functions are given along with synthesis of an example function, to illustrate the steps of the algorithm.

The algorithm has been implemented in C and tested for 5500 randomly generated 4-valued 2-variable functions. These functions are classified into 11 categories based on the number of non-zero minterms. Each category consists of 500 example functions. It is found that for 27% of the functions the number of PTs required for the realization based on the Set 1 operator using the developed program are the same as compared to the realizations obtained using HAMLET (Gold), which are based on Set 3 operator. For 69% of the functions the developed program performs better.
and for 4% of the functions HAMLET (Gold) approach is better. For all the 5500 functions, a similar analysis was carried out for the developed program using Set 1 and Set 2 operators for realizing PTs. The number of required PTs are the same for 60% of the functions for both set of operators. For 39% of the functions, realizations based on Set 1 operator need fewer PTs as compared to Set 2 realizations. There are 1% of the functions for which Set 2 realizations are superior as compared to Set 1 realizations. The overall average number of PTs for 5500 functions using the developed program based on Set 1 and Set 2 operators, and HAMLET (Gold) using Set 3 operators are 5.53, 6.02, and 6.62, respectively.
8. Conclusions and Future Work

The objectives of the research work reported in this thesis were the following:

1. Analyse the existing sets of operators and identify new set(s) of operators that can be realized using existing binary logic CMOS fabrication processes.

2. Develop a heuristic-based technique for the sum-of-product realization of MVL functions using the new set(s) of operators, with an objective that further improvements could be achieved as compared to the realizations based on the existing set of operators.

3. Investigate new techniques for MVL function realization that allow further reduction in the chip area required.

8.1 Conclusions

This thesis considers multiple-valued logic design in current-mode CMOS. In this technology, multiple-valued logic levels are represented in terms of current values. For example, logic 0 = no current, logic 1 = $I_0$, logic 2 = $2I_0$, and so on, where $I_0 = 20 \mu$A. A given MVL function can be realized in the sum-of-product (SOP) form. The cost of realization, i.e., number of required product terms (PTs), depends on which of the existing sets of operators is used. The choice of operators is technology dependent. In order to obtain cost effective realizations for MVL functions using current-mode CMOS, a new set of operators has been identified. The set consists of literal, cycle, complement of literal, complement of cycle, min, and tsum operators. These operators have been selected after analysing the existing set of operators (for example, literal, min, and tsum; cycle, unary inverter, min, and max; and literal, complement of literal, min, and max). This was the first task stated in the research objectives.

In order to realize MVL circuits in current-mode CMOS, a general structure has been identified. The structure consists of three blocks: input, control, and output.
Among these, the input and the output blocks deal primarily with multiple-valued current-mode signals. However, the control block input/output signals are binary and are represented as voltage signals. Using this general structure, circuit realizations have been reported for the proposed set of operators. These circuit realizations are suitable for an existing binary logic CMOS fabrication process (CMOS3DLM). It has been shown, that the cost (in terms of number of transistors) of realizing the literal or complement of literal operator is the same, and this cost is comparable to that of the cycle operator. Thus, the costs of realizing the new set of operators and the existing set of operators (i.e., literal, min, and tsum) are comparable. The functionality of the designed circuits has been verified for 4-valued logic by HSPICE transient analysis simulation results using Northern Telecom CMOS3DLM process parameters provided by Canadian Microelectronics Corporation (CMC).

Current-mode CMOS logic is not a self restoring logic and there is some deterioration of MVL current levels as signals propagate through various circuit elements. Therefore, it may be necessary to restore the MVL current values to proper levels. For 4-valued logic, two realizations have been reported for level restorer circuits. Again the functionality of the designed circuits has been verified by HSPICE transient analysis simulation.

For some example functions, it has been shown that by using the new set of operators it is possible to reduce the number of PTs required to realize a given MVL function in a SOP form, as compared to that required by using an existing set of operators. In order to conduct a comprehensive comparison, a program has been written for r-valued n-variable MVL functions to determine how the required number of PTs vary in the SOP realization using different set of operators. The program has been tested for 3-valued 2-variable functions. The maximum number of product terms required to realize all 19683 functions is reduced from 6 using the existing set of operators to 3 using the new set of operators. In addition, the average number of product terms is decreased from 3.61 using the existing set of operators to 2.61 using the new set of operators. It has been shown, with illustrative examples, that
the improvement achieved by the new set of operators is not restricted to 3-valued 2-variable functions. The maximum propagation delays for the example circuits, which depend on the current levels switched at the input and the output, are in the range of 3.77 ns to 13.97 ns for typical process parameters.

A new heuristic based synthesis program has been developed to realize a given MVL function in terms of the new set of operators, which is the second task stated in the research objectives. The program has been tested for 4-valued 2-variable functions. There are $4^{(4^2)}$ (i.e., 4,294,967,296) different possible functions. It is impossible to realize all of these. However, there are a maximum of 16 minterms possible for these functions and the functions can be divided into 17 categories depending on the number of non-zero minterms. Out of these 17 categories, the developed program has been tested for 11 categories which represent more than 99.9% of the functions. For each category, 500 functions have been generated randomly using the HAMLET [32] program. The new program has been tested for these 5500 functions. It is found that for 27% of the functions the number of PTs required for the realization using the developed program based on the new set of operators are the same as compared to the realization obtained using HAMLET (Gold), which is based on the existing set of operators. For 69% of the functions the developed program performs better and for 4% of the functions the HAMLET (Gold) approach is better. The average number of PTs required by the developed program and HAMLET (Gold) are 5.53 and 6.62, respectively. It is seen that the use of new set of operators reduces the number of PTs required for the realization of MVL functions. However, the developed program does not always generate a better realization as compared to HAMLET (Gold). This can be attributed to the heuristics used in the developed program, which needs further investigation.

In order to further reduce the number of product terms required for MVL function realization, many examples have been studied for the SOP realization. This resulted in the identification of two new approaches, which is the third task stated in the research objectives. The new approaches are based on: difference-of-sum-of-
products (DOSOP) form and sum-of-terms (SOT) realization of MVL functions. It has been shown, with illustrative examples, that using the new approaches the number of required terms can be reduced in the realization of some MVL functions as compared to those obtained using the SOP realization. In the DOSOP realization, the original function is partitioned into two sub-functions. Individually, these sub-functions are realized in the SOP form, and the difference of these realizations forms the original function. Some patterns have been identified for which there is some possible improvements using the DOSOP realization of MVL functions as compared to that obtained for the SOP realization. The range of improvements is from 1 to 5 PTs for the reported examples. The SOT form of MVL function realization makes effective use of the general structure identified for current-mode CMOS realization of MVL circuits. Using different designs to realize the control block, it is possible to realize powerful terms, that allow reduction in the required number of PTs for the SOP realization.

8.2 Future Directions

There are some future research topics that are prompted by the work reported in this thesis.

1. Incorporate difference-of-sum-of-products and sum-of-terms forms of realization in the synthesis program for obtaining better realizations. This may require a further investigation of difference-of-sum-of-products and sum-of-terms forms of realizations. In addition, investigate improvements in the heuristics used in the synthesis program reported in this thesis.

2. Using other process technologies, like CMOS4S, BiCMOS, and GaAs MESFET, design circuits that are suitable for the new set of operators. A preliminary investigation has been carried out for BiCMOS. Some of the circuits reported in this thesis have been been appropriately modified for the BiCMOS process technology. These modified circuits have been simulated using HSPICE transient analysis simulation for default parameters. It has been observed that propagation delay for BiCMOS circuits is reduced by a factor of 2 as compared to the corresponding CMOS circuit realizations.

3. In order to automate the design cycle of MVL functions, it is necessary to generate standard cell layouts for the circuits reported in this thesis. The standard
cells can be used for realizing circuits for various applications of MVL, such as design of arithmetic circuits. In addition, the realizations of MVL functions reported in this thesis can be described in a hardware description language in terms of these cells. These descriptions can be used to automatically generate geometric layout for the function realization.

4. The circuits reported in this thesis can also be used to realize MVL neural networks. An example neuron has been designed and simulated [69].
References


Appendix A. Typical Process Parameters for CMOS3DLM

Following is a list of typical process parameters for CMOS3DLM. These parameters have been provided by Canadian Microelectronics Corporation (CMC) and have been used for HSPICE simulations.

.MODEL MOD1 NMOS ( LEVEL=3 VTO=0.7 KP=40E-6 GAMMA=1.1 PHI=0.6 + LAMBDA=1.0E-2 PB=0.7 CGSO=3.E-10 CGDO=3.E-10 CGBO=5.0E-10 + RSH=25 CJ=4.4E-4 MJ=0.5 CJSW=4E-10 MJSW=0.3 JS=1.0E-5 + TOX=5.0E-8 NSUB=1.7E+16 TPG=1 XJ=6.0E-7 LD=3.5E-7 U0=775 )

.MODEL MOD2 PMOS ( LEVEL=3 VTO=-0.8 KP=12.E-6 GAMMA=0.6 PHI=0.6 + LAMBDA=3.0E-2 PB=0.6 CGSO=2.5E-10 CGDO=2.5E-10 CGBO=5.0E-10 + RSH=80 CJ=1.5E-4 MJ=0.6 CJSW=4.E-10 MJSW=0.6 JS=1.0E-5 + TOX=5.0E-8 NSUB=5.0E+15 TPG=1 XJ=5.E-7 LD=2.50E-7 U0=250 )