Algorithms in Computer Aided Design of VLSI Circuits

by

Meng Yang
BEng(Hon.), M.Sc.

A thesis submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

Napier University
School of Engineering
Edinburgh, UK

June 2006

Dedicated to

my great parents

Mrs. Fengying Zhang and Mr. Yaoshun Yang

my grandpa

Mr. Keshen Zhang

and my uncle

Mr. Yine Zhang

Abstract

With the increased complexity of Very Large Scale Integrated (VLSI) circuits, Computer Aided Design (CAD) plays an even more important role. Top-down design methodology and layout of VLSI are reviewed. Moreover, previously published algorithms in CAD of VLSI design are outlined.

In certain applications, Reed-Muller (RM) forms when implemented with AND/XOR or OR/XNOR logic have shown some attractive advantages over the standard Boolean logic based on AND/OR logic. The RM forms implemented with OR/XNOR logic, known as Dual Forms of Reed-Muller (DFRM), is the Dual form of traditional RM implemented with AND/XOR.

Map folding and transformation techniques are presented for the conversion between standard Boolean and DFRM expansions of any polarity. Bidirectional multi-segment computer based conversion algorithms are also proposed for large functions based on the concept of Boolean polarity for canonical product-of-sums Boolean functions. Furthermore, another two tabular based conversion algorithms, serial and parallel tabular techniques, are presented for the conversion of large functions between standard Boolean and DFRM expansions of any polarity. The algorithms were tested for examples of up to 25 variables using the MCNC and IWLS'93 benchmarks.

Any n-variable Boolean function can be expressed by a Fixed Polarity Reed-Muller (FPRM) form. In order to have a compact Multi-level MPRM (MMPRM) expansion, a method called on-set table method is developed. The method derives MMPRM expansions directly from FPRM expansions. If searching all polarities of FPRM expansions, the MMPRM expansions with the least number of literals can be obtained. As a result, it is possible to find the best polarity expansion among 2^n FPRM expansions instead of searching $2^{n2^{n-1}}$ MPRM expansions within reasonable time for large functions. Furthermore, it uses on-set coefficients only and hence reduces the usage of memory dramatically.

Currently, XOR and XNOR gates can be implemented into Look-Up Tables (LUT) of Field Programmable Gate Arrays (FPGAs). However, FPGA placement is categorised to be NP-complete. Efficient placement algorithms are very important to CAD design tools. Two algorithms based on Genetic Algorithm (GA) and GA with Simulated Annealing (SA) are presented for the placement of symmetrical FPGA. Both of algorithms could achieve comparable results to those obtained by Versatile Placement and Routing (VPR) tools in terms of the number of routing channel tracks.

Declaration

The work in this thesis is based on research carried out at the School of Engineering, Napier University, Edinburgh, UK. No part of this thesis has been submitted elsewhere for any other degree or qualification and it all my own work unless referenced to the contrary in the text.

Acknowledgments

First of all, my wholehearted thanks go to my beloved mum, Fengying Zhang, and great dad, Yaoshun Yang, who took care of everything for me since I was a crying baby, for their everlasting love and understanding. Without their sacrifice, I am nowhere near the completion of this thesis.

I deeply appreciate my research supervisor Prof. A.E.A. Almaini, School of Engineering, Napier University, for his constant guidance, encouragement, friendship in the past several years. With his help, I was able to motivate myself to continue Doctor degree after completing project of Master of Science with him. Throughout the period of research, he not only provided comments and invaluable suggestions on the meeting but spent his precious time in correcting each single of my published scientific papers and this thesis patiently as well. It is my great pleasure and privilege to study under his supervision.

I would like to thank my second supervisor Prof. Lingli Wang, who was a former member of the digital techniques group and was a senior software engineer at the Altera European Technology Center, Buckinghamshire, UK, and is currently with Fudan University, Shanghai, China. His essential and consistent supports help me learn Linux operating system at the very beginning of the research and GNU C language programming skills during the research. In addition, he patiently provided useful comments and suggestions on the

direction of research in the past several years.

I would like to thank my internal supervisor Dr. M. Sharif for his support and guidance at the very beginning of the research.

My gratitude is given to former members of the digital techniques group, Prof. Yinshui Xia, Dr Belgasem Ali and current member Miss Pauline Oh, and also Mr. Lunyao Wang, Prof. Pengjun Wang and Prof. Hongying Xu, who were visiting Scholars at Napier University, for their interesting and various discussions.

Extra special thanks must go to Prof. Pengjun Wang, who was a visiting Scholar at Napier University and is with Ningbo University, Ningbo, China, for his constant encouragement and friendship during my research.

This research work was funded by the School of Engineering, Napier University, Edinburgh, UK. The support is gratefully acknowledged.

Last certainly not least, many thanks go to my relatives and friends for their supports in all different kinds of ways throughout my research work, especially Mr. Yine Zhang, who not only financially supported my Master's degree in the UK, but gave me priceless suggestions and guidance as well.

Contents

List of Abbreviations

ASIC	Application Specific Integrated Circuit
BDD	Binary Decision Diagram
CAD	Computer Aided Design
CCF	Conjunctive Canonical Form
CLB	Configurable Logic Block
COC	Canonical OR-Coincidence
CPOS	Canonical Product-of-Sums
CSOP	Canonical Sum-of-Products
DCF	Disjunctive Canonical Form
DFRM	Dual Forms of Reed-Muller
ESOP	Exclusive-OR Sum-of-Products
FPGA	Field Programmable Gate Array
FPRM	Fixed Polarity Reed-Muller
FSM	Finite State Machine

GA	Genetic Algorithm
GCC	GNU C Compiler
GRM	Generalised Reed-Muller
HDL	Hardware Description Language
HGA	Hybrid Genetic Algorithm
IOB	Input/Output Block
IWLS	International Workshop on Logic Synthesis
KRO	Kronecker
LUT	Look-Up Table
MCNC	Microelectronics Center of North Carolina
MMPRM	Multi-level Mixed Polarity Reed-Muller
MPGA	Mask Programmable Gate Array
MPRM	Mixed Polarity Reed-Muller
NP	Nondeterministic Polynomial, Non-Polynomial
NPRM	Negative Polarity Reed-Muller
NRE	Non-Recurring Engineering
OBDD	Ordered Binary Decision Diagram
OFDD	Ordered Function Decision Diagram
P&R	Placement and Routing

PAL	Programmable Array Logic
PPRM	Positive Polarity Reed-Muller
PSDKRO	Pseudo Kronecker
PSDRM	Pseudo Reed-Muller
PSO	Particle Swarm Optimisation
PLA	Programmable Logic Array
PTT	Parallel Tabular Technique
RM	Reed-Muller
RMBDD	Reed-Muller Binary Decision Diagram
RTL	Register Transfer Level
SA	Simulated Annealing
SGA	Standard Genetic Algorithm
SOP	Sum-of-Products
STT	Serial Tabular Technique
VLSI	Very Large Scale Integrated
VPR	Versatile Placement and Routing
WLM	Wire Load Model
XOR	Exclusive-OR operation
XNOR	Inclusive-OR operation

List of Symbols

a_j	CSOP coefficients
b_j	RM coefficients
c_j	CPOS coefficients
d_j	COC coefficients
$\mathcal{F}$	Reed-Muller expansion
$i, i1, i2, i3$	index of variables, $0 < i1 < i2 < i3 < i \leq n-1$
j	index of coefficients, $0 \leq j \leq 2^n - 1$
$j1, j2$	index of coefficients, $0 < j1 < j2 < j$
$j3, j4$	index of coefficients, $j2 < j3 < j4 < j$
n, out	the number of input variables and outputs
p, P	polarity for fixed and mixed expansion
π_j	CSOP minterm
M_j	CPOS maxterm
S_j	COC maxterm

$\mathcal{O}$	on-set coefficient set
$\mathcal{T}$	on-set coefficient table
U	the number of sub-tables in $\mathcal{T}$
$\ddagger$	the Kronecker matrix sum operator
Θ	the matrix multiplication operator
$\prod$	the AND operator
$\odot\prod$	the XNOR operator
$\sum$	the sum operator
$\oplus\sum$	the XOR operator
$[\,]$	the integer operator
t	the transpose operator
$\bigwedge$	AND bitwise operator
$\ll$	LEFT SHIFT bitwise operator
$\bigoplus$	XOR bitwise operator
$\phi(\eta)$	the number of "0"s in η
q	the possible maximum number of on-set coefficients in one segment
w	the number of segments
T	temperature
T_n	transformation matrix for the n-variable function

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

In order to reduce the complexity of design process in modern Integrated Circuit (IC) chips, the typical Computer Aided Design (CAD) system for very large scale integration design consists of several intermediate abstraction, such as logic synthesis, placement and so on. Hence, it enables designers to work progressively down from an abstract level of the design to the layout level.

In the logic synthesis process, Reed-Muller (RM) representation has drawn increasing attention because the AND/XOR realisation of the circuits require less layout area than their AND/OR counterparts in many applications [6,101,102]. Furthermore, in some cases, AND/XOR PLAs require fewer product terms than AND/OR PLAs [100,120]. Therefore, methods for RM expansions are important alternatives to the traditional Canonical Sum-of-Products (CSOP) and Canonical Product-of-Sums (CPOS) approaches to implement Boolean functions. The RM forms offer designers the opportunity to optimise functions that are difficult to simplify in the standard Boolean domain

and a large number of alternative representations. Canonical OR-Coincidence (COC) expansions [34, 51, 140], where the Boolean function is expressed in OR/XNOR form, are also known as Dual Forms of Reed-Muller (DFRM) [61]. Appendix A shows how a 3-variable function $f(x_2, x_1, x_0)$ can be better optimised in the OR/XNOR forms.

XOR/XNOR gates have the disadvantage of low speed and large area consumption. As the Field Programmable Gate Array (FPGA) technology has made significant progress in recent years, XOR/XNOR gates can be implemented into Look-up Tables (LUTs), resulting in XOR/XNOR gates that are as fast as other gates.

FPGA was firstly introduced in 1985 [27]. In the past 20 years, FPGAs have gained increasing popularity in implementing low volume digital circuits. Particularly when the process geometry has shrunk to 90nm process technology, the logic capacity has significantly increased up to 2.2 million Application Specific Integration Circuit (ASIC) gates in a single device and internal clock frequency rate has reached up to 500 MHz. Generic symmetrical FPGA architecture consists of routing resources and configurable blocks [97], in which routing resources occupy 70-90% of FPGA area [23], therefore efficient Placement and Routing (P&R) are essential.

1.2 Research Objectives

The objectives of the research are as follows.

1. Develop various methods and algorithms for efficient conversion between standard Boolean and DFRM forms,

2. Develop an efficient mapping method to find a good Multi-level Mixed

Polarity Reed-Muller (MMPRM) representation from Fixed Polarity Reed-Muller (FPRM) forms,

3. Symmetrical FPGA placement is studied and algorithms are developed and evaluated using Genetic Algorithm (GA) and GA with Simulated Annealing (SA).

1.3 Thesis Organisation

The rest of thesis is organised as follows.

1. In Chapter 2, background information on VLSI design process, VLSI layouts and CAD system of FPGA design and algorithms are given.

2. In Chapter 3, basic definition and terminology of COC expansions and transformation matrix of COC expansions are given. Also in Chapter 3, two map techniques are presented for conversion between standard Boolean and COC expansions of any polarity.

3. Chapter 4 proposes two algorithms for large functions to overcome the limitation of map techniques in Chapter 3.

4. Chapter 5 proposes two tabular techniques, Serial Tabular Technique (STT) and Parallel Tabular Technique (PTT), for large functions.

5. On-set table method is given in details in Chapter 6 for optimisation of MMPRM expansions.

6. In Chapter 7 symmetrical FPGA placement algorithms are developed by using GA and GA with SA respectively.

7. Conclusions and further work are then given in Chapter 8.

Chapter 2

Backgrounds

2.1 VLSI Design Process

The complexity of modern circuits is of the order of millions of transistors. Therefore the design of a VLSI circuit is understandably a complex task. In order to reduce the complexity of design process, several intermediate levels of abstractions are introduced. A top-down design methodology divides the whole design process into 6 phases, as shown in Figure 2.1.

These phases are summarised as following:

1. Design specification:

   Several important factors are to be considered, which are

   - the required performance of the system,
   - the architecture of the system,
   - the external interface and protocol,
   - the choice of manufacturing technology and
   - the available design tools

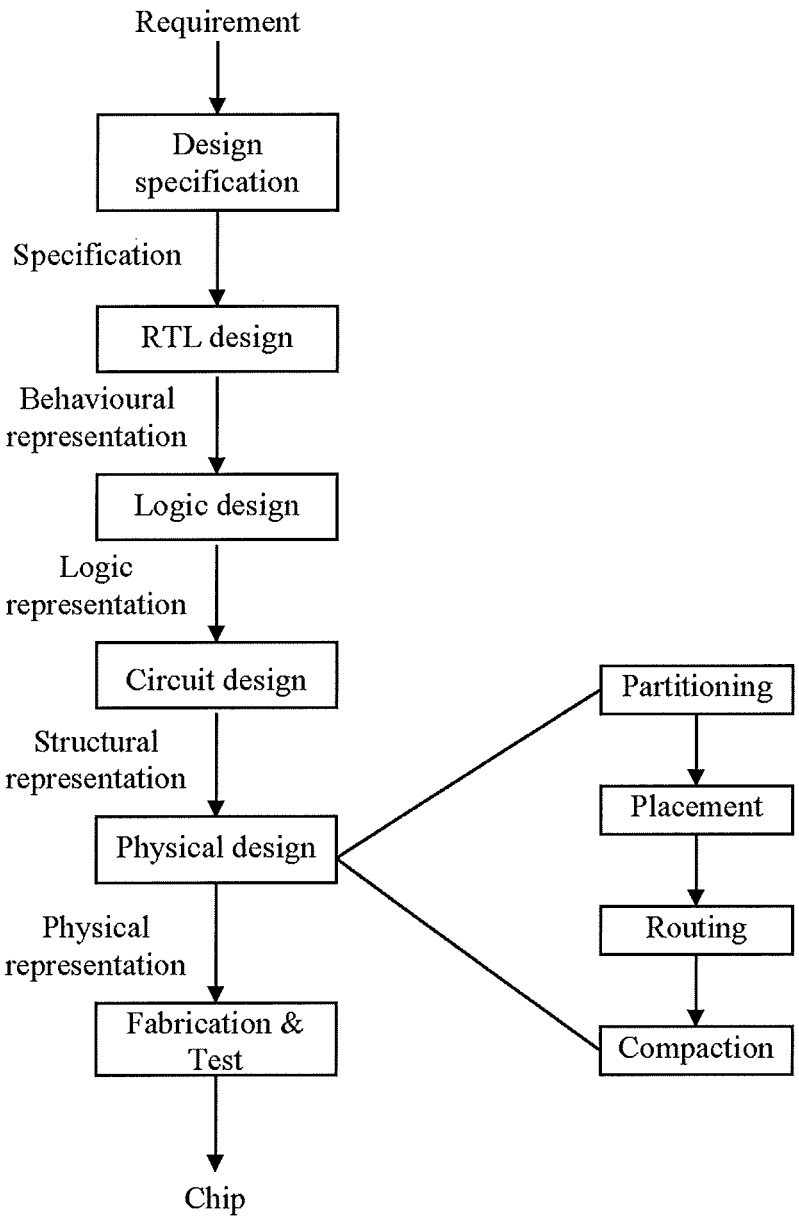


Figure 2.1: General overview of the levels of abstraction.

Furthermore, attentions should be paid to the following issues:

- the design methodology,
- the cost of the design and
- required time to complete the design

2. Architectural design:

   At this level, the behaviour of the system is described in an abstract manner that ignores the low-level details needed. High level abstractions are commonly known as the Register-Transfer Level (RTL). RTL models describe the operation of the system without reference to specific components.

3. Logic design:

   The logic representation that involves in translating the system blocks into a logic model is concerned. These representations are simulated at transistor, gate and register level.

4. Circuit design:

   Logic is represented by basic circuit elements such as resistors, transistors, capacitors and inductors. Transistors are sized to meet signal delay requirement. Analysis and timing verification are performed in this phase to meet signal delay requirement.

5. Physical design:

   The structural representations are transformed into physical package representation that is used in the fabrication of the system. This phase can be further subdivided into 4 steps, which are:

   - partitioning,
   - placement,
   - routing and
   - compaction

6. Fabrication and Testing:

   An actual IC is fabricated using physical package representation. Then manufacturing errors, if any, are determined and eliminated.

However, sometimes in order to achieve better performance, optimisation in several levels of abstraction are considered at the same time, for example a new technology called Wire Load Models (WLMs). They has been available since 1985 for wire load independent logic synthesis and timing closure. Born out of a need to account for the role of interconnect in delay, they have evolved over time to aid in the estimation of chip area and power. One of the principle advantages of the new technology is its ability to harmoniously merge low power and high performance goals. It minimises the coupling between the synthesis and placement and routing stages. Significant improvements in design time and quality should be expected as a result of the faster timing optimisation compared to the combinatorial complexity of standard cell libraries. The one-step timing closure eliminates the design iterations allowing more time for improving the chip layout since timing closure has turned out to be the biggest challenge for high speed sub-micron designs.

2.2 VLSI Layout

The current VLSI layout approaches used to generate physical representations of circuits are:

1. Full-custom,
2. Gate-array,
3. Standard-cell and

4. Macro-cell

These various layout approaches can be grouped to three general categories [99, 107].

1. The full-custom layout approach in which layout elements are hand-crafted in any size and can be placed anywhere on the layout surface.

2. The semi-custom layout approach which imposes some restrictions on the layout elements and surface in order to reduce the complexity of the layout tasks.

3. Universal layout approach which are pre-constructed without any knowledge of the circuit to be laid out in order to reduce the time of design cycle.

2.2.1 Full custom layout

In a full custom design, there are no restrictions on the size and the shape of the logic modules. These logic modules can be shaped to any size as needed, placed at any location of the surface of the chip board and connected in any path that designer wants, as seen in Figure 2.2. Note that the shapes, locations and orientations of logic modules are placed in arbitrary. And also note that the wires connected between logic modules have minimal constraints on their locations.

With the flexibility of the full custom design layout style, a faster and smaller design that efficiently utilises all available spaces on the layout surface results. On the other hand, because of the lack of restrictions in full custom design, the automation complexity becomes extremely high even for a small

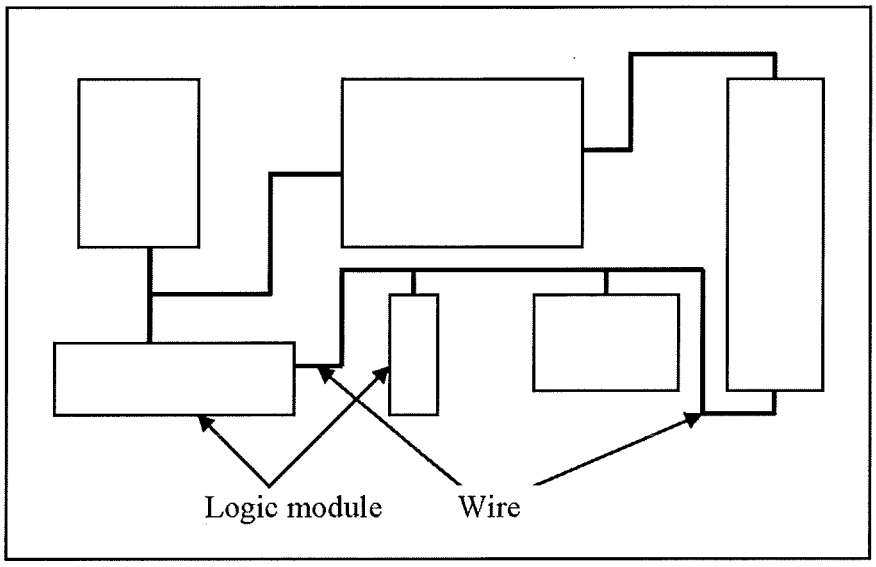


Figure 2.2: Full custom layout.

circuit. In addition, the physical construction of full custom design starts from scratch. It is therefore that large start-up costs are involved in full custom design and entire process is time consuming. As a result, full custom design is only used for the designs in which performance, speed and area, is of the utmost importance, or in which the circuits to be manufactured in large volumes to justify its extreme expense.

2.2.2 Semi-custom layout

In full custom design, since lack of constraints makes synthesis tools difficult to develop, the designer is responsible for layout optimisation and span all levels of abstraction. Therefore it is important to reduce the costs of design and time to market for some other applications without very high performance. With some sacrifice of speed or area, some restrictions can be imposed on the design in order to result in less complex automation than the full custom design does.

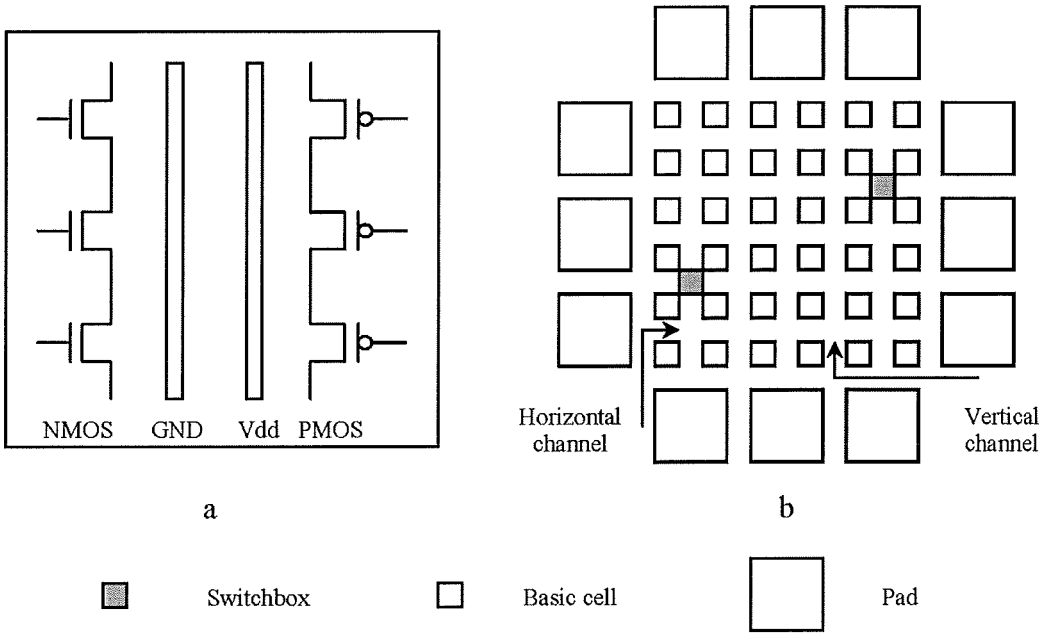


Figure 2.3: Gate array layout. (a) Floor plan of a gate-array. (b) A basic cell structure of a gate array.

There are 3 semi-custom layout styles, standard cell layout, gate-array layout and macro cell layout [107], which are discussed in the following subsections.

Gate array

The gate-array also called Mask Programmable Gate Array (MPGA) is structured as a regular two-dimensional array of basic cells, as seen in Figure 2.3. Each basic cell consists of certain number of uncommitted transistors, which have already been prefabricated on a wafer, as seen in Figure 2.3(a).

Initially the transistors in an array are not connected to one another. In order to realise a circuit on a gate-array, metal connections must be placed using the process of masking, which is called personalisation. There are typically four necessary masking steps in two layers of wiring, one each for the two metal layers and two contact layers. Personalisation involves two types

of interconnections. One is intra-cell wiring and the other is inter-cell wiring. Intra-cell wiring is carried out in a basic cell of a gate-array that implements small circuit module. For example, a two-input NAND gate can be implemented by connecting a group of transistors in a basic cell of a gate array. Thus intra-cell wring is independent of the circuit being implemented on the gate-array. However, inter-cell wiring is carried out in the horizontal channels and vertical channels, as shown in Figure 2.3(b).

Due to the limited routing space, if the connected basic cells are placed close together to avoid long inter-cell wiring, the track density of the channel will exceed its capacity. As a result, the local congestion makes the layout unrouteable. Although routing phase is big challenge in the gate-array layout, it takes a very short time to get a gate-array chip fabricated because all the processing other than personalisation is identical to all gate-arrays, regardless of the circuit to be implemented. So gate-array layout is suitable for prototyping and low-volume product.

A special case of the gate array layout is when routing channels are virtually absent. As a result, the chip consists of a closely packed array of transistors. Wire must therefore be routed over the transistors. This kind of layout is known as channel-less gate arrays and called sea-of-gates.

Standard cell

A standard cell, known also as a poly-cell, is a logic block that performs a standard function. Examples of standard cells are two-input NAND gate, two-input XOR gate, D flip-flop and so on. A cell library is a collection of information pertaining to standard cells. The relevant information about a cell consists of the name of the cell, its functionality, its pin structure and a

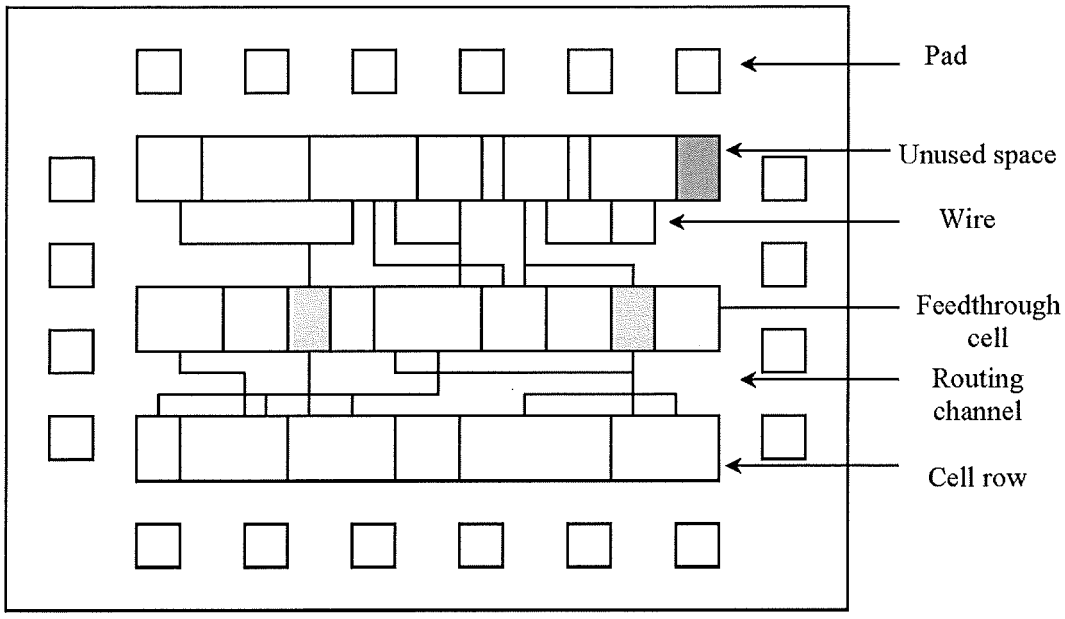


Figure 2.4: Standard cell layout.

layout for the cell in a particular technology such as 0.35 μm CMOS cells in the same library have standardised layouts, that is, all cells are constrained to have the same height.

In cell-based design, as shown in Figure 2.4, the layout is divided into several numbers of rows. Each row consists of cells placed next to each other. Since all the cells are pre-designed to have the same height, the height of a row is the same as the height of any cell in the row. The horizontal routing channels separate rows. Cells within the same row or cells from two facing rows can be interconnected by wire segment through the adjacent channel. If two cells in non-adjacent rows have to be connected, a more elaborate technique called feedthrough cell shown in Figure 2.4 is employed.

One of the advantages of the standard cell layout is that because many of the cell functions are in common in many designs, the circuit design phase can be completed rapidly by the reuse of pre-designed cell library. Furthermore,

since cell layouts are already available, the consideration of a layout will only be the location of each cell and interconnection of the cells. Consequently, the routing phase is typically greatly simplified. However, the loss of flexibility leads to slower solutions and larger circuits than full custom design does.

Macro cell

Both gate-array design and standard-cell design impose restrictions on the cells that are used to design the circuit. These restrictions, on one hand, simplify the flexibility of layout in full custom design and also significantly reduce the design cycle. But on the other hand, the gate-array design prohibits structure other than standard digital logic; namely, more complex logic function such as memory and analogous elements cannot be implemented in gate-array layout. Standard cell design is less restrictive than gate-array, but it is still too expensive to implement memory intensive functions such as RAMs and ROMs or regular functions such as datapaths and arithmetic logic units. The macro cell design fills in the gap. Since there is no restriction on size and shape of the cell in the macro-cell design, the cell can be designed to have efficient layout characteristics for complex logic function. These cells can be accommodated in the library. Macro-cell design comes closest to full custom design. It is therefore placement and routing phase is much more difficult compared to gate-array design and standard cell design.

2.2.3 Universal layout

As mentioned above, the design process is very time consuming and costly in full custom design. Alternatively, the design can be implemented in the semi-custom design with sacrifice of the speed and area efficiency to reduce time

cycle. But regardless of any alternative, there still exists time delay between the completion of the layout design and the physical completion of the circuit. Moreover, the layout can only be tested after physical completion of the circuit, which results in extra time delay.

In the universal layout, the chip has already been fabricated. There is no delay between the completion of the layout design and the physical completion of the circuit at all, which leads a very efficient design cycle. The programmable components are placed in a uniform pattern. The routing channels are also pre-fabricated separately around programmable components, which is very similar to MPGA layout. The routing resources are even more limited than semi-custom design does. What the designer needs to do is to pick an appropriate device firstly, then interconnect routing recourses between programmable components to implement the logic function. But the designer has to face the challenge of assigning programmable components to the locations in which routing can be completed within limited routing resources and also minimising the length of connections.

One typical example of universal circuitry is FPGA. FPGA have become very popular in industry because it is so economic to fabricate and has efficient design cycle. It is an ideal architecture for applications in which speed and area are not extremely important. And also it is ideal for prototyping circuit design. Because the test can be carried out before physical completion of the circuit, the different circuit designs can be carried out on the FPGA for design and testing. The selected design can then be implemented on a more sophisticated architecture.

2.3 CAD of FPGA Design and Algorithms

In 1985 Xilinx Inc. introduced the first LUT based FPGA [27]. FPGAs are more flexible and complex than other programmable devices such as Programmable Logic Array (PLA) and Programmable Array Logic (PAL). With the rapid improvements in the performance and logic densities of the FPGAs, the number of applications where they can be used is increasing. Thus FPGAs are used to implement various complex logic circuits.

The Xilinx FPGA [134, 135] consists of Configurable Logic Blocks (CLBs) which typically contain either combinational or sequential logic circuit, Input/Output Blocks (IOBs) and routing resources such as wire segments and programmable switches. Programmable switches configure the wire segments between logic blocks and between CLBs and IOBs. In order to achieve efficient density and speed of FPGA, several CLBs are grouped together into one block called cluster-based CLB [81, 82]. Compared with MPGAs, FPGAs have the following advantages:

1. Lower Non-Recurring Engineering (NRE) charges incur.
2. The product development time decreases substantially and enhancements and modifications are made much easier.
3. The cost of change for a design is small and also errors in design can be easily corrected before physical completion of the chip.
4. In circuit reprogramming for certain programming technology, is permitted.

FPGAs have some drawbacks that are summarised as follows.

1. FPGAs are roughly three times slower than MPGAs [23].

2. The logic density of an FPGA is about a factor of eight to twelve times less than that of MPGA [23] resulting in a lower yield per wafer.

The typical CAD system for FPGAs, as illustrated in Figure 2.5 resembles that in VLSI design, which consists of several intermediate abstractions summarised as follows.

1. Logic synthesis:

   Boolean equations are optimised so as to optimise area, delay, power dissipation or a combination and converted to logic cells which can be implemented on an FPGA.

2. Placement and routing:

   The specific location of logic blocks and I/O blocks on an FPGA are selected for each logic cell and then routing resources are utilised to connect those logic cells. In this phase, timing and routing constraints must be taken into consideration.

3. Programming of the FPGA:

   On successful completion of the placement and routing, the programming unit configures an FPGA to implement the desired digital logic function.

2.3.1 Logic optimisation

The input to the CAD system is a functional description, which is usually expressed in standard Boolean equations and can also be represented in other forms such as schematic diagrams and Hardware Description Language (HDL).

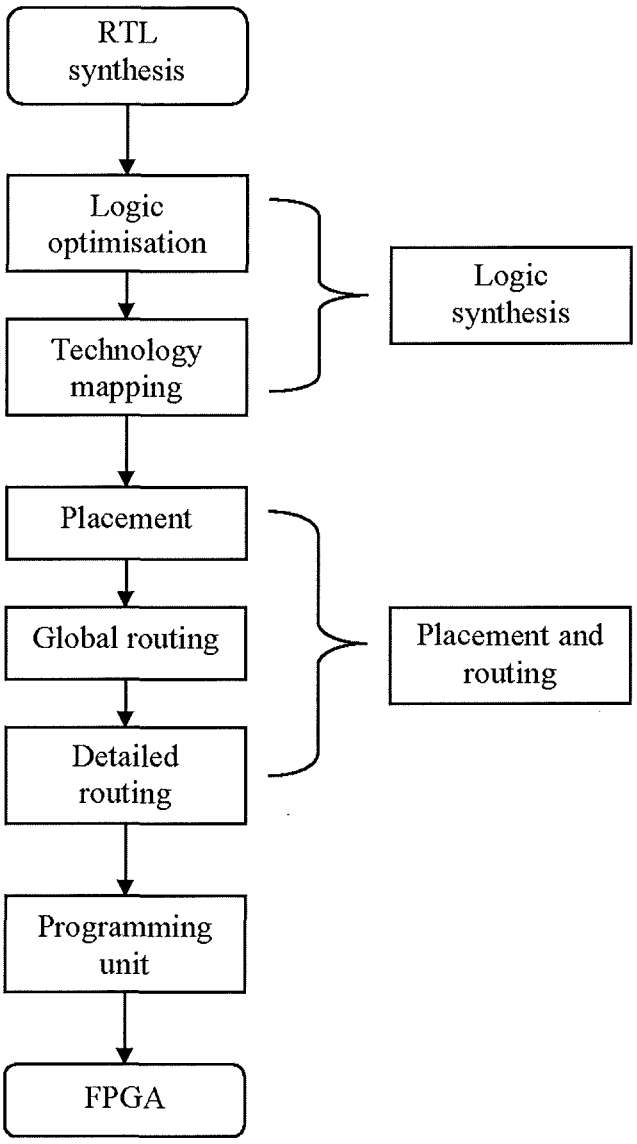


Figure 2.5: A Typical FPGA CAD system.

The aim of logic optimisation is to optimise area, speed or a combination of area and speed by removing redundant logic expression or using another method representing equivalent Boolean equations. Because it does not consider the type of the element that will be used for target circuit, logic optimisation is also known as technology independent optimisation.

Any n-variable Boolean function can be expanded by Shannon expansion

based on AND/OR operations as follows.

$$f(x_{n-1}, x_{n-2}, \cdots, x_0) = \bar{x}_i f_{x_i=0} + x_i f_{x_i=1} \tag{2.1}$$

where $f_{x_i=0}$ and $f_{x_i=1}$ are called the cofactors of $f(x_{n-1}, x_{n-2}, \cdots, x_0)$ with respect to x_i and $0 \leq i \leq n-1$.

Various techniques used for logic optimisation were described in combinational logic [20–22, 62], sequential logic [14, 117] and multi-level simplification [122].

Alternatively, any Boolean function can be represented based on AND/XOR operations, which is called Reed-Muller expansion [90,96]. In contrast to (2.1), there are three basic expansions using AND/XOR operations, which are shown in (2.2) to (2.4). Equations (2.3) and (2.4) are called positive Davio expansion and negative Davio expansion respectively.

$$f(x_{n-1}, x_{n-2}, \cdots, x_0) = \bar{x}_i f_{x_i=0} \oplus x_i f_{x_i=1} \tag{2.2}$$

$$f(x_{n-1}, x_{n-2}, \cdots, x_0) = f_{x_i=0} \oplus x_i (f_{x_i=0} \oplus f_{x_i=1}) \tag{2.3}$$

$$f(x_{n-1}, x_{n-2}, \cdots, x_0) = f_{x_i=1} \oplus \bar{x}_i (f_{x_i=0} \oplus f_{x_i=1}) \tag{2.4}$$

By using (2.3) and (2.4) for each variable in function $f(x_{n-1}, x_{n-2}, \cdots, x_0)$, several classes of RM expansions [60,101] can be obtained and shown as follows.

Fixed polarity Reed-Muller expansion

If (2.3) is used recursively to function $f(x_{n-1}, x_{n-2}, \cdots, x_0)$, an expression consisting of positive literals is obtained, in which all variables appear in true form. The expansion is called a Positive Polarity Reed-Muller (PPRM) expansion. If (2.4) is used recursively to function $f(x_{n-1}, x_{n-2}, \cdots, x_0)$, an expression consisting of negative literals is obtained, in which all variables appear in

complemented form. The expansion is called a Negative Polarity Reed-Muller (NPRM) expansion. If either the type (2.3) or the type (2.4) expansion is applied to each variable, FPRM expansion can be obtained, in which variables appear in true form or complemented form but not both. PPRM and NPRM expansions are the special cases of the FPRM expansions.

Kronecker expansion

When either type (2.2), (2.3) or (2.4) expansion is applied to each variable, an expression is called a Kronecker (KRO) expansion, which is more general than FPRM expansion. Kronecker expansions may be termed mixed polarity RM expansions as each expansion variable may appear in both true and complemented forms throughout an expression and must be present in each product terms. A KRO expansion is constructed from an initial switching function by expanding the function for each variable using one of the expansions given in (2.1) to (2.4). There are 3^n possible variations of these three equations. As a result, any n-variable switching function can be represented by a total of 3^n KRO expansions, each of which is a canonical form.

Each KRO expansion may be identified by means of a polarity number P, $0 \leq P \leq 3^n - 1$. The polarity number is the decimal equivalent of the ternary n-tuple $< P_n P_{n-1} \cdots P_1 >$, where P_i is replaced by 0 if x_i is present throughout the KRO expansion, and replaced by 1 if $\bar{x}_i$ is present in the expansion. If the variable is present in both true and complemented forms, namely, x_i and $\bar{x}_i$, then P_i is replaced by 2. The FPRM expansions will correspond to all polarity numbers whose ternary forms consist of only "0"s and "1"s.

Example 2.1. Given $n = 4$ and $P = 75$, hence $P = 75 =< 2210 >$

$$P_3 = 2 \Rightarrow \quad x_3, \bar{x}_3$$
$$P_2 = 2 \Rightarrow \quad x_2, \bar{x}_2$$
$$P_1 = 1 \Rightarrow \quad \bar{x}_1$$
$$P_0 = 0 \Rightarrow \quad x_0$$

Pseudo Kronecker expansion

When either type (2.2), (2.3) or (2.4) expansion is applied to f, two sub-functions are obtained. A more general expansion than a KRO expansion is obtained when either type (2.2), (2.3) or (2.4) expansion is applied to each sub-function, assuming that different expansions for each sub-function is used. This is called a Pseudo Kronecker (PSDKRO) expansion. There are 3^{2^n-1} possible variations of PSDKRO expansions.

Pseudo Reed-Muller expansion

When either type (2.3) or (2.4) expansion is applied to f, two sub-functions are obtained. A more general expansion than a FPRM expansion is obtained when either type (2.3) or (2.4) expansion is applied to each sub-function, assuming that different expansions for each sub-function is used. This is called a Pseudo Reed-Muller (PSDRM) expansion. There are 2^{2^n-1} different PSDRM expansions.

Generalised Reed-Muller expansion

If the polarities of the literals are chosen freely in FPRM expansion, a more general expression than a FPRM is obtained, which is called a Generalised Reed-Muller (GRM) expansion. There are 2^{n2^n-1} GRM expansions.

Exclusive-OR sum-of-products expansion

The most general AND-XOR expression is Exclusive-OR Sum-of-Products (ESOP) expansion if arbitrary product terms are combined by XORs. There are up to $3^{n\tau}$ GRM expansions, where τ is the number of the products.

Rather than using standard Boolean representation, various methods in Reed-Muller representation were also proposed. A coefficient map method was introduced for mapping coefficients of a possible polarity RM expansion to find the minimum fixed polarity solution without undertaking exhausting search of all possible expansions [125]. An efficient computer method based on coefficient maps to generate all 2^n sets of generalised RM coefficients of an n-variable Boolean function [17]. Almaini presented a tabular method for generating FPRM expansions for a given polarity vector from CSOP expansions [5]. Tabular techniques were also reported for mixed polarity RM such as KRO expansions in [10, 87] and PSDRM expansions [70]. Other methods based on tabular technique were presented in [86, 108, 140].

A coefficient matrix method [66] was presented by Harking for the construction of polarity coefficient matrix of RM expansions without matrix multiplication. The advantage of this method is that the computation of the coefficients of RM expansion for a given polarity is possible without construction of the whole matrix, resulting in less memory storage. Lui and Muzio [80] identified fixed polarity modulo-2 canonical expansions and fixed-biased modulo-2 canonical expansions. Algorithms which perform fast matrix transforms were presented and employed to derive fixed polarity and fixed basis expansions efficiently. Other coefficient-matrix methods were reported in [65, 66, 79].

Falkowski and Chang [49] presented an low complexity and non-exhaustive algorithm for finding optimal FPRM expansions of logic functions using Walsh

spectra of logic function to provide information on polarity matrices of FPRM expansion. Falkowski and Perkowski [50] presented a technique, in which each product term of the initial disjoint Sum-of-Products (SOP) expression is expanded to represent the equivalent product terms of the given polarity FPRM expression. Duplicate product terms must then be located and deleted before the final FPRM expression is realised.

Graph-based algorithm such as Ordered Binary Decision Diagram (OBDD) and Ordered Function Decision Diagram (OFDD) methods have been proposed for the optimisation of RM expansions [42,43,95,111]. However, the size of the graph representation of the function is highly sensitive to the variable ordering. To find the optimal variable ordering for a Binary Decision Diagram (BDD) that minimises the size of the graph is an NP-complete problem due to the larger number of permutations involved.

Other methods were proposed for optimisation of FPRM expansions in [11,69,89,112,115,118,136] and more recently in [109,119,121,128,130] and Mixed Polarity Reed-Muller (MPRM) expansions in [16,40,120,131,133,136]. In addition, an IC chip can be designed for the optimisation of generalised FPRM expansions [13].

2.3.2 Technology mapping

After the Boolean equations have been optimised, the optimised network is then fed to a program called technology mapping, which transforms Boolean network to a certain internal form and then uses an internal optimisation function to group simple logic functions together into logic blocks that can be placed onto FPGA logic blocks. As it performs optimisation that is directly dependent upon the particular technology, e.g., technologies used be-

tween LUT-based FPGA and multiplexer-based FPGA are different. This is also referred to as a technology dependent optimiser. The technology mapping is responsible for minimising either the total number of logic cells required to realise the desired functionality, i.e. area optimisation, or the number of logic cells in time-critical paths, i.e. delay optimisation, and also determining the distribution of pins for each net. As a result, the overall quality of an electronic design automation system depends heavily on the optimisation function used by the technology mapping. Some early technology mapping algorithms can be found in [32, 35, 36, 52–54, 124]. And more recent works [37, 38, 67, 72, 98] have been published in the literature.

2.3.3 Placement

Upon completion of the technology mapping phase, the network of logic function is in a particular format of FPGA logic block, then those logic blocks are assigned a particular place on a FPGA. Part of the goal of placement is to predict the quality of the final routing result so as to improve the routability and performance of final routing. The aim of the P&R tool in an FPGA is to utilise prefabricated programmable routing switches and routing channels in an FPGA as less as possible to achieve 100% successful P&R. FPGA placement is categorised to be NP-complete. Numerous methods have been proposed to solve placement problem such as GA, SA and so forth.

Traditionally, researches on placement can be classified into two categories:

1. Partitioning based placement, such as bi- and quadri- section min cut,

2. Iterative search strategies, such as SA and GA.

The partitioning based placement tools cuts chip area into two sections.

The process is applied recursively to each partition until the area is small enough to complete the placement easily.

Simulated annealing is a heuristic based algorithm that is analogous to the annealing process in metals. Initially, the temperature is set to a high value so that all moves that result in a reduction in cost are accepted and then the temperature is gradually reduced to zero. For each temperature, two blocks are randomly selected first to generate new positions and then interchanging their positions. The moves that result in a reduction in cost are accepted. On the other hand, some of the moves that lead to increase in cost are also accepted. The probability of acceptance of moves that result in an increase in cost depends on the increase in cost and the temperature. At high temperature, the probabilities of acceptance of moves that increase the cost are high. As the temperature gradually decreases, the moves that result in increase in the cost therefore have less chance to be accepted. Ultimately, at very low temperature or even zero temperature only moves that result in reduction in the cost are accepted.

SA has achieved a tremendous success in solving VLSI/CAD problems after rigorous mathematical models were formulated to explain the behaviour of the algorithm and how to select the proper tuning parameters [84]. However, the main disadvantage of simulated annealing is that the best results are obtained by tuning many controlling parameters [106]. Moreover, SA requires a large amount of CPU time and memory.

The widely used and highly successful placement tool for standard cell TimberWolf package [104] and symmetrical FPGA Versatile Placement and Routing (VPR) package [18, 83] are all based on the SA. Recently developed fast FPGA placement techniques [44, 45] are still based on SA.

The other powerful search strategy, genetic algorithm, is also widely used recently, which works by emulating the natural process of evolution as a mean of progressing towards the global optimum.

2.3.4 Global routing

Although several levels of abstraction are introduced to reduce the complexity of design, the complexity of routing phase is still very high. To further reduce the complexity in the final routing phase, global routing is sometimes used before detailed routing. As is known, the routing resources are already prefabricated and very limited in the FPGA. In order to achieve less expensive and faster implementation, global routing tries to balance the channel congestion and minimize the channel density, and also reduce the propagation delay for each net. Note that in global routing the exact wire segments are not chosen yet. Techniques for independent global routing has been published in the literature [28, 29, 74, 110]. However, there are many CAD tools [1, 2, 18, 30] that perform placement and global routing simultaneously.

2.3.5 Detailed routing

In detailed routing, each net is assigned to a particular wire segment within a given channel. Only when every net is fit into given width channel of FPGA, detailed routing is successful. As we have known, connections between wire segments are made by "fusing" the programmable switches. The programmable switches consume significant area of FPGA. In order to achieve area efficiency, none of architecture of switch block and architecture of connection block is made universal connection. In addition, the architecture of switch block and architecture of connection block are taken into consideration neither in the

placement phase nor global routing phase. Therefore, some of the wire segments within a given channel, which is made from global routing phase, are impossible to be connected through the switch block and connection block. Detailed routing then will adjust channel width predicated in the global routing to achieve successful routing.

Given same channel width and architecture, some FPGA designs may only have a very limited number of ways or even only one way to connect all of the nets, while some other designs may have significant number of ways to connect all of the nets within sparse routing resources. The primary concern in the routing phase of FPGA design is to find a complete routing that legally connects all nets to realise logic functionality. When a complete routing of the nets can be accomplished in various ways, other performance criteria then become important.

A smaller FPGA with a small channel width is typically less expensive and has better performance than a larger FPGA. Detailed routing aims mainly to minimise the overall channel width required to route all nets. Because of the large parasitic capacitance and resistance of programmable switches, it takes significant amount of time to propagate a signal from the source of the net to its most distant sink. Furthermore, a net that is spread out across the entire FPGA typically takes longer to realise its functionality than a net with a smaller length. To achieve high performance of FPGA design, minimisation of the net length of the various nets or the delay of the net becomes essential.

Numerous notable routing algorithms have been proposed to achieve successful routing including the CGE router [24] and the SEGA Router [73], one two-step router [74], the graph-based router [2], auction-based Quark [85] and timing-driven router [31].

2.4 Genetic Algorithms

GA is a search technique based on the mechanisms of natural genetics and natural selection [56], which incorporates a simulation of evolution as a search heuristic when finding a good solution. It operates on a population of individuals that are coded and called chromosomes in the search space instead of one solution. Each individual has some fitness value and is measured by an evaluation function termed fitness function. The approach of the algorithm is to explore the search space and to discover better solutions by allowing the individuals to evolve over time termed generations. The GA uses these individuals in the generation to produce a new generation of hopefully better solutions. In each generation, two of the individuals are selected probabilistically as parents, with the selection probability proportional to their fitness. It is therefore a fitter individual, i.e. possibly containing some useful features, has a higher probability of propagating itself. Crossover with high probability is then performed on these two individuals to generate two new individuals called offspring, by changing parts of their structure. As a result, each offspring inherits a combination of features from both parents. This enables the GA to try out various features in different combinations and see whether the individuals still retain their fitness.

Mutation with small probability is then performed to explore new features that may not be in the population yet. After mutation on an individual, it not only has just the combination of the features inherited from its two parents, but also incorporates this additional change caused by mutation. This ensures the GA does not converge to local optimum. The main advantage lies in the robustness of search and problem independence [56]. Therefore, it has been successfully applied to many optimisation problems such as FPGA

placement [137,138], Reed-Muller Binary Decision Diagram (RMBDD) [7,9,12, 111], Finite State Machine (FSM) [129,132], state assignment [8,126], evolvable hardware [3, 4], compact test pattens for decision diagram [26], minimisation of FPRM expansions [41,42,89,136], minimisation of MPRM expansions [111], standard cell placement [78,84,104–106], macro cell placement [46,48,84,103], ratio-cut partitioning problem on hyper-graphs [25], channel routing [55, 75–77,91–94], switch-box routing [77], over-the-cell routing [57,59], power driven over-the-cell routing [58].

The GA has to evaluate the population of individuals in every generation which results in long CPU time, especially in the later part of the process. Simulated annealing, which mimics the annealing process in metal, is also a widely used technique because of its fast convergence. A combination of the GA and simulated annealing algorithms were reported for traveling salesman problem [143], macro cell placement [47], non-slicing floorplan design [71] and symmetrical FPGA placement [139,141].

2.5 Summary

In this chapter, top down design methodology and existing full custom and semi custom layouts of VLSI design process are illustrated. FPGA design has its own design flow, which is very similar to that of VLSI design. Optimisation algorithms for each level of abstraction are reviewed. Moreover, because GA as one of evolutionary computation algorithms is one of main research objectives, the procedure of standard GA is given and the various applications in VLSI design based on GA are reviewed as well.

Chapter 3

Map Techniques for Dual Forms of RM Expansions

3.1 Introduction

In logic synthesis, there are some circuits that can not be minimised in the standard Boolean domain, but might be optimised well in the Reed-Muller forms. Reed-Muller forms increase the search space of optimisation. Extensive research had been carried out in the Reed-Muller forms based on the AND/XOR logic. However, sometimes the circuits can be better simplified in dual forms of Reed-Muller implemented in OR/XNOR logic, as shown in Appendix A. The dual forms of RM expansion is based on the features of coincidence operation. Hence the expansion is referred as Canonical OR Coincidence expansion. In this chapter, graphical techniques are presented for conversion between standard Boolean and fixed polarity COC expansions of any polarity. The rest of the chapter is organised as follows. In Section 3.2, the basic definitions and preliminary of fixed polarity COC expansions are

given. The transformation matrix of fixed polarity COC expansions is given in Section 3.3. Map techniques using CPOS maxterms and CSOP minterms for conversion of fixed polarity COC expansion of any polarity are shown in Section 3.4.

3.2 Preliminaries

3.2.1 Basic definitions

Definition 3.1. a_j, c_j and d_j are coefficients of CSOP, CPOS and COC expansions respectively, where $0 \leq j \leq 2^n - 1$.

The COC expansion is based on the XNOR and OR operations. The XNOR has the following properties:

$$\begin{aligned} 0 \odot 0 &= 1 \\ 0 \odot 1 &= 0 \\ 1 \odot 0 &= 0 \\ 1 \odot 1 &= 1 \\ x \odot x &= 1 \\ x \odot 1 &= x \\ x \odot 0 &= \bar{x} \\ x \odot y &= \bar{x} \odot \bar{y} \end{aligned} \tag{3.1}$$

From (3.1), it can be seen that the XNOR operation result is 0 when the number of zeros is odd otherwise the XNOR operation result is 1.

Definition 3.2. $\eta = \bigodot\prod_{j=0}^{2^n-1} c_j$, where $\bigodot\prod$ is the XNOR operator, $c_j \in \{0,1\}$. The number of "0"s involved in the calculation of η is denoted by $\phi(\eta)$. For example, $\phi(\eta) = 3$ if $\eta = 1 \odot 0 \odot 0 \odot 0$, and $\eta = 0$ because $\phi(\eta) = 3$ is odd number.

Definition 3.3. The Kronecker matrix sum is defined as

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \cdots & \mathbf{A}_{1(y-1)} & \mathbf{A}_{1y} \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \cdots & \mathbf{A}_{2(y-1)} & \mathbf{A}_{2y} \\ & & \vdots & & \\ \mathbf{A}_{(x-1)1} & \mathbf{A}_{(x-1)2} & \cdots & \mathbf{A}_{(x-1)(y-1)} & \mathbf{A}_{(x-1)y} \\ \mathbf{A}_{x1} & \mathbf{A}_{x2} & \cdots & \mathbf{A}_{x(y-1)} & \mathbf{A}_{xy} \end{bmatrix}$$

$$\ddagger \begin{bmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} & \cdots & \mathbf{B}_{1(y-1)} & \mathbf{B}_{1y} \\ \mathbf{B}_{21} & \mathbf{B}_{22} & \cdots & \mathbf{A}_{2(y-1)} & \mathbf{B}_{2y} \\ & & \vdots & & \\ \mathbf{B}_{(x-1)1} & \mathbf{B}_{(x-1)2} & \cdots & \mathbf{B}_{(x-1)(y-1)} & \mathbf{B}_{(x-1)y} \\ \mathbf{B}_{x1} & \mathbf{B}_{x2} & \cdots & \mathbf{B}_{x(y-1)} & \mathbf{B}_{xy} \end{bmatrix} \tag{3.2}$$

$$= \begin{bmatrix} \mathbf{A}_{11}+\mathbf{B}_{11} & \cdots & \mathbf{A}_{11}+\mathbf{B}_{1y} & \cdots & \mathbf{A}_{1y}+\mathbf{B}_{11} & \cdots & \mathbf{A}_{1y}+\mathbf{B}_{1y} \\ & \vdots & & \vdots & & \vdots & \\ \mathbf{A}_{11}+\mathbf{B}_{x1} & \cdots & \mathbf{A}_{11}+\mathbf{B}_{xy} & \cdots & \mathbf{A}_{1y}+\mathbf{B}_{x1} & \cdots & \mathbf{A}_{1y}+\mathbf{B}_{xy} \\ & \vdots & & \vdots & & \vdots & \\ \mathbf{A}_{x1}+\mathbf{B}_{11} & \cdots & \mathbf{A}_{x1}+\mathbf{B}_{1y} & \cdots & \mathbf{A}_{xy}+\mathbf{B}_{11} & \cdots & \mathbf{A}_{xy}+\mathbf{B}_{1y} \\ & \vdots & & \vdots & & \vdots & \\ \mathbf{A}_{x1}+\mathbf{B}_{x1} & \cdots & \mathbf{A}_{x1}+\mathbf{B}_{xy} & \cdots & \mathbf{A}_{xy}+\mathbf{B}_{x1} & \cdots & \mathbf{A}_{xy}+\mathbf{B}_{xy} \end{bmatrix}$$

where $\ddagger$ represents Kronecker matrix sum.

Definition 3.4. The matrix multiplication is defined as

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \cdots & \mathbf{A}_{1(y-1)} & \mathbf{A}_{1y} \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \cdots & \mathbf{A}_{2(y-1)} & \mathbf{A}_{2y} \\ & & \vdots & & \\ \mathbf{A}_{(x-1)1} & \mathbf{A}_{(x-1)2} & \cdots & \mathbf{A}_{(x-1)(y-1)} & \mathbf{A}_{(x-1)y} \\ \mathbf{A}_{x1} & \mathbf{A}_{x2} & \cdots & \mathbf{A}_{x(y-1)} & \mathbf{A}_{xy} \end{bmatrix} \Theta$$

$$\begin{bmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} & \cdots & \mathbf{B}_{1(y-1)} & \mathbf{B}_{1y} \\ \mathbf{B}_{21} & \mathbf{B}_{22} & \cdots & \mathbf{A}_{2(y-1)} & \mathbf{B}_{2y} \\ & & \vdots & & \\ \mathbf{B}_{(x-1)1} & \mathbf{B}_{(x-1)2} & \cdots & \mathbf{B}_{(x-1)(y-1)} & \mathbf{B}_{(x-1)y} \\ \mathbf{B}_{x1} & \mathbf{B}_{x2} & \cdots & \mathbf{B}_{x(y-1)} & \mathbf{B}_{xy} \end{bmatrix} =$$

$$\begin{bmatrix} (\mathbf{A}_{11} + \mathbf{B}_{11}) \odot \cdots \odot (\mathbf{A}_{1y} + \mathbf{B}_{x1}) & \cdots & (\mathbf{A}_{11} + \mathbf{B}_{1y}) \odot \cdots \odot (\mathbf{A}_{1y} + \mathbf{B}_{xy}) \\ (\mathbf{A}_{21} + \mathbf{B}_{11}) \odot \cdots \odot (\mathbf{A}_{2y} + \mathbf{B}_{x1}) & \cdots & (\mathbf{A}_{21} + \mathbf{B}_{1y}) \odot \cdots \odot (\mathbf{A}_{2y} + \mathbf{B}_{xy}) \\ & \vdots & \\ (\mathbf{A}_{x1} + \mathbf{B}_{11}) \odot \cdots \odot (\mathbf{A}_{xy} + \mathbf{B}_{x1}) & \cdots & (\mathbf{A}_{x1} + \mathbf{B}_{1y}) \odot \cdots \odot (\mathbf{A}_{xy} + \mathbf{B}_{xy}) \end{bmatrix} \tag{3.3}$$

where Θ represents matrix multiplication based on XNOR and OR operations.

Example 3.1. Use matrix multiplication based on XNOR and OR operations.

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \Theta \begin{bmatrix} z_3 \\ z_2 \\ z_1 \\ z_0 \end{bmatrix} = \begin{bmatrix} (0+z_3) \odot (1+z_2) \odot (1+z_1) \odot (1+z_0) \\ (0+z_3) \odot (0+z_2) \odot (1+z_1) \odot (1+z_0) \\ (0+z_3) \odot (0+z_2) \odot (0+z_1) \odot (1+z_0) \\ (0+z_3) \odot (0+z_2) \odot (0+z_1) \odot (0+z_0) \end{bmatrix}$$

$$= \begin{bmatrix} z_3 \odot 1 \odot 1 \odot 1 \\ z_3 \odot z_2 \odot 1 \odot 1 \\ z_3 \odot z_2 \odot z_1 \odot 1 \\ z_3 \odot z_2 \odot z_1 \odot z_0 \end{bmatrix} = \begin{bmatrix} z_3 \\ z_3 \odot z_2 \\ z_3 \odot z_2 \odot z_1 \\ z_3 \odot z_2 \odot z_1 \odot z_0 \end{bmatrix}$$

3.2.2 CSOP minterms, CPOS and COC maxterms

Given a truth table for a Boolean function, two standard algebraic forms of the function can be derived, namely CSOP form and CPOS form. CSOP form is in Disjunctive Canonical Form (DCF) and known as minterm expansion while CPOS form is in Conjunctive Canonical Form (CCF) and known as maxterm expansion.

Any n-variable Boolean function can be represented in either CSOP or CPOS expansion form. The CSOP expansion is shown as

$$f(x_{n-1}, \cdots, x_1, x_0) = \sum_{j=0}^{2^n-1} a_j m_j \tag{3.4}$$

where $\sum$ is the OR operator, m_j is the minterm and $0 \leq j \leq 2^n-1$. $a_j \in \{0,1\}$ and "1"s indicate the presence of the corresponding CSOP minterms in the expansion, known as on-set CSOP minterms. There are 2^n CSOP minterms

for an n-variable function. Minterm m_j can be expressed as

$$m_j = \acute{x}_{n-1} \cdots \acute{x}_1 \acute{x}_0 \tag{3.5}$$

$$\acute{x}_i = \begin{cases} \bar{x}_i & j_i = 0 \\ x_i & j_i = 1 \end{cases} \tag{3.6}$$

where $\bar{x}_i$ is the complemented form of x_i , j_i is the ith bit of j, j is in the binary form and $0 \leq i \leq n-1$.

The CPOS expansion is shown as

$$f(x_{n-1}, \cdots, x_1, x_0) = \prod_{j=0}^{2^n-1} (c_j + M_j) \tag{3.7}$$

where $\prod$ is the AND operator, M_j is the maxterm and $0 \leq j \leq 2^n - 1$. $c_j \in \{0, 1\}$ and "0"s indicate the presence of the corresponding CPOS maxterms in the expansion, known as on-set CPOS maxterms. There are 2^n CPOS maxterms for an n-variable function. Maxterm M_j can be expressed as

$$M_j = \acute{x}_{n-1} + \cdots + \acute{x}_1 + \acute{x}_0 \tag{3.8}$$

$$\acute{x}_i = \begin{cases} \bar{x}_i & j_i = 1 \\ x_i & j_i = 0 \end{cases} \tag{3.9}$$

where $\bar{x}_i$ is the complemented form of x_i , j_i is the ith bit of j, j is in the binary form and $0 \leq i \leq n-1$.

Equation (3.7) can be rewritten in fixed polarity COC expansion form as

$$f(\tilde{x}_{n-1}, \cdots, \tilde{x}_1, \tilde{x}_0) = \bigodot \prod_{j=0}^{2^n-1} (d_j^p + S_j) \tag{3.10}$$

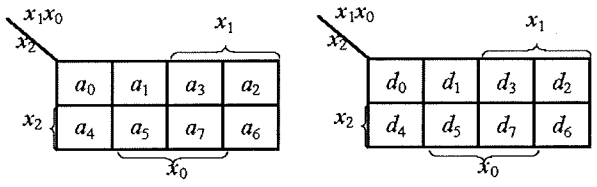
where S_j is the COC maxterm, $d_j^p \in \{0, 1\}$, $0 \leq j, p \leq 2^n - 1$ and "0"s indicate the presence of the corresponding COC maxterms in the expansion, known as on-set COC maxterms. The superscript p of d stands for the number of polarity of the fixed polarity COC expansion. For simplicity, d_j is used instead of d_j^0 if the polarity is zero. There are 2^n COC maxterms. The COC maxterm S_j can be expressed as

$$S_j = \grave{x}_{n-1} + \cdots + \grave{x}_1 + \grave{x}_0 \tag{3.11}$$

$$\grave{x}_i = \begin{cases} 0 & j_i = 1 \\ \tilde{x}_i & j_i = 0 \end{cases} \tag{3.12}$$

where j_i is the ith bit of j, j is in the binary form and $\tilde{x}_i$ can be in the true form or complemented form but not both.

Definition 3.5. For a given n-variable Boolean function $f(x_{n-1}, \cdots, x_1, x_0)$, there are 2^n fixed polarity COC expansions. Accordingly the function is notated as $f(\tilde{x}_{n-1}, \cdots, \tilde{x}_1, \tilde{x}_0)$. When each $\tilde{x}_i$ appears in true form, the polarity is called zero polarity or positive polarity. When each $\tilde{x}_i$ appears in complemented form, the polarity is called $2^n - 1$ polarity or negative polarity. For a 3-variable function $f(\tilde{x}_2, \tilde{x}_1, \tilde{x}_0)$, $f(x_2, x_1, x_0)$ is polarity 0, $f(x_2, x_1, \bar{x}_0)$ is polarity 1, $f(\bar{x}_2, \bar{x}_1, x_0)$ is polarity 6, $f(\bar{x}_2, \bar{x}_1, \bar{x}_0)$ is polarity 7 and so on.



(a) (b)

Figure 3.1: Coefficients maps. (a) A 3-variable c_j map of CPOS expansion, (b) A 3-variable d_j-coefficient. map.

3.2.3 Map folding technique for positive polarity using maxterms

The COC expansion can be obtained from CPOS expansion by using map folding technique [34]. The map used in the COC expansion is referred as d_j map [34] in contrast with b_j map of the Reed-Muller expansion [125].

Example 3.3. Obtain the positive polarity COC expansion by using maxterm map folding technique for a 3-variable function $f(x_2, x_1, x_0) = (x_2 + x_1 + x_0)(x_2 + \bar{x}_1 + x_0)(\bar{x}_2 + x_1 + \bar{x}_0)(\bar{x}_2 + \bar{x}_1 + x_0)$.

The c_j and d_j coefficient map can be drawn as shown in Figure 3.1(a) and Figure 3.1(b) respectively. The on-set CPOS maxterms for a given function are obtained as $f(x_2, x_1, x_0) = \prod(0, 2, 5, 6)$, as shown in Figure 3.2(a).

Step 1:

Fold the map along the x_2 border (i.e. $x_2 = 1$ is folded on $x_2 = 0$) and XNOR the contents of section $x_2 = 0$ of the map. Section $x_2 = 1$ is not affected by the XNOR operation, as shown in Figure 3.2(a), and then unfold the map.

Step 2:

Repeat Step 1 for the remaining variables, as shown in Figure 3.2(b) and Figure 3.2(c).

Example 3.2. When $n = 3$, $f(x_2, x_1, x_0)$ can be represented by CPOS expansion as

$$\begin{aligned} f(x_2, x_1, x_0) &= (c_0 + x_2 + x_1 + x_0)(c_1 + x_2 + x_1 + \bar{x}_0) \\ &\quad (c_2 + x_2 + \bar{x}_1 + x_0)(c_3 + x_2 + \bar{x}_1 + \bar{x}_0) \\ &\quad (c_4 + \bar{x}_2 + x_1 + x_0)(c_5 + \bar{x}_2 + x_1 + \bar{x}_0) \\ &\quad (c_6 + \bar{x}_2 + \bar{x}_1 + x_0)(c_7 + \bar{x}_2 + \bar{x}_1 + \bar{x}_0) \end{aligned}$$

If all the variables appear in complemented form, the negative polarity (polarity 7) COC expansion is given as follows.

$$\begin{aligned} f(\bar{x}_2, \bar{x}_1, \bar{x}_0) &= (d_0^7 + S_0) \odot (d_1^7 + S_1) \odot (d_2^7 + S_2) \odot (d_3^7 + S_3) \\ &\quad \odot (d_4^7 + S_4) \odot (d_5^7 + S_5) \odot (d_6^7 + S_6) \odot (d_7^7 + S_7) \end{aligned}$$

$$\begin{aligned} S_0 &= \bar{x}_2 + \bar{x}_1 + \bar{x}_0 \\ S_1 &= \bar{x}_2 + \bar{x}_1 \\ S_2 &= \bar{x}_2 + \bar{x}_0 \\ S_3 &= \bar{x}_2 \\ S_4 &= \bar{x}_1 + \bar{x}_0 \\ S_5 &= \bar{x}_1 \\ S_6 &= \bar{x}_0 \\ S_7 &= 0 \end{aligned}$$

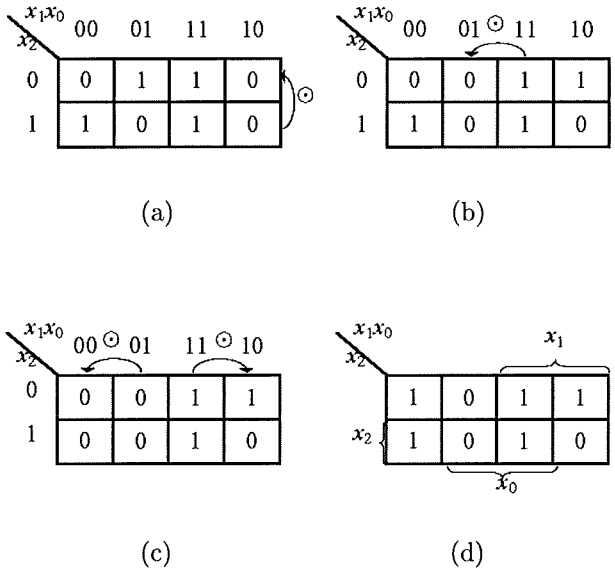


Figure 3.2: Map folding technique for positive polarity using maxterms. (a) Folding the map along the x_2 border, (b) Folding the map along the x_1 border, (c) Folding the map along the x_0 border, (d) Final coefficients map.

x_2 \ x_1x_0	00	01	11	10
0	$x_2+x_1+x_0$	x_2+x_1	x_2	x_2+x_0
1	x_1+x_0	x_1	0	x_0

Figure 3.3: A 3-variable COC expansion of d_j coefficients map.

Step 3:

Read the "0"s from the map as shown in Figure 3.2(d), $f(x_2, x_1, x_0) = \odot\prod(1, 5, 6)$ is obtained. Compared the 3-variable d_j map shown in Figure 3.2(d) and Figure 3.3, the positive polarity COC expansion is therefore obtained as

$$f(x_2, x_1, x_0) = (x_2 + x_1) \odot x_1 \odot x_0$$

3.3 Transformation Matrix for COC Expansions

One way to generate the transformation matrix for positive polarity COC expansion is to derive it from the transformation matrix for RM [34]. It requires the transpose of RM matrix and the replacement of the elements of RM matrix by changing the "0"s to "1"s and "1"s to "0"s. As a result, large computation is involved especially when the number of variables is large. However, there is another easy way [61].

Corollary 3.1. *The relationship between c_j coefficients of CPOS expansion and d_j coefficients of positive polarity COC expansion is*

$$d = [T_n]\Theta c \tag{3.13}$$

where $c = [c_{2^n-1} \cdots c_1\ c_0]^t$, $d = [d_{2^n-1} \cdots d_1\ d_0]^t$ *and*

$$[T_n] = \begin{bmatrix} T_{n-1} & 1 \\ T_{n-1} & T_{n-1} \end{bmatrix} \tag{3.14}$$

$$[T_1] = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \tag{3.15}$$

Proof. Considering the features of the coincidence operation, Kronecker matrix sum and matrix multiplication, equation (3.7) can be rewritten as

$$f(x_{n-1}, \cdots, x_1, x_0) = \{[\bar{x}_{n-1}\ \ x_{n-1}] \ddagger \cdots \ddagger [\bar{x}_1\ \ x_1] \ddagger [\bar{x}_0\ \ x_0]\}\Theta c \tag{3.16}$$

Because $\bar{x}_i = 0 \odot x_i$ and $x_i = 1 \odot x_i$, $[\bar{x}_i\ \ x_i] = [0\ \ x_i]\Theta \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$.

Let $\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} = [T_1]$,

$$\begin{aligned} f(x_{n-1}, \cdots, x_1, x_0) &= \{\{[0 \;\; x_{n-1}] \ominus [T_1]\} \ddagger \cdots \ddagger \{[0 \;\; x_1] \ominus [T_1]\} \\ &\quad \ddagger \{[0 \;\; x_0] \ominus [T_1]\}\} \ominus c \qquad (3.17) \\ &= \{\{[0 \;\; x_{n-1}] \ddagger \cdots \ddagger [0 \;\; x_1] \ddagger [0 \;\; x_0]\} \\ &\quad \ominus \{\underbrace{[T_1] \ddagger \cdots \ddagger [T_1]}_{n}\}\} \ominus c \qquad (3.18) \end{aligned}$$

Equation (3.10) for n-variable positive polarity COC expansion can be rewritten as

$$f(x_{n-1}, \cdots, x_1, x_0) = \{[0 \;\; x_{n-1}] \ddagger \cdots \ddagger [0 \;\; x_1] \ddagger [0 \;\; x_0]\} \ominus d \qquad (3.19)$$

Because (3.16) and (3.19) are equal,

$$\begin{aligned} \{[0 \;\; x_{n-1}] \ddagger \cdots \ddagger [0 \;\; x_1] \ddagger [0 \;\; x_0]\} \ominus d &= \{[0 \;\; x_{n-1}] \ddagger \cdots \ddagger [0 \;\; x_1] \ddagger [0 \;\; x_0]\} \\ &\quad \ominus \{\underbrace{[T_1] \ddagger \cdots \ddagger [T_1]}_{n}\}\} \ominus c \end{aligned}$$

As a result,

$$d = \left\{\underbrace{[T_1] \ddagger \cdots \ddagger [T_1]}_{n}\right\} \ominus c$$

Hence, $d = [T_n] \ominus c$, where

$$[T_n] = \begin{bmatrix} T_{n-1} & 1 \\ T_{n-1} & T_{n-1} \end{bmatrix} = \left\{\underbrace{[T_1] \ddagger \cdots \ddagger [T_1]}_{n}\right\} \ominus c$$

□

Example 3.4. Given a 3-variable $f(x_2, x_1, x_0)$ function, derive the relationship between c_j coefficients of CPOS expansion and d_j coefficients of positive polarity COC expansion.

The 3-variable function $f(x_2, x_1, x_0)$ can be rewritten as in the following two equations.

$$\begin{aligned}
f(x_2, x_1, x_0) &= \{[0 \quad x_2] \ddagger [0 \quad x_1] \ddagger [0 \quad x_0]\} \Theta d \\
&= \{[0 \quad x_1 \quad x_2 \quad x_2 + x_1] \ddagger [0 \quad x_0]\} \Theta d \\
&= \{[0 \quad x_0 \quad x_1 \quad x_1 + x_0 \quad x_2 \quad x_2 + x_0 \\
&\qquad x_2 + x_1 \quad x_2 + x_1 + x_0]\} \Theta d \\
&= (d_7 + 0) \odot (d_6 + x_0) \odot (d_5 + x_1) \\
&\quad \odot (d_4 + x_1 + x_0) \odot (d_3 + x_2) \odot (d_2 + x_2 + x_0) \\
&\quad \odot (d_1 + x_2 + x_1) \odot (d_0 + x_2 + x_1 + x_0)
\end{aligned}$$

$$\begin{aligned}
f(x_2, x_1, x_0) &= \{[\bar{x}_2 \quad x_2] \ddagger [\bar{x}_1 \quad x_1] \ddagger [\bar{x}_0 \quad x_0]\} \Theta c \\
&= \{[\bar{x}_2 + \bar{x}_1 \quad \bar{x}_2 + x_1 \quad x_2 + \bar{x}_1 \quad x_2 + x_1] \ddagger [\bar{x}_0 \quad x_0]\} \Theta c \\
&= \{[\bar{x}_2 + \bar{x}_1 + \bar{x}_0 \quad \bar{x}_2 + \bar{x}_1 + x_0 \quad \bar{x}_2 + x_1 + \bar{x}_0 \\
&\qquad \bar{x}_2 + x_1 + x_0 \quad x_2 + \bar{x}_1 + x_0 \quad x_2 + \bar{x}_1 + x_0 \\
&\qquad x_2 + x_1 + \bar{x}_0 \quad x_2 + x_1 + x_0]\} \Theta c \\
&= (c_7 + \bar{x}_2 + \bar{x}_1 + \bar{x}_0) \odot (c_6 + \bar{x}_2 + \bar{x}_1 + x_0) \\
&\quad \odot (c_5 + \bar{x}_2 + x_1 + \bar{x}_0) \odot (c_4 + \bar{x}_2 + x_1 + x_0) \\
&\quad \odot (c_3 + x_2 + \bar{x}_1 + \bar{x}_0) \odot (c_2 + x_2 + \bar{x}_1 + x_0) \\
&\quad \odot (c_1 + x_2 + x_1 + \bar{x}_0) \odot (c_0 + x_2 + x_1 + x_0)
\end{aligned}$$

Thus

$c_7 = f(111) = d_7 \odot 1 \odot 1 \odot 1 \odot 1 \odot 1 \odot 1 \odot 1 = d_7$

$c_6 = f(110) = d_7 \odot d_6 \odot 1 \odot 1 \odot 1 \odot 1 \odot 1 \odot 1 = d_7 \odot d_6$

$c_5 = f(101) = d_7 \odot 1 \odot d_5 \odot 1 \odot 1 \odot 1 \odot 1 \odot 1 = d_7 \odot d_5$

$c_4 = f(100) = d_7 \odot d_6 \odot d_5 \odot d_4 \odot 1 \odot 1 \odot 1 \odot 1 = d_7 \odot d_6 \odot d_5 \odot d_4$

$c_3 = f(011) = d_7 \odot 1 \odot 1 \odot 1 \odot a_3 \odot 1 \odot 1 \odot 1 = d_7 \odot d_3$

$c_2 = f(010) = d_7 \odot d_6 \odot 1 \odot 1 \odot d_3 \odot d_2 \odot 1 \odot 1 = d_7 \odot d_6 \odot d_3 \odot d_2$

$c_1 = f(001) = d_7 \odot 1 \odot d_5 \odot 1 \odot d_3 \odot 1 \odot d_1 \odot 1 = d_7 \odot d_5 \odot d_3 \odot d_1$

$c_0 = f(000) = d_7 \odot d_6 \odot d_5 \odot d_4 \odot d_3 \odot d_2 \odot d_1 \odot d_0$

d_j can then be derived from c_j in above equations. For example, $d_7 \odot d_6 = c_6$ can be rewritten as

$$\begin{aligned} d_7 \odot d_6 \odot d_7 &= c_6 \odot d_7 \\ d_6 \odot 1 &= c_6 \odot d_7 \\ d_6 &= c_6 \odot c_7 \end{aligned}$$

It is straightforward to find the following relationships

$d_5 = c_7 \odot c_5$

$d_4 = c_7 \odot c_6 \odot c_5 \odot c_4$

$d_3 = c_7 \odot c_3$

$d_2 = c_7 \odot c_6 \odot c_3 \odot c_2$

$d_1 = c_7 \odot c_5 \odot c_3 \odot c_1$

$d_0 = c_7 \odot c_6 \odot c_5 \odot c_4 \odot c_3 \odot c_2 \odot c_1 \odot c_0$

Corollary 3.2. *The conversion between d_j coefficients of fixed polarity COC expansion and c_j coefficients of CPOS expansion is reversible so that*

$$c = [T_n]\Theta d \tag{3.20}$$

Proof. If $d = [d_{2^n-1}d_{2^n-2}\cdots d_0]^t$ is replaced with $c = [c_{2^n-1}c_{2^n-2}\cdots c_0]^t$ in (3.19), equation (3.19) can be rewritten as

$$f(x_{n-1}, x_{n-2}\cdots, x_0) = \{[0 \quad x_{n-1}] \ddagger [0 \quad x_{n-2}] \ddagger \cdots \ddagger [0 \quad x_0]\}\Theta c$$

Similarly, if $c = [c_{2^n-1}c_{2^n-2}\cdots c_0]^t$ is replaced with $d = [d_{2^n-1}d_{2^n-2}\cdots d_0]^t$ in (3.16), equation (3.16) can be rewritten as

$$f(x_{n-1}, x_{n-2}, \cdots, x_0) = \{[\bar{x}_{n-1} \quad x_{n-1}] \ddagger [\bar{x}_{n-2} \quad x_{n-2}] \ddagger \cdots \ddagger [\bar{x}_0 \quad x_0]\}\Theta d$$

In the same way as proved in Corollary 3.1, the conversion from d_j coefficients of fixed polarity COC expansion to c_j coefficients of CPOS expansion is

$$c = [T_n]\Theta d$$

where

$$[T_n] = \begin{bmatrix} T_{n-1} & 1 \\ T_{n-1} & T_{n-1} \end{bmatrix}$$

$$[T_1] = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

From (3.13) and (3.20), the conversion is reversible. □

Example 3.5. When $n = 3$, (3.13) can be written as

$$\begin{bmatrix} d_7 \\ d_6 \\ d_5 \\ d_4 \\ d_3 \\ d_2 \\ d_1 \\ d_0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \Theta \begin{bmatrix} c_7 \\ c_6 \\ c_5 \\ c_4 \\ c_3 \\ c_2 \\ c_1 \\ c_0 \end{bmatrix}$$

While (3.20) can be written as

$$\begin{bmatrix} c_7 \\ c_6 \\ c_5 \\ c_4 \\ c_3 \\ c_2 \\ c_1 \\ c_0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \Theta \begin{bmatrix} d_7 \\ d_6 \\ d_5 \\ d_4 \\ d_3 \\ d_2 \\ d_1 \\ d_0 \end{bmatrix}$$

Corollary 3.3. *If CSOP minterm coefficients* $c = [\bar{c}_{2^n-1} \cdots \bar{c}_1 \bar{c}_0]^t$ *is used for the transformation to COC maxterm coefficients* $d = [d_{2^n-1} \cdots d_1 d_0]^t$ *instead of using CPOS maxterm coefficients* $c = [c_{2^n-1} \cdots c_1 c_0]^t$*, the COC maxterm coefficients* $d = [d_{2^n-1} \cdots d_1 d_0]^t$ *can be obtained in the same way as using CPOS maxterm coefficients but the resulting coefficients need to be modified by complementing* d_{2^n-1}.

Proof. Originally, the transformation from CPOS maxterm coefficients to COC maxterm coefficients is $d = [T_n] \Theta c$. If the CSOP minterms are used instead of CPOS maxterms, the coefficients in $c = [c_{2^n-1} \cdots c_1 c_0]^t$ are complimented. As a result, $c = [\bar{c}_{2^n-1} \cdots \bar{c}_1 \bar{c}_0]^t$.

Because $\bar{x} \odot \bar{y} = x \odot y$, the transformation from $c = [\bar{c}_{2^n-1} \cdots \bar{c}_1 \bar{c}_0]^t$ to $d = [d_{2^n-1} \cdots d_1 d_0]^t$ does not change at all for those rows with even number of zeros. There is only one row with odd number of zeros, which is the first row, thus $d_{2^n-1} = c_{2^n-1}$. Because c_{2^n-1} is complemented, d_{2^n-1} should be complemented to keep the same logic functionality. □

3.4 Map Techniques

3.4.1 Map folding technique for positive polarity using minterms

The positive polarity COC expansion can be obtained from CPOS expansion by using map folding technique [34]. However, based on Corollary 3.3 the positive polarity COC expansion can be obtained from CSOP expansion by using map folding technique. This method will be called minterm map folding technique and is shown in Procedure 3.1.

Procedure 3.1. *Minterm map folding technique for positive polarity expansion conversion from c_j map to d_j map.*

1. Draw c_j map from CSOP expansion by marking "0" for on-set CSOP minterm coefficients and "1" otherwise.

2. Fold the map along the x_i border (i.e. $x_i = 1$ is folded on $x_i = 0$), XNOR the contents of section $x_i = 0$ of the map and then unfold the map, where $0 \leq i \leq n-1$.

3. Repeat Step 2 for the remaining variables.

4. Modify d_{2^n-1} in the map obtained in Step 3 by complementing d_{2^n-1}.

5. Output positive polarity COC expansion according to on-set COC maxterms.

Example 3.6. Given a 3-variable function $f(x_2, x_1, x_0) = \bar{x}_2\bar{x}_1x_0 + \bar{x}_2x_1x_0 + x_2\bar{x}_1\bar{x}_0 + x_2x_1x_0$, obtain the positive polarity COC expansion by using minterm map folding technique.

Step 1:

The coefficients map is shown in Figure 3.4(a) by marking "0" for the corresponding on-set CSOP minterms, i.e., $f(x_2, x_1, x_0) = \sum(1, 3, 4, 7)$.

Step 2:

Fold the map along the x_2 border (i.e. $x_2 = 1$ is folded on $x_2 = 0$) and XNOR the contents of section $x_2 = 0$ of the map as shown in Figure 3.4(a).

Step 3:

Repeat Step 2 for the remaining variables, as shown in Figure 3.4(b) and Figure 3.4(c).

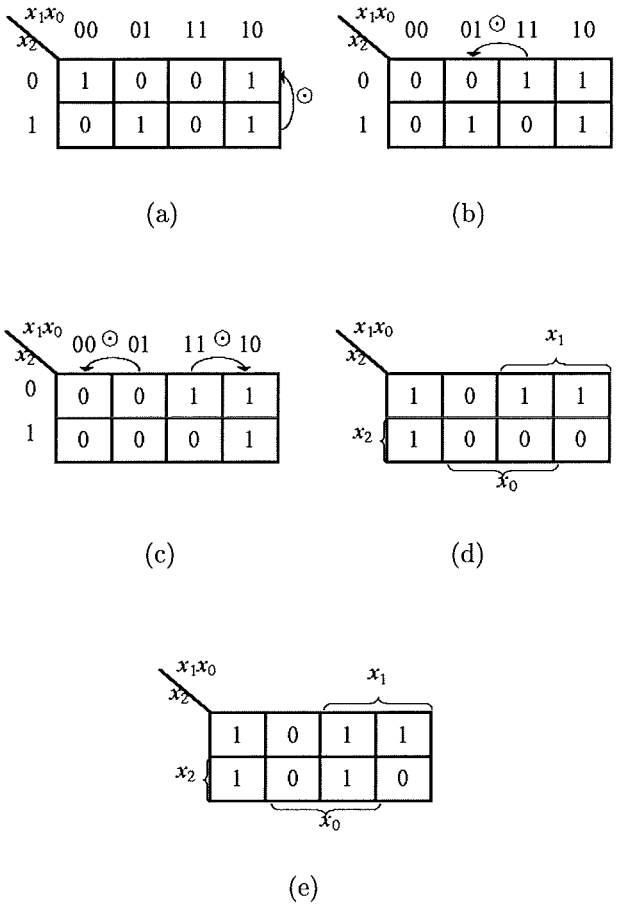


Figure 3.4: Map folding technique for positive polarity using minterms. (a) Folding the map along the x_2 border, (b) Folding the map along the x_1 border, (c) Folding the map along the x_0 border, (d) Coefficients map before modification, (e) Final coefficients map.

Step 4:

Read the "0"s from the map as shown in Figure 3.4(d). $f(x_2, x_1, x_0) = \bigodot \prod(1, 5, 6, 7)$ is obtained. Complement d_7 in the map shown in Figure 3.4(d) for d_7. The resulting coefficients map is shown in Figure 3.4(e).

Step 5:

On-set COC maxterms are obtained by reading the "0"s from the map obtained in Step 4, hence $f(x_2, x_1, x_0) = \bigodot \prod(1, 5, 6)$. The positive polarity

COC expansion is therefore obtained as

$$f(x_2, x_1, x_0) = (x_2 + x_1) \odot x_1 \odot x_0$$

3.4.2 Map folding technique for any polarity

Once the coefficient map for a positive polarity expansion is found, the map folding technique can be carried out to generated fixed polarity COC expansions of any polarity for up to six variables. It is shown in the following procedure.

Procedure 3.2. *Map folding technique for any polarity p conversion from c_j map to d_j map.*

1. Determine and mark the variables which need to be altered according to polarity p when $p_i = 1$, where $p = (p_{n-1} \cdots p_1 p_0)$ is the polarity in the binary form and $0 \leq i \leq n-1$.

2. Fold the map along the x_i border (i.e. $x_i = 0$ is folded on $x_i = 1$), XNOR the contents of section $x_i = 1$ of the map and then unfold the map, where i is the index of the corresponding variables which need to be altered.

3. Repeat Step 2 for the remaining variables which need to be altered.

4. Output the fixed polarity COC expansion of polarity p by replacing x_i with $\bar{x}_i$ in the positive polarity expansion.

For example, the maxterm coefficients of the fixed polarity COC expansion for polarity 1 can be generated by folding d_j coefficient map along the x_0 border (i.e. $x_0 = 0$ is folded on $x_0 = 1$), as shown in Figure 3.5. As a result, the coefficients map for polarity 1 shown in Figure 3.6 is obtained.

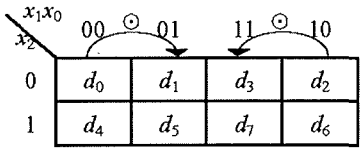


Figure 3.5: Folding d_j coefficient map along x_0 border.

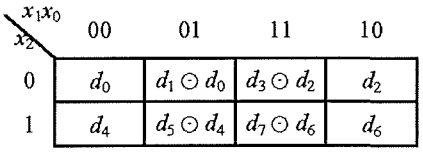


Figure 3.6: A 3-variable d_j coefficient map for polarity 1 after folding.

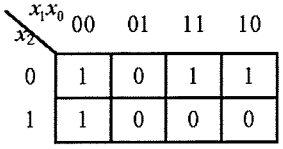


Figure 3.7: A 3-variable d_j coefficient map for polarity 1.

If the on-set COC maxterm coefficients of a 3-variable positive polarity expansion are given as $f(x_2, x_1, x_0) = \bigodot \prod(1, 5, 6)$. The on-set COC maxterm coefficients as shown in Figure 3.7 can be obtained as

$$f(x_2, x_1, \bar{x}_0) = \bigodot \prod(1, 5, 6, 7)$$

The COC expansion of polarity 1 can be obtained by replacing x_0 with $\bar{x}_0$ in the positive polarity expansion shown in Figure 3.3. Hence,

$$f(x_2, x_1, \bar{x}_0) = (x_2 + x_1) \odot x_1 \odot \bar{x}_0 \odot 0$$

The expansion of other polarities can be generated using the same procedure as the generation of the expansion for polarity 1.

3.4.3 Map transformation technique for any polarity

Equation (3.10) can be rewritten by replacing d_j with c_j for positive polarity COC expansion as

$$f(x_{n-1}, \cdots, x_1, x_0) = \bigodot \prod_{j=0}^{2^n-1} (c_j + \bigodot \prod_{e} S_h) \tag{3.21}$$

$$h_i = \begin{cases} \times & j_i = 1 \\ 0 & j_i = 0 \end{cases} \tag{3.22}$$

where both h and j are in binary form, e is the number of "1"s in h, S_h is the COC maxterm and j_i is the ith bit of j.

Example 3.7. Given a 3-variable positive polarity COC expansion, the expansion can be rewritten as

$$\begin{aligned} f(x_2, x_1, x_0) \;=\; & (c_0 + (x_2 + x_1 + x_0)) \\ & \odot (c_1 + (x_2 + x_1 + x_0) \odot (x_2 + x_1)) \\ & \odot (c_2 + (x_2 + x_1 + x_0) \odot (x_2 + x_0)) \\ & \odot (c_3 + (x_2 + x_1 + x_0) \odot (x_2 + x_1) \odot (x_2 + x_0) \odot x_2) \\ & \odot (c_4 + (x_2 + x_1 + x_0) \odot (x_1 + x_0) \\ & \odot (c_5 + (x_2 + x_1 + x_0) \odot (x_2 + x_1) \odot (x_1 + x_0) \odot x_1) \\ & \odot (c_6 + (x_2 + x_1 + x_0) \odot (x_2 + x_0) \odot (x_1 + x_0) \odot x_0) \\ & \odot (c_7 + (x_2 + x_1 + x_0) \odot (x_2 + x_1) \odot (x_2 + x_0) \odot x_2 \\ & \quad \odot (x_1 + x_0) \odot x_1 \odot x_0 \odot 0) \end{aligned}$$

The expansion can be simplified by replacing COC maxterms as shown in (3.11) as

$$\begin{aligned} f(x_2, x_1, x_0) &= (c_0 + S_0) \odot (c_1 + S_0 \odot S_1) \odot (c_2 + S_0 \odot S_2) \\ &\odot (c_3 + S_0 \odot S_1 \odot S_2 \odot S_3) \odot (c_4 + S_0 \odot S_4) \\ &\odot (c_5 + S_0 \odot S_1 \odot S_4 \odot S_5) \odot (c_6 + S_0 \odot S_2 \odot S_4 \odot S_6) \\ &\odot (c_7 + S_0 \odot S_1 \odot S_2 \odot S_3 \odot S_4 \odot S_5 \odot S_6 \odot S_7) \end{aligned}$$

From (3.21), the COC expansion of positive polarity can be obtained directly from the manipulation of c_j coefficient map. It can be seen in Example 3.7 for 3-variable function that if c_0 is on-set CPOS maxterm coefficient, c_0 will only affect S_0, if c_1 is on-set CPOS maxterm coefficient, c_1 will affect S_0 and S_1, and so on. As off-set CPOS maxterm coefficients will not affect the result, the COC expansion can be easily determined from the appearances of the on-set CPOS maxterm coefficient c_j in the c_j coefficient map. Only if the number of the appearance of the CPOS maxterm in the c_j map is odd, the COC maxterms in the same square of d_j map is the on-set COC maxterm.

Definition 3.6. For a given n-variable Boolean function $f(x_{n-1}, \cdots, x_1, x_0)$, p-point is defined by the position of COC maxterm appearing in the square of the d_j coefficients map, where p is same number as the number of polarity.

Example 3.8. Figure 3.3 shows d_j coefficients map for 3 variables. COC Maxterm $x_2 + x_1 + x_0$ appears at position 0 (binary position "000") of the d_j coefficients map, p-point is 0-point. COC Maxterm x_0 appears at position 6 (binary position "110") of the d_j coefficients map, p-point is thus 6-point.

If c_j replaces d_j^p in (3.10), an equation similar to (3.21) can be obtained for COC expansions of any polarity p.

For example, a 3-variable COC expansion of polarity 6 is written as

$$\begin{aligned} f(x_2, x_1, x_0) &= (c_0 + S_0 \odot S_2 \odot S_4 \odot S_6) \\ &\odot (c_1 + S_0 \odot S_1 \odot S_2 \odot S_3 \odot S_4 \odot S_5 \odot S_6 \odot S_7) \\ &\odot (c_2 + S_2 \odot S_6) \odot (c_3 + S_2 \odot S_3 \odot S_6 \odot S_7) \\ &\odot (c_4 + S_4 \odot S_6) \odot (c_5 + S_4 \odot S_5 \odot S_6 \odot S_7) \\ &\odot (c_6 + S_6) \odot (c_7 + S_6 \odot S_7) \end{aligned}$$

The generation of fixed polarity COC expansions with any polarity is shown in following procedure.

Procedure 3.3. *Map Transformation for fixed polarity COC expansion of any polarity p from c_j map to d_j map.*

1. Draw c_j map according to the Boolean function, in which only on-set CPOS maxterms appear on the map as "0".

2. Find the smallest circle on the map that encloses on-set CPOS maxterms and the p-point.

3. Mark "0" for any square in the map enclosed by the circle obtained in Step 2.

4. Repeat Steps 2 and 3 for the rest of on-set maxterms one by one until all the on-set maxterms are enclosed.

5. When the number of "0"s marked for each square in the map is odd, output the corresponding coefficient as the on-set maxterm coefficients of polarity p by XORing the coefficients with polarity p.

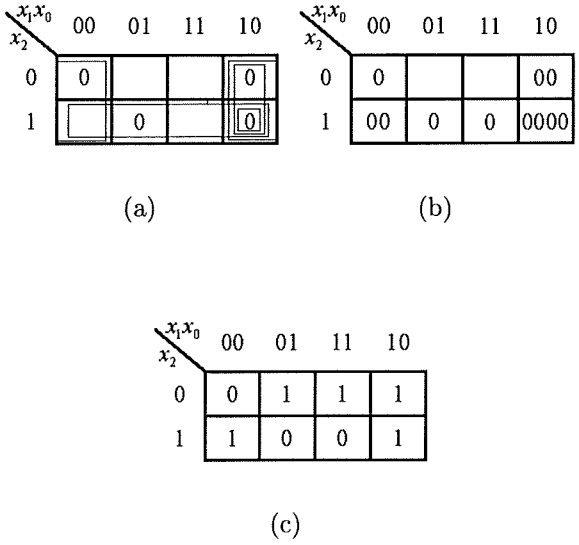


Figure 3.8: Map transformation technique for obtaining COC expansion of polarity 6 for a 3-variable function in CPOS form $f(x_2, x_1, x_0) = \prod(0, 2, 5, 6)$. (a) Circle on-set maxterms via 6-point, (b) Resulting map after marking "0"s, (c) Simplified coefficients maps.

6. Output the fixed polarity COC expansion of polarity p according to the on-set COC maxterm coefficients.

Example 3.9. Obtain fixed polarity COC expansion of polarity 6 using map transformation for a function in CPOS form $f(x_2, x_1, x_0) = \prod(0, 2, 5, 6)$.

The c_j map for the 3-variable function can be drawn and shown in Figure 3.8(a). There are 4 on-set CPOS maxterms.

For fixed polarity COC expansion of polarity 6 conversion, a circle should enclose the 6-point and on-set CPOS maxterms. For on-set CPOS maxterm c_0, the circle encloses S_0, S_2 , S_4 and S_6. Mark "0" for S_0, S_2 , S_4 and S_6 in the corresponding square. For on-set CPOS maxterm c_2, the circle encloses S_2 and S_6. Mark "0" for S_2 and S_6 in the corresponding square. For on-set CPOS maxterm c_5, the circle encloses S_4, S_5, S_6 and S_7. Mark "0" for S_4, S_5, S_6

and S_7 in the corresponding square. For on-set CPOS maxterm c_6, the circle encloses S_6 only. Mark "0" for S_6.

Figure 3.8(a) and Figure 3.8(b) show c_j map after drawing the circles and marking "0" respectively. Those with odd number of "0"s appearing "0" in the map, as shown in Figure 3.8(c), are the on-set maxterm coefficients of COC expansion before XOR operation. Since the polarity is 6, the coefficients need to be XORed. Hence, the corresponding on-set maxterm coefficients and fixed polarity COC expansion of polarity 6 are given in (3.23) and (3.24) respectively.

$$f(\bar{x}_2, \bar{x}_1, x_0) = \bigodot \prod (1, 3, 6) \tag{3.23}$$

$$f(\bar{x}_2, \bar{x}_1, x_0) = (\bar{x}_2 + \bar{x}_1) \odot \bar{x}_2 \odot x_0 \tag{3.24}$$

For other polarities, Figure 3.9 shows the map after marking "0"s and simplified COC coefficient maps are shown in Figure 3.10. The results for other polarities are given in following equations.

$$f(x_2, x_1, x_0) = (x_2 + x_1) \odot x_1 \odot x_0 \tag{3.25}$$

$$f(x_2, x_1, \bar{x}_0) = (x_2 + x_1) \odot x_1 \odot \bar{x}_0 \odot 0 \tag{3.26}$$

$$f(x_2, \bar{x}_1, x_0) = (x_2 + \bar{x}_1) \odot x_2 \odot \bar{x}_1 \odot x_0 \odot 0 \tag{3.27}$$

$$f(x_2, \bar{x}_1, \bar{x}_0) = (x_2 + \bar{x}_1) \odot x_2 \odot \bar{x}_1 \odot \bar{x}_0 \tag{3.28}$$

$$f(\bar{x}_2, x_1, x_0) = (\bar{x}_2 + x_1) \odot x_0 \tag{3.29}$$

$$f(\bar{x}_2, x_1, \bar{x}_0) = (\bar{x}_2 + x_1) \odot \bar{x}_0 \odot 0 \tag{3.30}$$

$$f(\bar{x}_2, \bar{x}_1, \bar{x}_0) = (\bar{x}_2 + \bar{x}_1) \odot \bar{x}_2 \odot \bar{x}_0 \odot 0 \tag{3.31}$$

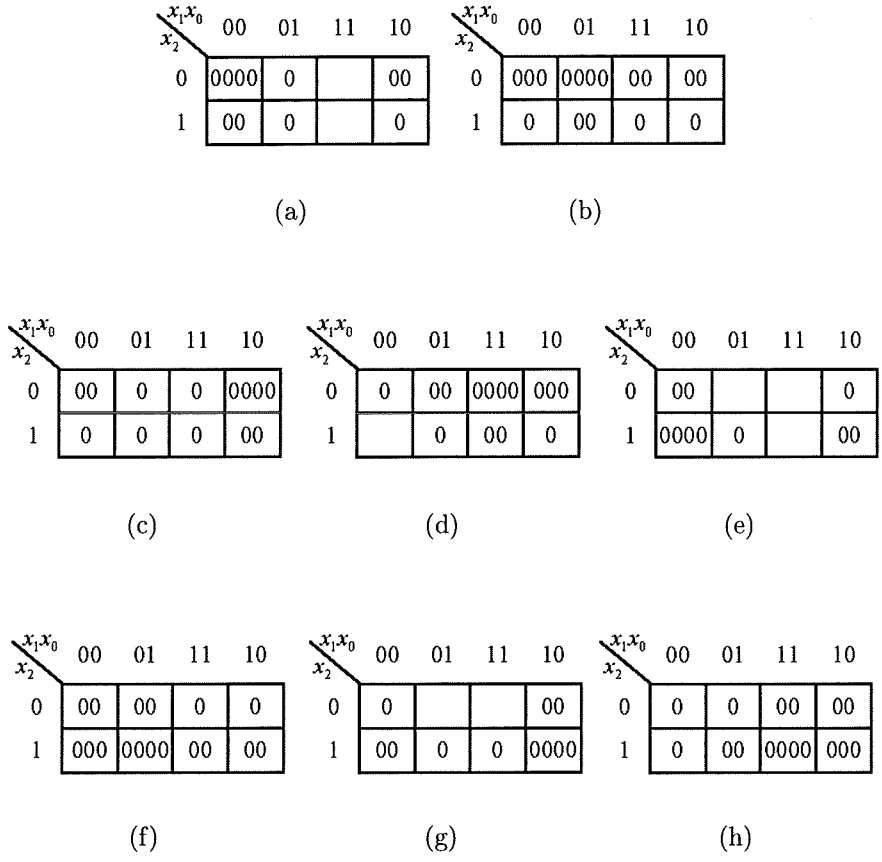


Figure 3.9: Mapping transformation technique for obtaining COC expansion of any polarity. (a) Polarity 0 via 0-point, (b) Polarity 1 via 1-point, (c) Polarity 2 via 2-point, (d) Polarity 3 via 3-point, (e) Polarity 4 via 4-point, (f) Polarity 5 via 5-point, (g) Polarity 6 via 6-point, (h) Polarity 7 via 7-point.

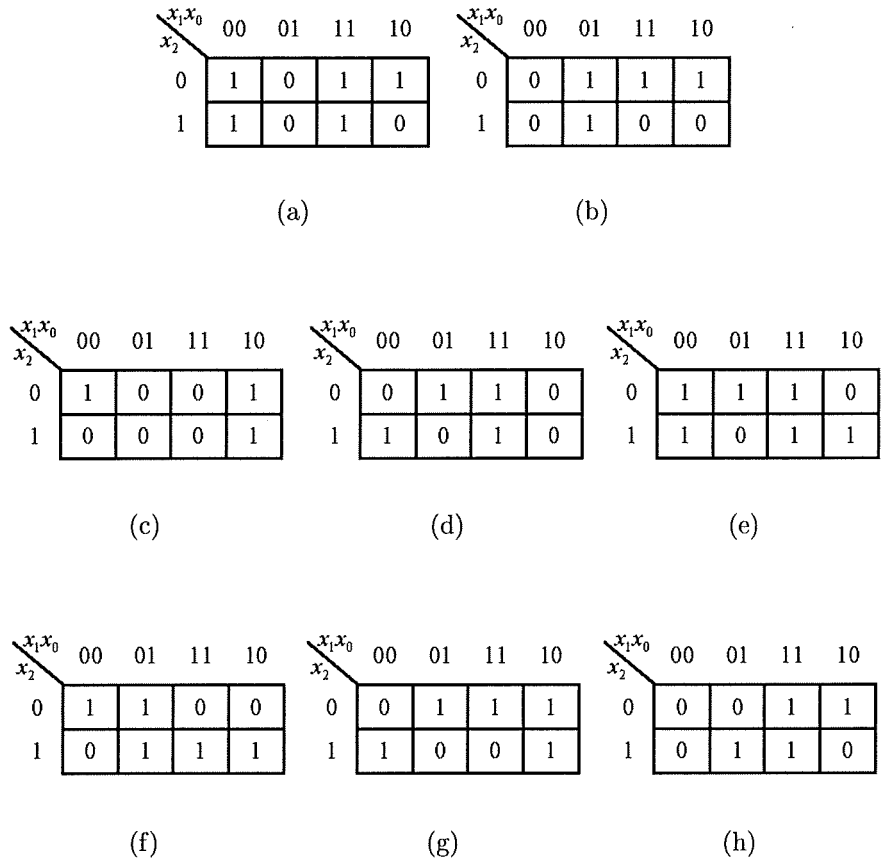


Figure 3.10: Coefficients of COC expansion of all the 8 polarities transformation. (a) Polarity 0, (b) Polarity 1, (c) Polarity 2, (d) Polarity 3, (e) Polarity 4, (f) Polarity 5, (g) Polarity 6, (h) Polarity 7.

3.5 Summary

In this chapter, the basic definitions for the polarity of fixed polarity COC expansions and CSOP and CPOS expansions are given. Based on the features of coincidence operation, the COC expansions of logical functions as DFRM form are discussed. The transformation matrix for conversion between COC and CPOS expansions is also given.

Map folding technique is used for conversion between standard Boolean and fixed polarity COC expansions for up to 6 variables. It starts with generating the positive polarity COC expansions first and then derives other fixed polarity COC expansions of any polarity. Another simple but useful map transformation technique is introduced to generate fixed polarity COC expansions of any polarity directly from the c_j map. Drawing circles, however, becomes inconvenient if the number of variables is greater than 4.

Chapter 4

Multi-segment Method for Dual Forms of RM Conversion

4.1 Introduction

Due to the limitation of map methods, map folding and map transformation technique can only be used for functions in which the number of variables is less than or equal to 6. It is not practical to use map techniques to optimise large variable functions. Other methods are required for large functions. Conversion method between Canonical Product-of-Sums (CPOS) and fixed polarity COC expansions based on the matrix method were proposed in [34]. The method based on the matrix multiplication for large functions requires significant CPU time. Reference [51] used XNOR operation for on-set maxterms which again required significant CPU time for large variables and the results were only given for up to 17 variables. In this chapter, algorithms are presented for conversion between standard Boolean and fixed polarity COC expansions to any polarity. The rest of the chapter is organised as follows. In Section 4.2, generalised

method is proposed for conversion between CPOS and fixed polarity COC expansions for large functions. Section 4.3 shows the proposed multi-segment methods to achieve efficient conversion in details. In Section 4.4 algorithms are further extended to the conversion of any polarity. Moreover, a new minterm method utilises on-set CSOP minterms instead of on-set CPOS maxterms to reduce CPU time is given in Section 4.5. Experimental results are then given in Section 4.6.

4.2 Generalised Method Based on On-set Coefficient Coverage

By using map based techniques, COC maxterm coefficients can be obtained up to 6 variables. The c_j coefficients and d_j coefficients have the following relationship if represented in the binary form when $n = 3$.

$d_{111} = c_{111}$

$d_{110} = c_{111} \odot c_{110}$

$d_{101} = c_{111} \odot c_{101}$

$d_{100} = c_{111} \odot c_{110} \odot c_{101} \odot c_{100}$

$d_{011} = c_{111} \odot c_{011}$

$d_{010} = c_{111} \odot c_{110} \odot c_{011} \odot c_{010}$

$d_{001} = c_{111} \odot c_{101} \odot c_{011} \odot c_{001}$

$d_{000} = c_{111} \odot c_{110} \odot c_{101} \odot c_{100} \odot c_{011} \odot c_{010} \odot c_{011} \odot c_{000}$

Based on (3.13) an n-variable function can be rewritten as

$$\begin{aligned} d_j &= d_{u_{n-1}u_{n-2}\cdots u_1u_0} \\ &= \odot\prod c_{v_{n-1}v_{n-2}\cdots v_1v_0} \end{aligned} \tag{4.1}$$

where $u = (u_{n-1}u_{n-2} \cdots u_1u_0)$ and $v = (v_{n-1}v_{n-2} \cdots v_1v_0)$ are in binary form and $0 \leq j, u, v \leq 2^n - 1$.

The ith bit v_i in c_j coefficient and u_i in d_j coefficient have the following relationship:

$$v_i = \begin{cases} \times, & u_i = 0 \\ 1, & u_i = 1 \end{cases} \tag{4.2}$$

where $\times$ stands for "don't care" and $0 \leq i \leq n - 1$.

Definition 4.1. Given two decimal integers, $u = (u_{n-1}u_{n-2} \cdots u_1u_0)$ and $v = (v_{n-1}v_{n-2} \cdots v_1v_0)$, which are in binary form, u covers v if $u_i \leq v_i$ numerically for all i, $u_i, v_i \in \{0, 1\}$ and $0 \leq i \leq n - 1$.

For example, $(14)_{10} = (1110)_2$ covers $(15)_{10} = (1111)_2$ and $(14)_{10} = (1110)_2$.

With reference to (4.1) and (4.2), any d_j coefficient depends on g_j^v as

$$g_j^v = \bigwedge_{i=0}^{n-1} (\bar{v}_i, u_i) \tag{4.3}$$

$$\phi(d_j) = \sum_v \overline{g_j^v} \tag{4.4}$$

where $\bigwedge$ is the bitwise "AND" operator. If the result is "0" for every i, $g_j^v = 0$. Otherwise, $g_j^v = 1$. $\sum$ is the sum operator and $0 \leq j \leq 2^n - 1$.

According to (4.3) and (4.4), the on-set COC maxterm coefficients can be generalised and calculated by using bitwise "INVERSE" and "AND" operations. d_j is on-set COC maxterm coefficient when $\phi(d_j)$ is odd number.

Example 4.1. Given a 4-variable function in CPOS form $f(x_3, x_2, x_1, x_0) = \prod(0, 1, 3, 5, 7, 9, 10, 12, 13, 14)$, obtain on-set COC maxterm coefficient.

For example, for the first CPOS on-set coefficient $v = 0$, g_{14}^0 is calculated as

$$\begin{aligned} g_{14}^0 &= \bar{0}\bar{0}\bar{0}\bar{0} \bigwedge 1110 \\ &= 1111 \bigwedge 1110 \\ &= 1110 = 1 \end{aligned}$$

For $v = 1$, g_{14}^1 is calculated as

$$\begin{aligned} g_{14}^1 &= \bar{0}\bar{0}\bar{0}\bar{1} \bigwedge 1110 \\ &= 1110 \bigwedge 1110 \\ &= 1110 = 1 \end{aligned}$$

In the same way, the rest of g_{14}^v for on-set CPOS coefficients are obtained as follows.

$$\begin{array}{llll} g_{14}^3 = 1 & g_{14}^5 = 1 & g_{14}^7 = 1 & g_{14}^9 = 1 \\ g_{14}^{10} = 1 & g_{14}^{12} = 1 & g_{14}^{13} = 1 & g_{14}^{14} = 0 \end{array}$$

Coefficient d_{14} is then calculated as

$$\phi(d_{14}) = \sum_v \overline{g_{14}^v} = 1$$

Because $\phi(d_{14})$ is odd number, d_{14} is on-set COC maxterm coefficient.

The rest of coefficients are obtained in the same way, which are

$$\begin{array}{llll} \phi(d_{15}) = 0 & \phi(d_{13}) = 1 & \phi(d_{12}) = 3 & \phi(d_{11}) = 0 \\ \phi(d_{10}) = 2 & \phi(d_9) = 2 & \phi(d_8) = 5 & \phi(d_7) = 1 \\ \phi(d_6) = 2 & \phi(d_5) = 3 & \phi(d_4) = 5 & \phi(d_3) = 2 \\ \phi(d_2) = 4 & \phi(d_1) = 6 & \phi(d_0) = 10 & \end{array}$$

Hence, $f(x_3, x_2, x_1, x_0) = \bigodot \prod(4, 5, 7, 8, 12, 13, 14)$.

4.3 Multi-segment Method Based on Maxterms

When the number of on-set CPOS maxterms increases, the computation time for conversion increases significantly. However, if the number of on-set CPOS maxterm coefficients can be divided into several groups and the groups of the coefficients can be reused, the CPU time can be reduced considerably. Multi-segment method was initially introduced in [118]. The multi-segment method based on maxterms is called maxterm multi-segment method.

Definition 4.2. For an n-variable Boolean function $f(x_{n-1}, \cdots, x_1, x_0)$, the on-set CPOS maxterm coefficients can be grouped into w segments, assuming the on-set CPOS maxterm coefficients are pre-ordered in decreasing order. The index of the on-set CPOS maxterm coefficients are assigned to the segments following the rule of (4.5).

$$j \in W_k, \quad k * 2^l \leq j < (k+1) * 2^l \tag{4.5}$$

where $*$ is the multiplication operator, j is the jth on-set c_j coefficient, W_k is the kth segment and $0 \leq k \leq w-1$. 2^l is the maximum number of on-set coefficients in one segment and $0 \leq l \leq n-1$. w is the number of segments and is selected for simplicity of the algorithm as in (4.6), i.e., each on-set coefficient is divided into two parts.

$$w = 2^{n-l} = 2^{n-[\frac{n}{2}]} \tag{4.6}$$

where $[\,]$ is an integer operator and $1 \leq w \leq 2^n$.

Example 4.2. Given a 4-variable function in CPOS form $f(x_3, x_2, x_1, x_0) = \prod(14, 13, 12, 10, 9, 7, 5, 3, 1, 0)$, assign the on-set coefficients to the respective segments.

The on-set CPOS maxterm coefficients are grouped into $w = 2^{n-\lceil \frac{n}{2} \rceil} = 4$ segments and each segment has maximum 4 on-set coefficients, hence

$$\begin{aligned} W_0 &= \{14, 13, 12\} \\ W_1 &= \{10, 9\} \\ W_2 &= \{7, 5\} \\ W_3 &= \{3, 1, 0\} \end{aligned}$$

Corollary 4.1. *Given an integer* $u = (u_{n-1}u_{n-2} \cdots u_1u_0)$ *and a set of integers* V*, where* v *is an integer and is one of integers in* V*. The number of elements in* V *covered by* u *remains unchanged after adding an integer* $((2^{n-l}-1)-u'')*2^l$ *to* i *and elements of* V *if* $u'' = v''$*, where* $u'' = (u_{n-1}u_{n-2} \cdots u_{l+1}u_l)$*,* $v'' = (v_{n-1}v_{n-2} \cdots v_{l+1}v_l)$*,* $0 \leq l \leq n-1$*.*

Proof. The integer numbers, u and v, where v is any integer of V, can be rewritten as

$$\begin{aligned} u &= u_{n-1}u_{n-2} \cdots u_1u_0 \\ &= \underbrace{u_{n-1}u_{n-2} \cdots u_{l+1}u_l}_{u''}\underbrace{u_{l-1}u_{l-2} \cdots u_1u_0}_{u'''} \end{aligned}$$

$$\begin{aligned} v &= v_{n-1}v_{n-2} \cdots v_1v_0 \\ &= \underbrace{v_{n-1}v_{n-2} \cdots v_{l+1}v_l}_{v''}\underbrace{v_{l-1}v_{l-2} \cdots v_1v_0}_{v'''} \end{aligned}$$

Because the significant bits between l and $n-1$ are the same for u and V, i.e., $u'' = v''$, u'' covers $v''$by Definition 4.1.

Let u' and v' be the integers and V' be the set of integers after addition. After adding an integer $((2^{n-l}-1)-u'')*2^l$ to u and V, all those $n-l$ bits between l and $n-1$ are set to be "1"s for u' and V'. However, g_j^v remains unchanged for those l bits between 0 and $l-1$. Thus the number of integers in V covered by u remains unchanged after addition. □

Example 4.3. Given $u = (1000)_2$, $V = \{(1001)_2, (1010)_2, (1011)_2\}$, $n = 4$ and $l = 2$, all of numbers in V and u begin with "$(10)_2$", that is $u'' = (2)_{10}$ and $v'' = (2)_{10}$.

Add $((2^{n-l}-1)-u'')*2^l = ((2^{4-2}-1)-2)*2^2 = (4-1-2)*4 = 4$ to u and V, $u' = (1100)_2$ and $V' = \{(1101)_2, (1110)_2, (1111)_2\}$. The number of integers in V' covered by u' is the same as the number of integers in V covered by u before addition, which is 3.

Based on Corollary 4.1, (4.3) and (4.4), the maxterm multi-segment method for conversion from CPOS expansion to positive polarity COC expansion is achieved as follows.

Procedure 4.1. *The maxterm multi-segment method for conversion from CPOS expansion to positive polarity COC expansion.*

1. Obtain on-set CPOS maxterm coefficients from CPOS expansion and sort the on-set CPOS maxterm coefficients into decreasing order.

2. Divide the on-set CPOS coefficients into w segments and modify the coefficients in each segment based on Definition 4.2 and Corollary 4.1.

3. Find the covered coefficients and calculate COC maxterm coefficients for each segment by using bitwise "INVERSE" and "AND" operations,

according to (4.3) and (4.4). Those numbers of covered coefficients that are odd number are the on-set COC maxterm coefficients.

4. Output the positive polarity COC expansion according to the on-set COC maxterm coefficients.

As it can be seen in (3.20), the conversion from positive polarity COC expansion to CPOS expansion is the reserve of the conversion from CPOS expansion to positive polarity COC expansion. Thus, the same procedure can be applied to obtain CPOS expansion by simply replacing the on-set CPOS maxterm coefficients in Procedure 4.1 with on-set COC maxterm coefficients.

Example 4.4. Given a 4-variable function $f(x_3, x_2, x_1, x_0) = (x_3 + x_2 + x_1 + x_0)(x_3+x_2+x_1+\bar{x}_0)(x_3+x_2+\bar{x}_1+\bar{x}_0)(x_3+\bar{x}_2+x_1+\bar{x}_0)(x_3+\bar{x}_2+\bar{x}_1+\bar{x}_0)(\bar{x}_3+x_2+x_1+\bar{x}_0)(\bar{x}_3+x_2+\bar{x}_1+x_0)(\bar{x}_3+\bar{x}_2+x_1+x_0)(\bar{x}_3+\bar{x}_2+x_1+\bar{x}_0)(\bar{x}_3+\bar{x}_2+\bar{x}_1+x_0)$ and $w = 4$, calculate the positive polarity COC expansion.

Let $D[\,]$ store the number of covered coefficients in each segment.

Step 1:

Sort the on-set CPOS coefficients into decreasing order as

$$f(x_3, x_2, x_1, x_0) = \prod(14, 13, 12, 10, 9, 7, 5, 3, 1, 0)$$

Step 2:

The on-set coefficients are divided into 4 segments, $W_0 = \{14, 13, 12\}$, $W_1 = \{10, 9\}$, $W_2 = \{7, 5\}$ and $W_3 = \{3, 1, 0\}$ according to (4.5). The segments are then modified to $W_0 = \{14, 13, 12\}$, $W_1 = \{14, 13\}$, $W_2 = \{15, 13\}$ and $W_3 = \{15, 13, 12\}$.

Step 3:

Firstly, coefficient 15 is found in the W_2 and W_3 segments because coefficient $(15)_{10} = (1111)_2$ covers coefficient $(15)_{10} = (1111)_2$ only. The number of covered coefficient is stored in 2nd and 3rd segments. As a result, $D[0] = 0$, $D[1] = 0$, $D[2] = 1$ and $D[3] = 1$.

$\phi(d_{15}) = D[0] = 0$

$\phi(d_{11}) = D[0] + D[1] = 0 + 0 = 0$

$\phi(d_7) = D[0] + D[2] = 0 + 1 = 1$

$\phi(d_3) = D[0] + D[1] + D[2] + D[3] = 0 + 0 + 1 + 1 = 2$

Only d_7 should be included because the number of covered coefficients is odd.

Secondly, because coefficient $(14)_{10} = (1110)_2$ covers coefficients $(14)_{10} = (1110)_2$ and $(15)_{10} = (1111)_2$, both coefficients 14 and 15 need to be found. Coefficient 14 is found in the W_0 and W_1. Coefficient 15 is found in the W_2 and W_3 segments. As a result, $D[0] = 1$, $D[1] = 1$, $D[2] = 1$ and $D[3] = 1$.

$\phi(d_{14}) = D[0] = 1$

$\phi(d_{10}) = D[0] + D[1] = 1 + 1 = 2$

$\phi(d_6) = D[0] + D[2] = 1 + 1 = 2$

$\phi(d_2) = D[0] + D[1] + D[2] + D[3] = 1 + 1 + 1 + 1 = 4$

Only d_{14} should be included because the number of covered coefficients is odd.

Thirdly, because coefficient $(13)_{10} = (1101)_2$ covers coefficients $(13)_{10} = (1101)_2$ and $(15)_{10} = (1111)_2$, both coefficients 13 and 15 need to be found. Coefficient 13 is found in the W_0, W_1, W_2 and W_3. Coefficient 15 is found in the W_2 and W_3 segments. As a result, $D[0] = 1$, $D[1] = 1$, $D[2] = 2$ and $D[3] = 2$.

$\phi(d_{13}) = D[0] = 1$

$\phi(d_9) = D[0] + D[1] = 1 + 1 = 2$

$\phi(d_5) = D[0] + D[2] = 1 + 2 = 3$

$\phi(d_1) = D[0] + D[1] + D[2] + D[3] = 1 + 1 + 2 + 2 = 6$

d_{13} and d_5 should be included because the numbers of covered coefficients are odd.

Fourthly, because coefficient $(12)_{10} = (1100)_2$ covers four coefficients $(12)_{10} = (1100)_2$, $(13)_{10} = (1101)_2$, $(14)_{10} = (1110)_2$ and $(15)_{10} = (1111)_2$, coefficients 12, 13, 14 and 15 need to be found. Coefficient 12 is found in the W_0 and W_3. Coefficient 13 is found in the W_0, W_1, W_2 and W_3. Coefficient 14 is found in the W_0 and W_1. Coefficient 15 is found in the W_2 and W_3 segments. As a result, $D[0] = 3$, $D[1] = 2$, $D[2] = 2$ and $D[3] = 3$.

$\phi(d_{12}) = D[0] = 3$

$\phi(d_8) = D[0] + D[1] = 3 + 2 = 5$

$\phi(d_4) = D[0] + D[2] = 3 + 2 = 5$

$\phi(d_0) = D[0] + D[1] + D[2] + D[3] = 3 + 2 + 2 + 3 = 10$

d_{12} , d_8 and d_4 should be included because the numbers of covered coefficients are odd.

The final solution is

$$\begin{aligned} f(x_3, x_2, x_1, x_0) &= \bigodot \prod (4, 5, 7, 8, 12, 13, 14) \\ &= (x_3 + x_1 + x_0) \odot (x_3 + x_1) \odot x_3 \\ &\quad \odot (x_2 + x_1 + x_0) \odot (x_1 + x_0) \odot x_1 \odot x_0 \end{aligned}$$

4.4 Generalised Polarity Conversion

For any n-variable Boolean function $f(x_{n-1}, \cdots, x_1, x_0)$, there are 2^n fixed polarity COC expansions. COC expansions with different polarities may have different number of on-set coefficients. To simplify the conversion algorithm, the concept of Boolean polarity is adapted from [89,118] and applied to CPOS expansion as well, which is given in the following definition.

Definition 4.3. Any n-variable Boolean function $f(x_{n-1}, \cdots, x_1, x_0)$ that is represented in maxterms expansion as in (3.7) is defined as the CPOS expansion with zero polarity. Any n-variable Boolean function $f(\ddot{x}_{n-1}, \cdots, \ddot{x}_1, \ddot{x}_0)$ can be in canonical form of expansion with polarity p, where p is the polarity in binary form and $p = (p_{n-1} \cdots p_1 p_0)$. Any variable $\ddot{x}_i$ can be represented as in (4.7).

$$\ddot{x}_i = \begin{cases} \bar{x}_i & p_i = 1 \\ x_i & p_i = 0 \end{cases} \tag{4.7}$$

where $0 \leq i \leq n-1$. $\ddot{x}_i$ can be in true or complemented form but not both, $\bar{x}_i$ is the complemented form of x_i. There are totally 2^n fixed polarities for a n-variable Boolean function in CPOS form.

Accordingly, (3.7) should be extended to (4.8).

$$f(\ddot{x}_{n-1}, \cdots, \ddot{x}_1, \ddot{x}_0) = \prod_{j=0}^{2^n-1} (c_j + M_j) \tag{4.8}$$

where M_j is in the binary form and $M_j = \acute{x}_{n-1} + \cdots + \acute{x}_1 + \acute{x}_0$.

$$\acute{x}_i = \begin{cases} \bar{\ddot{x}}_i & j_i = 1 \\ \ddot{x}_i & j_i = 0 \end{cases} \tag{4.9}$$

where $\bar{\ddot{x}}_i$ is the complemented form of $\ddot{x}_i$.

Equation (4.10) can be derived from (4.9)

$$j_i = \begin{cases} 1 & \acute{x}_i = \bar{\ddot{x}}_i \\ 0 & \acute{x}_i = \ddot{x}_i \end{cases} \tag{4.10}$$

Corollary 4.2. *Let R be the number of on-set CPOS maxterm coefficients for a given n-variable Boolean function $f(x_{n-1}, \cdots, x_1, x_0)$, there is always R numbers of on-set CPOS maxterm coefficients for any of the 2^n fixed polarity expansions.*

Proof. Let σ be the σth on-set coefficient of n-variable $f(x_{n-1}, \cdots, x_1, x_0)$ with polarity p and ϖ be the ϖth on-set coefficient of n-variable $f(\ddot{x}_{n-1}, \cdots, \ddot{x}_1, \ddot{x}_0)$ with polarity $\ddot{p}$. If n-variable $f(x_{n-1}, \cdots, x_1, x_0)$ is converted to n-variable $f(\ddot{x}_{n-1}, \cdots, \ddot{x}_1, \ddot{x}_0)$, there is a one-to-one matching between the on-set CPOS maxterm coefficient with polarity p and on-set CPOS maxterm coefficient with polarity $\ddot{p}$, because $\varpi = \sigma \oplus p \oplus \ddot{p}$, where $0 \leq \sigma, \varpi \leq 2^n - 1$ and $0 \leq p, \ddot{p} \leq 2^n - 1$. Hence the number of on-set CPOS maxterm coefficients, R, remains the same for the expansion of any polarity. □

Corollary 4.3. *If polarity $\tilde{p}$ is applied to both sides of $d = [T_n]\Theta c$, the transformation for positive polarity expansion, the index of maxterm coefficients of CPOS expansion changes for different polarities, but the index of maxterm coefficients of COC expansion remains unchanged. The transformation matrix can be used for conversion to fixed polarity COC expansion of any polarity.*

Proof. If polarity $\tilde{p}$ is applied to both sides of $d = [T_n]\Theta c$ for positive polarity, $d = [T_n]\Theta c$ can be rewritten as

$$\tilde{d} = [T_n]\Theta\tilde{c} \tag{4.11}$$

where $\tilde{c} = [\tilde{c}_{2^n-1} \cdots \tilde{c}_1 \tilde{c}_0]^t$, $\tilde{d} = [\tilde{d}_{2^n-1} \cdots \tilde{d}_1 \tilde{d}_0]^t$ and $0 \leq \tilde{p} \leq 2^n - 1$.

Because $\tilde{c} = c \oplus \tilde{p}$, where $c = [c_{2^n-1} \cdots c_1 c_0]^t$, the index of maxterm coefficients of CPOS expansion in $\tilde{c}$ changes for different polarities. Because the transformation is for positive polarity expansion, the coefficients need to be XORed polarity $\tilde{p}$ again, $\tilde{d} = d \oplus \tilde{p} \oplus \tilde{p} = d$. As a result, the index of maxterm coefficients of fixed polarity COC expansion remains unchanged. It is therefore that the transformation matrix can be used for the conversion to fixed polarity COC expansion of any polarity. □

Example 4.5. Obtain the fixed polarity COC expansion of polarity 6 for a 3-variable function $f(x_2, x_1, x_0) = (x_2 + x_1 + x_0)(x_2 + \bar{x}_1 + x_0)(\bar{x}_2 + x_1 + \bar{x}_0)(\bar{x}_2 + \bar{x}_1 + x_0)$.

The function in CPOS form is $f(x_2, x_1, x_0) = \prod(0, 2, 5, 6)$. Since the polarity $\ddot{p} = 6$, from (4.7) $\ddot{x}_2 = \bar{x}_2$, $\ddot{x}_1 = \bar{x}_1$ and $\ddot{x}_0 = x_0$ are obtained. Since $\bar{\bar{x}} = x$, from (4.8) to (4.10), the function can be represented as $f(\bar{x}_2, \bar{x}_1, x_0)$ with polarity 6 as

$$\begin{aligned} f(\ddot{x}_2, \ddot{x}_1, \ddot{x}_0) &= f(\bar{x}_2, \bar{x}_1, x_0) \\ &= (\bar{\bar{x}}_2 + \bar{\bar{x}}_1 + x_0)(\bar{\bar{x}}_2 + \bar{x}_1 + x_0)(\bar{x}_2 + \bar{\bar{x}}_1 + \bar{x}_0)(\bar{x}_2 + \bar{x}_1 + x_0) \\ &= \prod\{(110), (100), (011), (000)\} \\ &= \prod(6, 4, 3, 0) \end{aligned}$$

Hence, $\tilde{c} = [1, 0, 1, 0, 0, 1, 1, 0]^t$ and

$\tilde{d}$ is obtained as

$$\begin{bmatrix} \tilde{d}_7 \\ \tilde{d}_6 \\ \tilde{d}_5 \\ \tilde{d}_4 \\ \tilde{d}_3 \\ \tilde{d}_2 \\ \tilde{d}_1 \\ \tilde{d}_0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \Theta \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

As a result, the coefficients of COC expansion of polarity 6 and its expansion are

$$\begin{aligned} f(\bar{x}_2, \bar{x}_1, x_0) &= \bigodot \prod (1, 3, 6) \\ f(\bar{x}_2, \bar{x}_1, x_0) &= (\bar{x}_2 + \bar{x}_1) \odot \bar{x}_2 \odot x_0 \end{aligned}$$

Thus, any n-variable Boolean function that is converted from CPOS expansion to fixed polarity COC expansion with any polarity p can be generalised and is achieved as follows, according to Corollaries 4.2 and 4.3.

Procedure 4.2. *The maxterm multi-segment method for conversion from CPOS to fixed polarity COC expansions of any polarity p, where* $0 \leq p \leq 2^n - 1$.

1. Obtain on-set CPOS maxterm coefficients from CPOS expansion.
2. Derive on-set CPOS maxterm coefficients with polarity p by XORing on-set CPOS maxterm coefficients with polarity p.
3. Divide the newly generated on-set CPOS maxterm coefficients into w segments and modify the coefficients in each segment.

4. Find the covered coefficients and calculate COC maxterm coefficients for each segment by using bitwise "INVERSE" and "AND" operations. Those numbers of covered coefficients that are odd number are the on-set COC maxterm coefficients.

5. Output the fixed polarity COC expansion of polarity p according to the on-set COC maxterm coefficients obtained in Step 4.

Corollary 4.4. *The transformation matrix can be used for conversion from fixed polarity COC expansion of any polarity to CPOS expansion. The conversion procedure is the reverse of the procedure used for conversion from CPOS expansion to fixed polarity COC expansion of any polarity.*

Proof. As it is known from Corollary 3.2 and also in (3.13) and (3.20), the conversion between CPOS and positive polarity COC expansions is reversible. If polarity $\tilde{p}$ is applied to both sides of (3.20) for positive polarity, (3.20) can be rewritten as

$$\tilde{c} = [T_n]\Theta\tilde{d} \tag{4.12}$$

where $\tilde{c} = [\tilde{c}_{2^n-1} \cdots \tilde{c}_1\tilde{c}_0]^t$, $\tilde{d} = [\tilde{d}_{2^n-1} \cdots \tilde{d}_1\tilde{d}_0]^t$ and $0 \leq \tilde{p} \leq 2^n - 1$. The index of maxterm coefficients of CPOS expansion in $\tilde{c}$ changes for different polarities. Therefore the coefficients need to be XORed polarity $\tilde{p}$ as in $c = \tilde{c} \oplus \tilde{p}$, where $c = [c_{2^n-1} \cdots c_1c_0]^t$. □

The conversion from fixed polarity COC expansion of polarity p to CPOS expansion is the reserve of the conversion from CPOS expansion to fixed polarity COC expansion of polarity p.

Example 4.6. Obtain the CPOS expansion for a 3-variable fixed polarity COC expansion of polarity 6 $f(\bar{x}_2, \bar{x}_1, x_0) = f(\bar{x}_2, \bar{x}_1, x_0) = (\bar{x}_2 + \bar{x}_1) \odot \bar{x}_2 \odot x_0$.

The function in COC form is $f(\bar{x}_2, \bar{x}_1, x_0) = \bigodot \prod(1, 3, 6)$. Hence, $\tilde{d} = [1, 0, 1, 1, 0, 1, 0, 1]^t$.

$\tilde{c}$ is obtained as

$$\begin{bmatrix} \tilde{c}_7 \\ \tilde{c}_6 \\ \tilde{c}_5 \\ \tilde{c}_4 \\ \tilde{c}_3 \\ \tilde{c}_2 \\ \tilde{c}_1 \\ \tilde{c}_0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \Theta \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

As a result, the coefficients of CPOS expansion of polarity 6 in CPOS form is $f(\bar{x}_2, \bar{x}_1, x_0) = \prod(0, 3, 4, 6)$. Hence,

$$\begin{aligned} f(x_2, x_1, x_0) &= \prod(0, 3, 4, 6) \oplus 6 = \prod(6, 5, 2, 0) \\ &= \prod(0, 2, 5, 6) \end{aligned}$$

The CPOS expansion is then

$$f(x_2, x_1, x_0) = (x_2 + x_1 + x_0)(x_2 + \bar{x}_1 + x_0)(\bar{x}_2 + x_1 + \bar{x}_0)(\bar{x}_2 + \bar{x}_1 + x_0)$$

Thus, fixed polarity COC expansion of polarity p is converted to CPOS expansion for any n-variable Boolean function can be generalised and is achieved as follows.

Procedure 4.3. *The maxterm multi-segment method for conversion from COC expansion of polarity p to CPOS expansion, where* $0 \leq p \leq 2^n - 1$.

1. Obtain the on-set COC maxterm coefficients from Fixed polarity COC expansion of polarity p.

2. Divide the newly generated on-set COC maxterm coefficients into w segments and modify the coefficients in each segment.

3. Find the covered coefficients and calculate CPOS maxterm coefficients for each segment by using bitwise "INVERSE" and "AND" operations. Those numbers of covered coefficients that are odd number are the on-set CPOS maxterm coefficients.

4. Use XOR operation to derive on-set CPOS maxterm coefficients from on-set CPOS maxterm coefficients with p polarity obtained in Step 3.

5. Output the CPOS expansion from on-set CPOS maxterm coefficients .

4.5 Conversion Method Based on Minterms

In the previous section, the polarity conversion methods are based on on-set CPOS maxterms. In other words, if the conversion is from CPOS expansion to COC expansion, the on-set maxterms of CPOS are involved in the conversion. However, sometimes the number of on-set CPOS maxterms is quite large which increases the conversion time. In addition, most of benchmark circuits are originally in PLA format, which is in AND/OR plane. The minterms conversion method is more convenient and improves the speed of conversion. The theory is proved and the same as mentioned in Corollary 3.3 in Chapter 3. The conversion method based on on-set CSOP minterms is called minterm method.

The multi-segment method based on on-set CSOP minterms is achieved as follows.

Procedure 4.4. *The minterm multi-segment method for conversion from CSOP expansion to fixed polarity COC expansion with any polarity p.*

1. Generate on-set CSOP minterm coefficients from PLA file.
2. On-set CSOP minterm coefficients are XORed with polarity p.
3. Divide the newly generated on-set CSOP minterm coefficients into w segments and modify the coefficients in each segment.
4. Find the coverage of the coefficients and calculate COC maxterm coefficients for each segment by using bitwise "INVERSE" and "AND" operations. Those numbers of covered coefficients that are odd number are the on-set COC maxterm coefficients.
5. Modify coefficient d_{2^n-1} by complementing d_{2^n-1}.
6. Output the fixed polarity COC expansion of polarity p from the on-set COC maxterm coefficients.

Example 4.7. Given a 4-variable function in CPOS form $f(x_3, x_2, x_1, x_0) = \prod(14, 13, 12, 10, 9, 7, 5, 3, 1, 0)$, as shown in Example 4.4, calculate the on-set coefficients of positive polarity COC expansion using minterm method.

d_{15} is calculated as

$$d_{15} = \bigodot \prod \bar{c}_{1111} = \bar{c}_{1111} = 0$$

Similarly d_{14} is calculated as

$$d_{14} = \prod \bar{c}_{1110} = \bar{c}_{1110} \odot \bar{c}_{1111} = \bar{c}_{14} \odot \bar{c}_{15} = 1 \odot 0 = 0$$

The rest of the coefficients are obtained in the same way.

$$d_{13} = 0 \quad d_{12} = 0 \quad d_{11} = 1 \quad d_{10} = 1$$
$$d_9 = 1 \quad d_8 = 0 \quad d_7 = 0 \quad d_6 = 1$$
$$d_5 = 0 \quad d_4 = 0 \quad d_3 = 1 \quad d_2 = 1$$
$$d_1 = 1 \quad d_0 = 1$$

Because the minterm method is used for conversion, d_{15} has to be recalculated. As a result, $d_{15} = c_{15} = 1$.

When $d_j = 0$, the coefficient is included so that the final solution is obtained and is the same as in the previous Examples 4.1 and 4.4.

$$f(x_3, x_2, x_1, x_0) = \bigodot \prod (4, 5, 7, 8, 12, 13, 14)$$

4.6 Experimental Results

The algorithms are implemented in the C language and the programs are complied by the GNU C Complier (GCC) as shown in Algorithm 4.1. The results are obtained using a PC with Intel Pentium IV (2.4 GHz) with 512 MB RAM under RedHat Linux AS 3. The algorithm requires time complexity of $w(w + w * w)$. Since w is equal to $2^{0.5n}$ approximately according to (4.6). Therefore time complexity of the multi-segment algorithm is $O(2^{1.5n})$. Space complexity is $O(2^n)$.

Table 4.1 shows the CPU conversion time in seconds by using multi-segment method, the number of on-set CPOS maxterms and CSOP minterms before conversion, the number of on-set COC maxterms after conversion and the improvement.

Algorithm 4.1 Multi-segment method for conversion from PLA to fixed polarity COC expansions.

q: the possible maximum number of coefficients in each segment
w: the number of segments
d: the COC coefficient
p: the polarity of COC expansion
$CSOP$: the flag indicates on-set CSOP or CPOS coefficients used where "1" and "0"means on-set CSOP and CSOP coefficients are used respectively.
C: the number of on-set COC maxterm coefficients
$coefficients[\,]$: the array stores all CSOP or CPOS coefficients
$D[\,]$: the array stores covered number for the CSOP or CPOS coefficients
$output[\,]$: the array stores COC coefficients
$infile$: input file used to read in on-set CSOP or CPOS coefficients
$outfile$: output file used to output on-set COC coefficients

```
begin
  C = 0;
  coefficients = read_input_and_sort_coefficients (infile, CSOP, p);
  divide_coefficients (q, w, n, coefficients);
  for i = 2^n − 1 to 2^n − 2^n/w do
    for j = 0 to w − 1 do
      find_covered_numbers (D[j]);
    end for
    for j = 0 to w − 1 do
      for k = 0 to w − 1 do
        d = 0;
        if (!(k ∧ j̄))
          d+ = D[k]);
        end if
      end for
      if (d ∧ 0x00000001 == 1)
        output[C + +] = (i − j ∗ 2^n/w);
      end if
    end for
  end for
  if (CSOP)
    modify_coc_coefficients (output, p);
  end if
  output_data(outfile, output);
end algorithm
```

Table 4.1: CPU Conversion time for IWLS93 benchmarks using maxterm and minterm multi-segment methods.

Name	n	CPOS before conv.	CSOP before conv.	COC after conv.	Maxterm multi-seg. method (s)	Minterm multi-seg. method (s)	imp (%)
5xp1	7	76	52	33	0	0	0
9sym	9	92	420	211	0	0	0
alu4	14	6944	9440	291	0.01	0.01	0
apex4	9	512	0	1	0	-	-
b12	15	26624	6144	17	0.02	0.01	50
bw	5	23	9	13	0	0	0
clip	9	256	256	117	0	0	0
con1	7	60	68	19	0	0	0
cps	24	14745200	2032016	8119	526.7	116.25	77.92
duke2	22	3829760	364544	19	38.23	11.94	68.77
ex1010	10	857	167	487	0	0	0
ex5	8	224	32	2	0	0	0
inc	7	80	48	21	0	0	0
misex1	8	224	32	9	0	0	0
misex2	25	33423360	131072	3	2585.7	51.19	98.02
misex3c	14	7680	8704	107	0.01	0.01	0
misex3	14	14848	1536	1785	0.02	0.01	50
pdc	16	60840	4696	33	0.08	0.03	62.5
rd53	5	26	6	15	0	0	0
rd73	7	64	64	21	0	0	0
rd84	8	136	120	37	0	0	0
sao2	10	1006	18	141	0.01	0.01	0
spla	16	49151	16385	11	0.05	0.05	0
squar5	5	23	9	15	0	0	0
table3	14	14900	1484	1912	0.02	0	100
table5	17	130956	116	129	0.22	0.06	72.72
vg2	25	33333248	221184	23	2477.57	90.82	96.33
xor5	5	16	16	5	0	0	0
X5xp1	7	103	25	31	0	0	0
Z9sym	9	92	420	211	0	0	0
Average	-	-	-	-	-	-	30.64

Table 4.2: CPU Conversion time in seconds compared to published work.

Benchmark	n	[34] (s)	[51] (s)	Maxterm multi-seg. method(s)	Minterm multi-seg. method(s)
apex4	9	~0	0	0	0
alu4	14	2.19	-	0.01	0.01
b12	15	3.3	-	0.02	0.01
clip	9	0.06	0	0	0
con1	7	~0	0	0	0
ex1010	10	0.11	0.01	0	0
misex1	8	~0	-	0	0
misex3c	14	1.59	-	0.01	0.01
pdc	16	16.86	-	0.08	0.03
rd84	8	~0	0.05	0	0
spla	16	15.49	0.931	0.05	0.05
table5	17	28.4	9.845	0.22	0.06

The improvement is defined in (4.13).

$$imp = \frac{CPU\ for\ maxterm - CPU\ for\ minterm}{CPU\ for\ maxterm} \times 100\% \qquad (4.13)$$

where $CPU\ for\ maxterm$ and $CPU\ for\ minterm$ stand for the CPU time used for the maxterm and minterm multi-segment methods respectively.

Each set of on-set CSOP minterms was obtained from benchmark in PLA format, in which the "don't cares" are set to 0. The set of corresponding on-set CPOS maxterms was obtained by complimenting the set of on-set CSOP minterms.

The maxterm and minterm multi-segment methods are compared to [34] and [51]. Table 4.2 shows the comparison results in terms of CPU conversion time in seconds. Experimental results in [34] and [51] were performed on PC with Pentium III (1 GHz) CPU with 256 MB RAM under windows and

Pentium IV (2.4 GHz) CPU with 512 MB RAM under windows. "-" stands for not available, "~0" means CPU time is almost zero and n stands for the number of input variables.

4.7 Summary

In this chapter, two algorithms based on the minterm and maxterm multi-segment methods are developed for large functions to overcome the limitation of the map methods in Chapter 3. Both of algorithms can be used for conversion between standard Boolean and COC expansions of any polarity. The maxterm multi-segment method took less than 0.22 seconds if the number of input variables is less than 17 and outperforms significantly the results given in [34] and [51] in terms of conversion time. Furthermore, a minterm method is even more efficient. The minterm multi-segment method achieved speed improvement of 98.02% and 96.33% for misex2 and vg2 with 25 input variables compared to the maxterm method. The average improvement is 30.64% for the 30 tested benchmarks.

Chapter 5

Tabular Based Techniques for Dual Forms of RM Conversion

5.1 Introduction

Map techniques can only perform for functions which are less than or equal to 6 variables. However, it can be generalised for large functions by using tabular technique [5, 10, 87]. Tabular technique is actually derived from the procedure of map folding technique but generalised for any number of variables. In this chapter, two tabular based techniques are presented for dual forms of Reed-Muller conversion. The rest of the chapter is organised as follows. Serial and parallel tabular techniques for conversion between standard Boolean and fixed polarity COC expansions of any polarity are given in Section 5.2 and Section 5.3 respectively. Experimental results of algorithms are shown in Section 5.4.

5.2 Serial Tabular Technique

It is not practical to use map techniques for large functions because of the limitation of map feature. Based on the map folding technique, the serial tabular technique is proposed for conversion between CPOS and fixed polarity COC expansions, which can be used for any number of variables.

Observation 5.1. *If the array of data structure is used to store newly generated coefficients and previous coefficients of the expansion, all the coefficients of the expansion are actually stored in an increasing order and the index of its ordered number is the same number as the coefficient.*

According to Observation 5.1, an array $flag[]$ can be used for storing all coefficients of the expansion. The index of the array $flag[]$ corresponds to the coefficient of the expansion. The content of the array $flag[]$ represents the presence of the coefficient of the expansion. If the CPOS maxterm coefficients are in use, "1"s indicate the CPOS maxterm coefficients are present while "0"s indicate those maxterm coefficients that are not present. If the CSOP minterm coefficients are in use, "0"s indicate the CSOP minterm coefficients are present while "1"s indicate those minterm coefficients that are not present.

The bitwise "AND" operation is used with "mask" to determine whether to generate new coefficient or not, where "mask" is n-bit binary form. The bitwise "XOR" operation is used with "mask" to generate new coefficients. And the bitwise "SHIFT" operation is used to generate new "mask" value for another variable. n "mask"s with n bits are pre-designed to generate new coefficients for the ith variable. For example, if the number of variables is 3, the "mask" value "$(001)_2$", "$(010)_2$" and "$(100)_2$" are used for the Least Significant Bit (LSB), the 2nd bit and the Most Significant Bit (MSB) respectively. By using

bitwise operations "AND" and "XOR" with mask, new coefficients can be easily generated. STT for conversion from CPOS to fixed polarity COC expansion of any polarity is shown in Procedure 5.1. The theory is proved and shown in Corollary 4.3 in Chapter 4.

Procedure 5.1. *Serial tabular technique for conversion from CPOS expansion to fixed polarity COC expansion of any polarity p.*

1. Clear all the contents of the array $flag[\,]$ to "0".
2. Read in on-set maxterms, XOR the on-set maxterms with polarity p and set "1" to the corresponding contents of the array, where $0 \leq p \leq 2^n - 1$ and the newly generated coefficients are used as the index of the array.
3. Whenever the content of the array is "1", generate a new coefficient if the ith bit of the index is "1", where the index of the array is in a binary form and $0 \leq i \leq n - 1$, and then replace the ith bit with "0" but leave others unchanged.
4. Check the existence of the new coefficient. If the content of the array is "0", set "1" to the content of the array. If the content of the array is "1", clear the content of the array to "0".
5. Repeat Steps 3 and 4 for the ith variable of all the indices of the array.
6. Repeat Steps 3 to 5 for the other variables.
7. Only when the content of the array is "1", output the index of the array as the on-set maxterm coefficient of fixed polarity COC expansion of polarity p.
8. Repeat Step 7 for all the indices of the array.

9. Output the COC expansion according to on-set COC maxterm coefficients.

Because logic synthesis benchmark circuits are originally in PLA format, which is in AND/OR plane, the generation of on-set CSOP minterms from PLA file is more convenient than on-set CPOS maxterms. The theory is proved and the same as mentioned in Corollary 3.3 in Chapter 3. Hence, Procedure 5.1 can be modified to Procedure 5.2 as follows.

Procedure 5.2. *Serial tabular technique for conversion from CSOP expansion to fixed polarity COC expansion of any polarity p.*

1. Set all the contents of the array $flag[\,]$ to "1".

2. Check the number of on-set CSOP minterms. If the number of on-set CSOP minterms is less than or equal to 2^{n-1}, on-set CSOP minterms will be used, called minterm method. Otherwise on-set CPOS maxterms will be used, called maxterm method.

3. Read in on-set CSOP minterm coefficients which are generated from PLA file.

4. XOR the on-set CSOP minterm coefficients with polarity p and set "0" to the corresponding contents of the array, where $0 \leq p \leq 2^n - 1$ and the newly generated coefficients are used as the index of the array.

5. If the minterm method is in use, whenever the content of the array is "0", generate a new coefficient if the ith bit of the index is "1", where the index of the array is in a binary form and $0 \leq i \leq n - 1$, and then replace the ith bit with "0" but leave others unchanged. If the maxterm

method is in use, whenever the content of the array is "1", generate a new coefficient in the same way as did for minterm method.

6. Check the existence of the new coefficient, which is used as the index of the array. No matter which method is used, minterm method or maxterm method, if the content of the array is "0", set "1" to the content of the array. If the content of the array is "1", clear the content of the array to "0".

7. Repeat Steps 5 and 6 for the ith variable of all the indices of the array.

8. Repeat Steps 5 to 7 for the other variables.

9. Only when minterm method is in use, modify the content of the array by complementing the content of the array $flag[2^n - 1]$.

10. If the minterm method is used, output the index of the array as the on-set COC maxterm coefficient of polarity p when the content of the array is "0". If the maxterm method is used, output the index of the array as the on-set COC maxterm coefficient of polarity p when the content of the array is "1".

11. Repeat Step 10 for all the indices of the array.

12. Output the COC expansion according to on-set COC maxterm coefficients.

The Pseudo code of STT from CSOP minterm coefficients to fixed polarity COC expansion of any polarity is given in Algorithm 5.1. Appendix B shows the CSOP minterm and CPOS maxterm coefficients and COC maxterm coefficients files format. The CSOP minterm coefficients are generated from the PLA file as shown in Appendix C.

Algorithm 5.1 Serial tabular technique for conversion from CSOP minterm coefficients to fixed polarity COC maxterm coefficients.

n: the number of variables $coefficients$: the number of coefficients
$mask$: hex constant new: newly generated coefficient
$polarity$: the polarity of fixed polarity COC expansion
$CSOP$: the flag indicates on-set CSOP minterm coefficients used
$flag[\,]$: an array indicates the occurrence of coefficients
$infile$: input file used to read on-set CSOP minterm coefficients
$outfile$: an output file used to output fixed polarity COC maxterm coefficients

```
begin
  mask = 0x00000001;
  read_input_data(infile, n, coefficients, flag, polarity)
  begin
    if (coefficients ≤ 2^(n-1))
      CSOP = 1;
    else
      coefficients = 2^n − coefficients;
    allocate_flag_and _set_flag (flag, CSOP);
    XOR_coefficients_with_polarity (coefficients, flag, polarity);
  end
  for i = 0 to n − 1 do
    for j = 0 to 2^n − 1 do
      if ( !CSOP&& flag[j] == 1) //1 means present
        if ((i ∧ mask) == 1)
          new = i ⊕ mask; // generate new coefficient
          if (flag[new] == 1) // check the existence
            flag[new] = 0;
          else
            flag[new] = 1;
      else if (CSOP&& flag[j] == 0) // 0 means term present
        if ((i ∧ mask) == 1)
          new = i ⊕ mask; // generate new coefficient
          if (flag[new] == 1) // check the existence
            flag[new] = 0;
          else
            flag[new] = 1;
    end for
    mask = mask ≪ 1; // left shift one bit for next variable
  end for
  if (CSOP)
    flag[2^n − 1] =!flag[2^n − 1];
  output_data (outfile, flag, polarity);
end algorithm
```

	Minterms				Minterms after XOR			
	x_2	x_1	x_0			x_2	x_1	x_0
0	0	0	0		0			
1	0	0	1		1			
2					2	0	1	0
3	0	1	1		3	0	1	1
4	1	0	0		4			
5	1	0	1		5	1	0	1
6					6	1	1	0
7					7	1	1	1
	(a)					(b)		

Figure 5.1: A list of on-set CSOP minterm coefficients. (a) Minterm coefficients before XOR, (b) Minterm coefficients after XORing with "110" for polarity 6.

Example 5.1. Obtain the on-set maxterm coefficients of fixed polarity COC expansion of polarity 6 by using STT for a 3-variable function $f(x_2, x_1, x_0) = \bar{x}_2\bar{x}_1\bar{x}_0 + \bar{x}_2\bar{x}_1x_0 + \bar{x}_2x_1x_0 + x_2\bar{x}_1\bar{x}_0 + x_2\bar{x}_1x_0$.

Firstly, allocate and initialise $flag[8]$ so that $flag[8] = \{1, 1, 1, 1, 1, 1, 1, 1\}$. Since the number of on-set CSOP minterms is 5, which is greater than $2^{n-1} = 2^{3-1} = 4$, maxterm method is used. The 3-variable function in CSOP form is $f(x_2, x_1, x_0) = \sum(0, 1, 3, 4, 5)$. Read in all the on-set CSOP minterms as "0"s and update $flag[8]$ so that $flag[8] = \{1, 1, 0, 0, 1, 0, 0, 0\}$ by XORing on-set CSOP minterm coefficients with "110" for polarity 6, as shown in Figure 5.1(a) and Figure 5.1(b) respectively.

Next, start with variable x_0 to generate new coefficients. The mask value for variable x_0 is "$(001)_2$". The first coefficient found in the array with "1" is $flag[0]$, i.e., "$(000)_2$" is found. Since the LSB is "0", do nothing. The second coefficient found in the array with "1" is $flag[1]$, i.e., "$(001)_2$" is found. Since the LSB is "1", a new coefficient "$(000)_2$", as shown in Figure 5.2, is generated by performing XOR between the coefficient "$(001)_2$" and the mask "$(001)_2$".

	Maxterms	New terms (x_0)	New maxterms
	$x_2\ x_1\ x_0$	$x_2\ x_1\ x_0$	$x_2\ x_1\ x_0$
0	0 0 0	0 0 0	
1	0 0 1		0 0 1
2			
3			
4	1 0 0		1 0 0
5			
6			
7			

Figure 5.2: Maxterms generation for variable x_0.

	Maxterms	New terms (x_1)	New maxterms
	$x_2\ x_1\ x_0$	$x_2\ x_1\ x_0$	$x_2\ x_1\ x_0$
0			
1	0 0 1		0 0 1
2			
3			
4	1 0 0		1 0 0
5			
6			
7			

Figure 5.3: Maxterms generation for variable x_1.

	Maxterms	New terms (x_1)	New maxterms
	$x_2\ x_1\ x_0$	$x_2\ x_1\ x_0$	$x_2\ x_1\ x_0$
0		0 0 0	0 0 0
1	0 0 1		0 0 1
2			
3			
4	1 0 0		1 0 0
5			
6			
7			

Figure 5.4: Maxterms generation for variable x_2.

The content of the array is updated so that $flag[8] = \{0, 1, 0, 0, 1, 0, 0, 0\}$. The last coefficient found in the array with "1" is $flag[4]$, i.e., "$(100)_2$" is found. Since the LSB is "0" , do nothing. The generation of new coefficients for x_0 is completed.

The following procedure for x_1 and x_2 is straightforward and is in the same way as for x_0. The arrays are $flag[8] = \{0,1,0,0,1,0,0,0\}$ and $flag[8] = \{1,1,0,0,1,0,0,0\}$ after generation of new coefficients for x_1 and x_2 respectively, as shown in Figure 5.3 and Figure 5.4. It can be seen that the on-set maxterm coefficients of fixed polarity COC expansion of polarity 6 presented in $flag[8]$ are 0, 1 and 4 when the content of the array is "1". Hence,

$$f(\bar{x}_2, \bar{x}_1, x_0) = \bigodot \prod (0,1,4) = (\bar{x}_2 + \bar{x}_1 + x_0) \odot (\bar{x}_2 + \bar{x}_1) \odot (\bar{x}_1 + x_0)$$

The conversion from fixed polarity COC expansion of any polarity p to CPOS expansion is the reserve of the conversion from CPOS expansion to fixed polarity COC expansion of any polarity p. The theory is proved and shown in Corollary 4.4 in Chapter 4.

Procedure 5.3. *Serial tabular technique for conversion from fixed polarity COC expansion of any polarity p to CPOS expansion.*

1. Clear all the contents of the array $flag[\,]$ to "0".

2. Check the number of on-set COC maxterms. If the number of on-set COC maxterms is less than or equal to 2^{n-1}, on-set COC maxterms will be used and called maxterm method. Otherwise off-set COC maxterm will be used and called minterm method.

3. Read in on-set COC maxterm coefficients and set "1"s to contents of the array, where the on-set COC maxterm coefficients are used as the index of the array.

4. If the maxterm method is used, whenever the content of the array is "1", generate a new coefficient if the index of the array is in a binary form and the ith bit of the index is "1", where $0 \leq i \leq n-1$, and then replace the ith bit with "0" but leave others unchanged. If the minterm method is used, whenever the content of the array is "0", generate a new coefficient in the same way as in maxterm method.

5. Check the existence of the new coefficient, where the new coefficient is used as the index of the array. No matter which method is used, minterm method or maxterm method, if the content of the array is "0", set "1" to the content of the array. If the content of the array is "1", clear the content of the array to "0".

6. Repeat Steps 4 and 5 for the ith variable of all the indices of the array.

7. Repeat Steps 4 to 6 for the other variables.

8. Modify the content of the array by complementing the content of the array $flag[2^n - 1]$, only when minterm method is in use.

9. If the maxterm method is used, when the content of the array is "1", XOR the index of the array in binary form with polarity p and output it as the on-set maxterm coefficient of CPOS expansion, where $0 \leq p \leq 2^n - 1$. If the minterm method is used, when the content of the array is "0", XOR the index of the array with polarity p and output it as the on-set maxterm coefficient of CPOS expansion.

10. Repeat Step 9 for all the indices of the array.

11. Output the CPOS expansion according to on-set CPOS maxterm coefficients.

Example 5.2. Obtain on-set maxterm coefficients of CPOS expansion by using STT for a 3-variable function in COC form $f(\bar{x}_2, \bar{x}_1, x_0) = \bigodot \prod(0, 1, 4)$ for the on-set maxterm coefficients of fixed polarity COC expansion of polarity 6.

Firstly, allocate and initialise $flag[8]$ so that $flag[8] = \{0, 0, 0, 0, 0, 0, 0, 0\}$. Since the number of on-set COC maxterms is 3, which is less than $2^{n-1} = 2^{3-1} = 4$, maxterm method is used. Read in all the on-set COC maxterms as "1"s and update $flag[8]$ so that $flag[8] = \{1, 1, 0, 0, 1, 0, 0, 0\}$.

The procedure to generate respective coefficients for x_0, x_1 and x_2 is straightforward. The arrays are updated so that $flag[8] = \{0, 1, 0, 0, 1, 0, 0, 0\}$, $flag[8] = \{0, 1, 0, 0, 1, 0, 0, 0\}$, and $flag[8] = \{1, 1, 0, 0, 1, 0, 0, 0\}$ after generation of new coefficients for x_0, x_1 and x_2 respectively.

The on-set CPOS maxterm coefficients are generated by XORing the index of the array with "110" for polarity 6 so that $flag[8] = \{0, 0, 1, 0, 0, 0, 1, 1\}$, where the contents of the array is "1" for the corresponding index. Hence,

$$f(x_2, x_1, x_0) = \prod(2, 6, 7) = (x_2 + \bar{x}_1 + x_0)(\bar{x}_2 + \bar{x}_1 + x_0)(\bar{x}_2 + \bar{x}_1 + \bar{x}_0)$$

5.3 Parallel Tabular Technique

STT deals with one variable at a time. To overcome the inherent serial aspect of STT, parallel tabular technique is proposed to generate new terms for all new COC maxterms in parallel. Similar to the implementation of STT, PTT also uses an array to deal with all the maxterm coefficients regardless of the number of on-set maxterm coefficients. PTT for conversion from CPOS to fixed polarity COC expansion of any polarity is shown in Procedure 5.4.

Procedure 5.4. *Parallel tabular technique for conversion from CPOS expansion to fixed polarity COC expansion of any polarity p.*

1. Clear all the contents of the array $flag[\,]$ to "0".

2. Read in on-set CPOS maxterms, XOR the on-set CPOS maxterms with polarity p and set "1"s to the corresponding contents of the array, where $0 \leq p \leq 2^n - 1$ and the newly generated coefficients are used as the index of the array.

3. Whenever the content of the array is "1", generate all possible new coefficients. The generation of the new coefficients is as follows. If the index of the array is in a binary form and the ith bit of the index is "1", consider it as "don't care", where $0 \leq i \leq n-1$. Based on the new binary form with "don't care" but excluding the old CPOS maxterm coefficient, $2^o - 1$ new coefficients are generated for each CPOS maxterm coefficient, where o is the number of "1"s in the CPOS maxterm coefficient. The newly generated coefficients are used as the index of the array and the corresponding contents of the array are all incremented by one.

4. Count the number of the content of the array. Only when the number is odd, the index is the on-set COC maxterm coefficient of polarity p.

5. Repeat Step 4 for all the indices of the array.

6. Output the COC expansion according to on-set COC maxterm coefficients.

Procedure 5.4 can be modified to Procedure 5.5. PTT for conversion from CSOP to fixed polarity COC expansion of any polarity is shown in Procedure 5.5 as follows.

Procedure 5.5. *Parallel tabular technique for conversion from CSOP expansion to fixed polarity COC expansion of any polarity p.*

1. Check the number of on-set CSOP minterms. If the number of on-set CSOP minterms is less than or equal to 2^{n-1}, on-set CSOP minterms will be used and clear all the contents of the array $flag[\,]$ to "0". The method is called minterm method. Otherwise on-set CPOS maxterms will be used and set all the contents of the array $flag[\,]$ to "1". The method is called maxterm method.

2. Read in on-set CSOP minterms which are generated from PLA file.

3. XOR the on-set CSOP minterms with polarity p, where $0 \leq p \leq 2^n - 1$. If minterm method is used, set "1" to the corresponding contents of the array. If maxterm method is used, set "0" to the corresponding contents of the array. The newly generated coefficients are used as the index of the array.

4. Whenever the content of the array is "1", generate all possible new coefficients. The generation of the new coefficients is as follows. If the index of the array is in a binary form and the ith bit of the index is "1", consider it as "don't care", where $0 \leq i \leq n - 1$. Based on the new binary form with "don't care" but excluding the old coefficient, $2^o - 1$ new coefficients are generated for each coefficient, where o is the number of "1"s in the old coefficient. The newly generated coefficients are used as the index of the array and the corresponding contents of the array are all added one.

5. Only when minterm method is in use, modify the content of the array by complementing the content of the array $flag[2^n - 1]$.

6. Count the number of the content of the array. Only when the number is odd, the index is the on-set COC maxterm coefficient of polarity p.

7. Repeat Step 6 for all the indices of the array.

8. Output the COC expansion according to on-set COC maxterm coefficients.

The Pseudo code of PTT from CSOP minterm coefficients to fixed polarity COC expansion of any polarity is given in Algorithm 5.2.

Example 5.3. Obtain the on-set maxterm coefficients of fixed polarity COC expansion of polarity 6 by using PTT for a 3-variable function in CSOP form $f(x_2, x_1, x_0) = \sum(0, 1, 3, 4, 5)$, as shown in Example 5.1.

Firstly, the number of on-set CSOP minterms is 5, which is greater than $2^{n-1} = 2^{3-1} = 4$. As a result, maxterm method is used. Allocate and initialise $flag[8]$ so that $flag[8] = \{1, 1, 1, 1, 1, 1, 1, 1\}$. Read in all the on-set CSOP minterms as "0"s which is shown in Figure 5.5 and update $flag[8]$ so that $flag[8] = \{1, 1, 0, 0, 1, 0, 0, 0\}$ by XORing on-set CSOP minterm coefficients with "110" for polarity 6.

Then, the first coefficient found in the array with "1" is "$(000)_2$". Since none of the bits is "1", no new coefficients are generated. The second coefficient found in the array with "1" is "$(001)_2$". Since the LSB is "1", this bit is considered as a "don't care". Thus, two new coefficients "$(000)_2$" and "$(001)_2$" are generated. But the old coefficient "$(001)_2$" should be excluded. As a result, only coefficient "$(000)_2$" is added to the array and $flag[8]$ is updated as $flag[8] = \{2, 1, 0, 0, 1, 0, 0, 0\}$.

The same procedure is applied to coefficient "$(100)_2$". Consequently, $flag[8]$

Algorithm 5.2 Parallel technique technique for conversion from CSOP minterm coefficients to fixed polarity COC maxterm coefficients.

n: the number of variables
$coefficients$: the number of coefficients
o: number of ones
$mask$: hex constant
$flag[\,]$: an array indicates the occurrence of coefficients
$CSOP$: the flag indicates on-set CSOP minterm coefficients used
$polarity$: the polarity of fixed polarity COC expansion
$infile$: input file used to read in on-set CSOP minterm coefficients
$outfile$: output file used to output fixed polarity COC maxterm coefficients

```
begin
  mask = 0x00000001;
  read_input_data(infile, n, coefficients, flag, polarity)
  begin
    if (coefficients ≤ 2^(n-1))
      CSOP = 1;
    else
      coefficients = 2^n − coefficients;
    end if
    allocate_flag_and _set_flag (flag, CSOP);
    XOR_coefficients_with_polarity (coefficients, flag, polarity);
  end
  for index1 = 0 to coefficients − 1 do
    o = count_the_no_of_one (i);
    save_the_position_of_ones ();
    for index2 = 1 to 2^o − 1 do
      generate_possible_new_terms (index2, mask, index3);
      flag[index3]+ = 1;
    end for
  end for
  if (CSOP)
    flag[2^n − 1] =!flag[2^n − 1];
  end if
  for index1 = 0 to 2^n − 1
    if (flag[index1] ∧ mask == 1) // odd number
      output_data (outfile, polarity);
    end if
  end for
end algorithm
```

	Maxterms $x_2\ x_1\ x_0$	New maxterms
0	0 0 0	0 0 0, 0 0 0
1	0 0 1	
2		
3		
4	1 0 0	
5		
6		
7		

Figure 5.5: Newly generated maxterm.

	Maxterms $x_2\ x_1\ x_0$	Number of occurrence
0	0 0 0	3
1	0 0 1	1
2	0 1 0	0
3	0 1 1	0
4	1 0 0	1
5	1 0 1	0
6	1 1 0	0
7	1 1 1	0

COC Maxterms $x_2\ x_1\ x_0$
0 0 0
0 0 1
1 0 0

Figure 5.6: COC maxterms.

is updated as $flag[8] = \{3, 1, 0, 0, 1, 0, 0, 0\}$. Figure 5.5 shows the corresponding newly generated coefficients.

Those indices with odd number are the on-set coefficients of fixed polarity COC expansion of polarity 6. It can be seen in Figure 5.6 that the on-set COC maxterm coefficients are 0, 1 and 4, which are the same as those obtained by using STT.

The conversion from fixed polarity COC expansion of any polarity p to CPOS expansion is the reserve of the conversion from CPOS expansion to fixed polarity COC expansion of any polarity p.

Procedure 5.6. *Parallel tabular technique for conversion from fixed polarity COC expansion of any polarity p to CPOS expansion.*

1. Check the number of on-set COC maxterms. If the number of on-set COC maxterms is less than or equal to 2^{n-1}, clear all the contents of the array $flag[\,]$ to "0". The method is called maxterm method. Otherwise set all the contents of the array $flag[\,]$ to "1". The method is called minterm method.

2. Read in on-set COC maxterms and set "1"s to contents of the array, where the on-set COC maxterms are used as the index of the array.

3. Whenever the content of the array is "1", generate all possible new coefficients. The generation of the new coefficients is as follows. If the index of the array is in a binary form and the ith bit of the index is "1", consider it as "don't care", where $0 \leq i \leq n-1$. Based on the new binary form with "don't care" but excluding the old coefficient, $2^o - 1$ new coefficients are generated for each old coefficient, where o is the number of "1"s in the old coefficient. The newly generated coefficients are used as the index of the array and the corresponding contents of the array are all added one.

4. Only when minterm method is in use, modify the content of the array by complementing the content of the array $flag[2^n - 1]$.

5. Count the number of the content of the array. Only when the number is odd, XOR the index of the array with polarity p and output it as the on-set maxterm coefficient of CPOS expansion, where $0 \leq p \leq 2^n - 1$.

6. Repeat Step 5 for all the indices of the array.

7. Output the CPOS expansion according to on-set CPOS coefficients.

Example 5.4. Obtain on-set maxterm coefficients of CPOS expansion by using PTT for a function in COC form $f(\bar{x}_2, \bar{x}_1, x_0) = \bigodot \prod(0, 1, 4)$ for the on-set maxterm coefficients of fixed polarity COC expansion of polarity 6.

Firstly, since the number of on-set COC maxterms is 3, which is less than $2^{n-1} = 2^{3-1} = 4$, maxterm method is used. Allocate and initialise $flag[8]$ so that $flag[8] = \{0, 0, 0, 0, 0, 0, 0, 0\}$. Read in all the on-set COC maxterm as "1"s and update $flag[8]$ so that $flag[8] = \{1, 1, 0, 0, 1, 0, 0, 0\}$.

The arrays are $flag[8] = \{1, 1, 0, 0, 1, 0, 0, 0\}$, $flag[8] = \{2, 1, 0, 0, 1, 0, 0, 0\}$ and $flag[8] = \{3, 1, 0, 0, 1, 0, 0, 0\}$ respectively after the generation of new coefficients for each on-set maxterm coefficients using PTT.

The on-set CPOS maxterm coefficients are generated by XORing the index of the array with "110" for polarity 6, where the contents of the array is odd number for the corresponding index. Hence, $f(x_2, x_1, x_0) = \prod(2, 6, 7)$, which are the same coefficients as in Example 5.2.

5.4 Experimental Results

The algorithms are implemented in the C language and the programs are complied by the GCC. The results are obtained using a PC with Intel Celeron 897 with 256 MB RAM under Linux. 3 sets of on-set CPOS maxterm coefficients are randomly generated to test the effectiveness of the algorithms. One set has 30 on-set CPOS maxterm coefficients. Another has 300. The other has 3000. Each set of coefficients is tested for both STT and PTT. When the number of variables $n < 15$, the CPU time is almost zero for both algorithms. The CPU time is slightly increased as the number of variables increases. Figure 5.7 and Figure 5.8 show the CPU time in seconds for different variables.

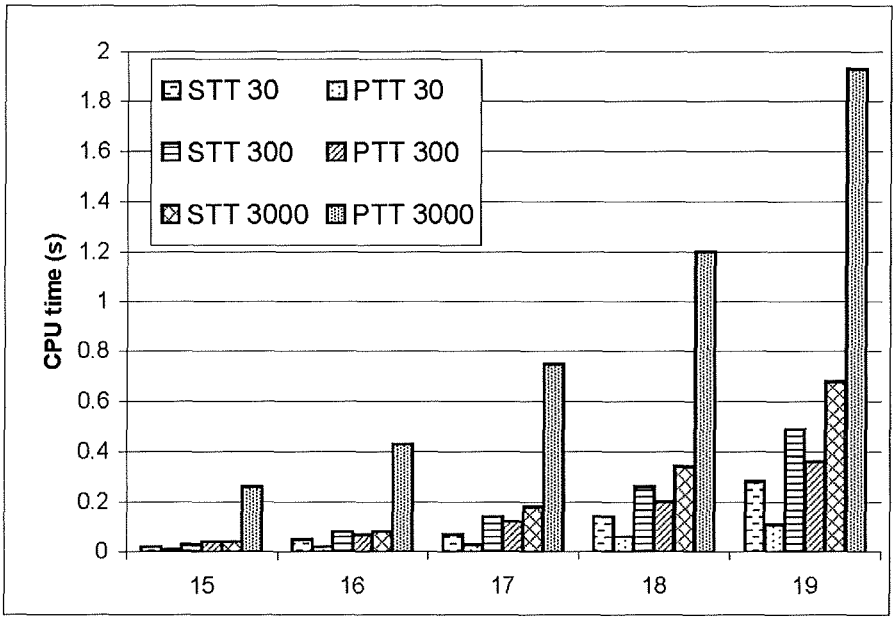


Figure 5.7: CPU conversion time for randomly generated CPOS expansions with 30, 300 and 3000 on-set CPOS maxterm coefficients when $15 \leq n \leq 19$.

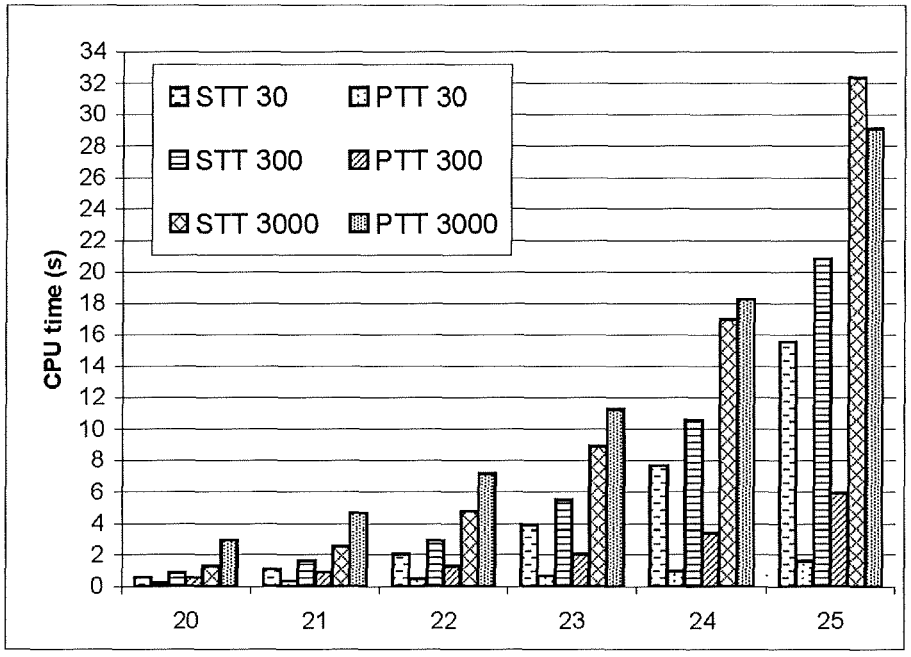


Figure 5.8: CPU conversion time for randomly generated CPOS expansions with 30, 300 and 3000 on-set CPOS maxterm coefficients with $20 \leq n \leq 25$.

Table 5.1: Comparison conversion CPU time for IWLS93 benchmarks.

Name	n	[34] (s)	[51] (s)	Multi-seg. tech. using maxterms (s)	Multi-seg. tech. using minterms (s)	STT (s)	PTT (s)
apex4	9	~0	0	0	0	0	~0
alu4	14	2.19	-	0.01	0.01	0.01	0.32
b12	15	3.3	-	0.02	0.01	0.02	0.53
clip	9	0.06	0	0	0	0	0
con1	7	~0	0	0	0	0	0
misex1	8	~0	-	0	0	0	0
misex3c	14	1.59	-	0.01	0.01	~0	0.06
pdc	16	16.86	0.931	0.08	0.03	0.11	0.13
rd84	8	~0	0.05	0	0	0	0
spla	16	15.49	0.931	0.05	0.05	0.11	0.07
table5	17	28.4	9.845	0.22	0.06	0.2	0.15

Our methods are compared to [34] and [51]. Table 5.1 shows the comparison results, in which experimental results in [34] and [51] were performed on a PC with Pentium III (1 GHz) CPU with 256 MB RAM under windows and Pentium IV (2.4 GHz) CPU with 512 MB RAM under windows. "~0" means CPU time almost zero, n stands for the number of variables. In all cases, STT and PTT outperform [34] and [51] significantly.

Time complexity of the STT and PTT are $O(n2^n)$ and $O(coefficients2^o)$ respectively, where o is the number of "1"s in each maxterm and $coefficients$ is the number of maxterms or minterms. Space complexity of the STT and PTT are both $O(2^n)$.

Table 5.2 shows the CPU time in seconds for maxterm STT method, minterm STT method for large randomly generated functions and percentage of improvement, where first column is the number of variables, the second one is the number of on-set CPOS maxterms before conversion and the third

Table 5.2: STT using maxterm and minterm methods.

n	no. of CPOS before conversion	no. of COC after conversion	maxterm STT (s)	minterm STT (s)	imp. %
20	1048546	146737	1.96	1.88	4.08
20	1048276	308855	2.14	2.05	4.21
20	1045576	461047	2.33	2.26	3.00
21	2097122	285795	4.15	3.88	6.51
21	2096852	499257	4.38	4.09	6.62
21	2094152	885755	4.8	4.66	2.92
22	4194274	337843	8.25	7.75	6.06
22	4194004	724669	8.65	8.4	2.89
22	4191304	1622279	9.67	9.28	4.03
23	8388578	372491	16.54	15.63	5.50
23	8388308	1118017	17.28	16.21	6.19
23	8385608	2962567	19.42	18.55	4.48
24	16777186	428775	33.57	31.84	5.15
24	16776916	1953563	34.78	33.64	3.28
24	16774216	5485627	39.07	37.16	4.89
25	33554402	570823	69.74	66.03	5.32
25	33554132	3360689	72.17	68.15	5.57
25	33551432	9989185	79.96	76.00	4.95
Ave.	11008938	1750776	23.83	22.64	4.99

one is the number of on-set COC maxterms after conversion.

The improvement rate is defined in (5.1).

$$imp = \frac{CPU\ in\ maxterm\ method - CPU\ in\ minterm\ method}{CPU\ in\ maxterm\ method} \times 100\% \quad (5.1)$$

where *CPU in maxterm method* and *CPU in minterm method* stand for the CPU time used for maxterm and minterm methods respectively.

5.5 Summary

STT and PTT for conversion between standard Boolean and fixed polarity COC expansions of any polarity are proposed in this chapter. By using bitwise operations and paying penalty of memory requirement, STT outperforms the other methods in the literature. PTT generates new terms in parallel instead of dealing with one variable at a time. However, as the number of product terms increases the performance of PTT degrades and can not achieve better performance than STT because of overhead computation. As a result, PTT is suitable for large sparse functions. While STT can be used when PTT can not perform well for those functions with a large number of coefficients. Minterm STT method can achieve improvement of 4.99% over maxterm STT method in terms of CPU conversion time for randomly generated large functions.

Chapter 6

On-set Table Method for Multi-level Mixed Polarity RM

6.1 Introduction

Conversion algorithms between standard Boolean and DFRM expansions have been investigated in [118, 140]. Furthermore, many optimisation techniques for two-level FPRM and MPRM forms were proposed in terms of area minimisation and/or power minimisation in [121, 127, 130, 131, 133]. The method in [121] proposed an exact minimisation by exhaustively searching all the possible fixed polarities to find the best polarity with least number of π-terms. For an n-variable function, there are 2^n polarities of the FPRM forms. It would be difficult to exhaustively search all possible MPRM forms since the number of polarities could be $2^{n2^{n-1}}$. Recently, truth vector based method for MMPRM optimisation was proposed in [131]. This method uses a truth vector with length of 2^n to represent an n-variable FPRM expansion. By elimination and decomposition, the compact representation of MMPRM form can be obtained

from the FPRM expansion. The main disadvantage of this method is the rapid increase in memory for large functions.

In this chapter, a novel onset table method is presented to obtain the MMPRM form directly from the FPRM expansion, which requires less memory so that it can target large functions. The rest of the chapter is organised as follows. Section 6.2 gives the properties of onset table. In Section 6.3, extraction of common variables from on-set table is given. The proposed algorithm for the optimisation of MMPRM is detailed in Section 6.4. Experimental results are presented in Section 6.5.

6.2 Properties of On-set Table and Basic Definitions

Any n-variable Boolean function can be expressed canonically by the SOP form in (6.1).

$$f(\dot{x}_{n-1}, \cdots, \dot{x}_1, \dot{x}_0) = \oplus \sum_{j=0}^{2^n-1} b_j \pi_j \tag{6.1}$$

where "$\oplus \sum$" is the XOR operator, $b_j \in \{0, 1\}$ and "1"s indicate the presence of the corresponding π-terms in the expansion. The subscript j is expressed in the binary form as $j = (j_{n-1} \cdots j_1 j_0)$. The π-term can be expressed as

$$\pi_j = \dot{x}_{n-1} \cdots \dot{x}_1 \dot{x}_0 \tag{6.2}$$

$$\dot{x}_i = \begin{cases} 1 & j_i = 0 \\ \dot{x}_i & j_i = 1 \end{cases} \tag{6.3}$$

where $0 \leq i \leq n-1$.

Definition 6.1. Any n-variable FPRM expansion of any polarity can be expressed with Set $\mathcal{O}$, which is composed of the decimal numbers equivalent to the coefficients of π-terms.

Example 6.1. Given a 3-variable function as an FPRM expansion of any polarity $\mathcal{F}(\dot{x}_2, \dot{x}_1, \dot{x}_0) = \dot{x}_0 \oplus \dot{x}_1\dot{x}_0 \oplus \dot{x}_2\dot{x}_0 \oplus \dot{x}_2\dot{x}_1\dot{x}_0$, it can be represented by π-terms as in $\mathcal{F}(\dot{x}_2, \dot{x}_1, \dot{x}_0) = \pi_1 \oplus \pi_3 \oplus \pi_5 \oplus \pi_7$. It also can be expressed by a Set $\mathcal{O}$ as $\mathcal{O} = \{1, 3, 5, 7\}$, in which the decimal numbers are the subscript of b_j in (6.1).

Definition 6.2. Onset table, called $\mathcal{T}$ table (briefly $\mathcal{T}$), is to describe the existence of each variable in each π-term of a FPRM expansion. The $\mathcal{T}$ has the following properties:

1. Each row of the $\mathcal{T}$ represents each element of Set $\mathcal{O}$ in binary form;
2. Each column of the $\mathcal{T}$ represents an input variable of the FPRM expansion;
3. $Q_{si} \in \{0, 1\}$ is a bit on the sth row and ith column of $\mathcal{T}$. $Q_{si} = 1$ means that the variable $\dot{x}_i$ on the ith column and sth row of $\mathcal{T}$ appears in its true form in the expansion. $Q_{si} = 0$ means that the variable $\dot{x}_i$ on the ith column and sth row of $\mathcal{T}$ does not appear in the expansion.

Example 6.2. Given $\mathcal{O} = \{1, 3, 5, 7\}$ for FPRM expansion of positive polarity, the corresponding $\mathcal{T}$ is shown in Figure 6.1.

Definition 6.3. $\mathcal{F} \rightarrow \mathcal{T}$ indicates that the FPRM expansion of any polarity $\mathcal{F}$ is mapped to $\mathcal{T}$. $\mathcal{T} \rightarrow \mathcal{F}$ indicates that the $\mathcal{T}$ is mapped to FPRM expansion of any polarity $\mathcal{F}$.

	x_2	x_1	x_0
π_1	0	0	1
π_3	0	1	1
π_5	1	0	1
π_7	1	1	1

Figure 6.1: An example of $\mathcal{T}$ for a given $\mathcal{O}$.

Corollary 6.1. *Given a FPRM expansion of any polarity $\mathcal{F}$ and $\mathcal{F} \rightarrow \mathcal{T}$. If any two columns of $\mathcal{T}$ are swapped, new $\mathcal{T}'$ is generated but $\mathcal{F} = \mathcal{F}'$, where $\mathcal{T}' \rightarrow \mathcal{F}'$.*

Proof. Swapping any two columns of $\mathcal{T}$ does not change the logic functionality of $\mathcal{F}$ but the order of the variables in $\mathcal{F}$. Hence, $\mathcal{F} = \mathcal{F}'$. □

Corollary 6.2. *Given a FPRM expansion of any polarity $\mathcal{F}$ and $\mathcal{F} \rightarrow \mathcal{T}$. If any two rows of $\mathcal{T}$ are swapped, new $\mathcal{T}'$ is generated but $\mathcal{F} = \mathcal{F}'$, where $\mathcal{T}' \rightarrow \mathcal{F}'$.*

Proof. If any two rows of $\mathcal{T}$ are swapped, it is equivalent to swapping any two on-set coefficients of $\mathcal{F}$, because XOR operator is commutative. As a result, $\mathcal{F} = \mathcal{F}'$. □

Example 6.3. Given a 3-variable function in the FPRM expansion of positive polarity in Example 6.1, $\mathcal{T}'$ is generated after swapping the third row and forth row of $\mathcal{T}$, i.e., π_5 and π_7 are swapped, as shown in Figure 6.2(a). Equation (6.4) shows that the FPRM expansion remains unchanged.

$\mathcal{T}'$ is generated after swapping the first column and third column of $\mathcal{T}$, i.e., variable x_0 and x_2 are swapped, as shown in Figure 6.2(b). Equation (6.5)

	x_2	x_1	x_0
π_1'	0	0	1
π_3'	0	1	1
π_5'	1	1	1
π_7'	1	0	1

(a)

	x_0	x_1	x_2
π_1'	1	0	0
π_3'	1	1	0
π_5'	1	0	1
π_7'	1	1	1

(b)

Figure 6.2: The resulting $\mathcal{T}'$ generated after swapping. (a) π_5 and π_7 are swapped, (b) Variable x_0 and x_2 are swapped.

shows that the FPRM expansion remains unchanged.

$$\begin{aligned} \mathcal{F}'(x_2, x_1, x_0) &= \pi_1' \oplus \pi_3' \oplus \pi_5' \oplus \pi_7' \\ &= \pi_1 \oplus \pi_3 \oplus \pi_7 \oplus \pi_5 \\ &= x_0 \oplus x_1x_0 \oplus x_2x_1x_0 \oplus x_2x_0 \\ &= x_0 \oplus x_1x_0 \oplus x_2x_0 \oplus x_2x_1x_0 \\ &= \mathcal{F}(x_2, x_1, x_0) \end{aligned} \tag{6.4}$$

$$\begin{aligned} \mathcal{F}'(x_2, x_1, x_0) &= \pi_1' \oplus \pi_3' \oplus \pi_5' \oplus \pi_7' \\ &= x_0 \oplus x_0x_1 \oplus x_0x_2 \oplus x_0x_1x_2 \\ &= x_0 \oplus x_1x_0 \oplus x_2x_0 \oplus x_2x_1x_0 \\ &= \mathcal{F}(x_2, x_1, x_0) \end{aligned} \tag{6.5}$$

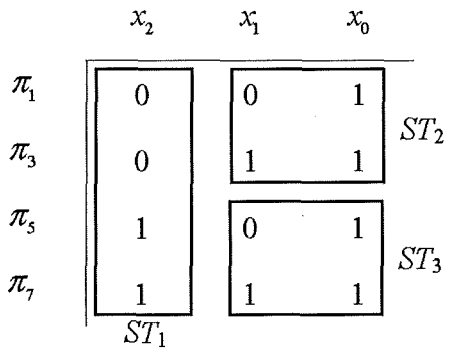


Figure 6.3: 3 possible sub-tables of $\mathcal{T}$.

Definition 6.4. $\mathcal{T}$ can be divided into U sub-tables either in the horizontal or vertical direction, where U is an integer and $U \geq 1$. The sub-table is notated as ST_t, where $0 \leq t \leq U-1$. The sub-table ST_t that includes some of the π-terms of $\mathcal{T}$ in horizontal direction is notated as PST_t. Any sub-table ST_t that includes some of the variables of $\mathcal{T}$ in vertical direction is notated as VST_t.

Example 6.4. If $\mathcal{T}$ shown in Figure 6.1 is divided into 3 sub-tables, as seen in Figure 6.3, $PST_1 = \{\pi_1, \pi_3, \pi_5, \pi_7\}$ and $VST_1 = \{x_2\}$. $PST_2 = \{\pi_1, \pi_3\}$ and $VST_2 = \{x_1, x_0\}$. $PST_3 = \{\pi_5, \pi_7\}$ and $VST_3 = \{x_1, x_0\}$.

Definition 6.5. If sub-table ST includes all the π-terms of the $\mathcal{T}$, it is defined as $WPST$. The variables included by the $WPST$ are notated as $VWPST$. If sub-table ST includes all the variables of the $\mathcal{T}$, it is defined as $WVST$.

Corollary 6.3. *Given any two sub-tables, ST_1 and ST_2, where $ST_1 \subset \mathcal{T}$ and $ST_2 \subset \mathcal{T}$, if $PST_1 = PST_2$, these two sub-tables, ST_1 and ST_2 can be grouped into another sub-table ST_3 in vertical direction without changing the logic functionality of $\mathcal{F}$, where $ST_3 = \{ST_1, ST_2\}$.*

Proof. Because $PST_1 = PST_2$, ST_1 and ST_2 are located at the same row of $\mathcal{T}$, as shown in Figure 6.4(a). $\mathcal{T}$ can be reorganised and generates $\mathcal{T}'$, as shown

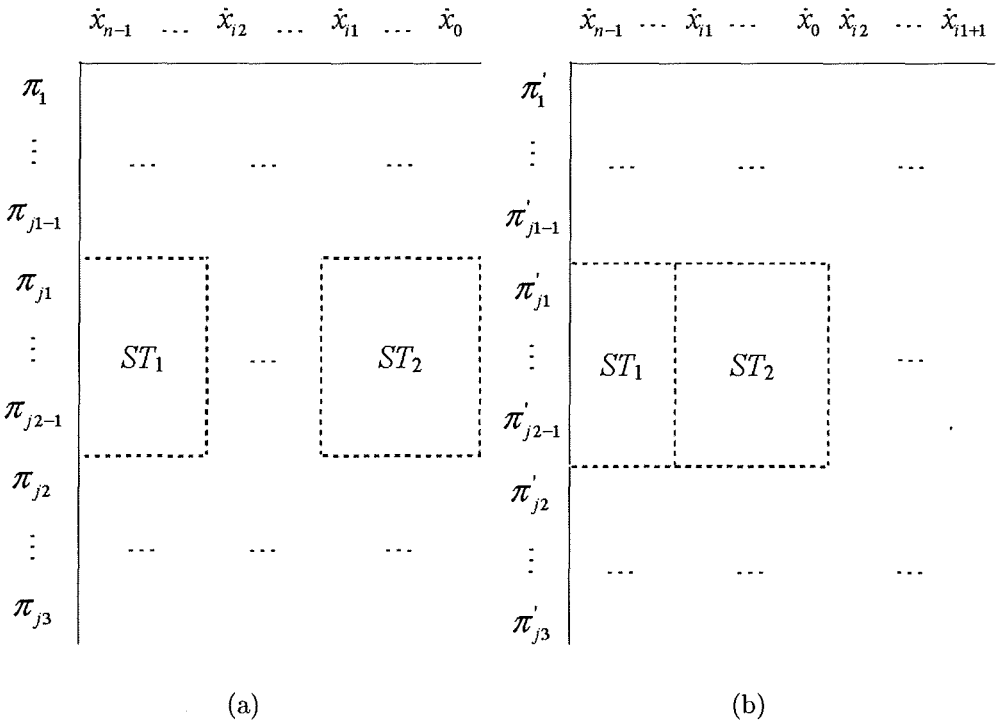


Figure 6.4: Exchange two sub-tables, ST_1 and ST_2 in the vertical direction. (a) $\mathcal{T}$ before exchange, (b) $\mathcal{T}'$ after exchange.

in Figure 6.4(b). Let $\mathcal{T}$ and $\mathcal{T}'$ be on-set tables before and after changing the position of ST_2. Let $\mathcal{F} \rightarrow \mathcal{T}$ and $\mathcal{T}' \rightarrow \mathcal{F}'$. According to Corollary 6.1, $\mathcal{F} = \mathcal{F}'$.

Let $\mathcal{T}''$ be on-set table after grouping ST_1 and ST_2 and $\mathcal{T}'' \rightarrow \mathcal{F}''$, as shown in Figure 6.5. Because after grouping, $\mathcal{T}'' = \mathcal{T}'$. Hence, $\mathcal{F}' = \mathcal{F}''$.

As a result, $\mathcal{F} = \mathcal{F}' = \mathcal{F}''$. □

Corollary 6.4. *Given any two sub-tables, ST_1 and ST_2, where $ST_1 \subset \mathcal{T}$ and $ST_2 \subset \mathcal{T}$, if $VST_1 = VST_2$, these two sub-tables, ST_1 and ST_2 can be grouped into another sub-table ST_3 in horizontal direction without changing the logic functionality of $\mathcal{F}$, where $ST_3 = \{ST_1, ST_2\}$.*

Proof. Because $VST_1 = VST_2$, ST_1 and ST_2 are located at the same column of

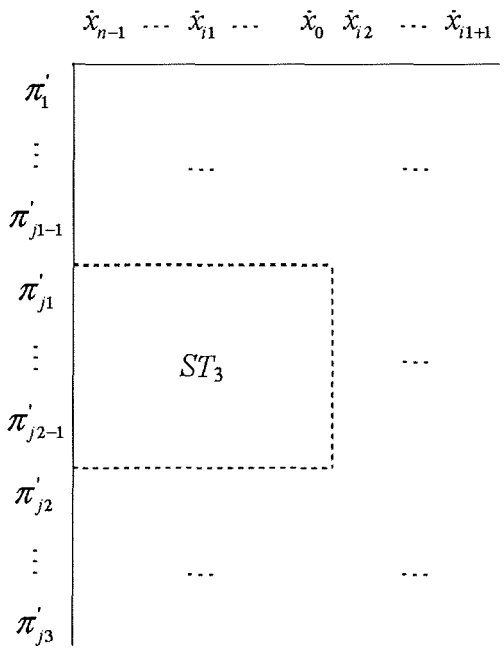


Figure 6.5: Resulting $\mathcal{T}''$ after grouping ST_1 and ST_2 into ST_3.

$\mathcal{T}$, as shown in Figure 6.6(a). $\mathcal{T}$ can be reorganised and generates $\mathcal{T}'$, as shown in Figure 6.6(b). Let $\mathcal{T}$ and $\mathcal{T}'$ be on-set tables before and after changing the position of ST_2. Let $\mathcal{F} \rightarrow \mathcal{T}$ and $\mathcal{T}' \rightarrow \mathcal{F}'$. According to Corollary 6.2, $\mathcal{F} = \mathcal{F}'$.

Let $\mathcal{T}''$ be on-set table after grouping ST_1 and ST_2 and $\mathcal{T}'' \rightarrow \mathcal{F}''$, as shown in Figure 6.7. Because after grouping, $\mathcal{T}'' = \mathcal{T}'$. Hence, $\mathcal{F}' = \mathcal{F}''$.

As a result, $\mathcal{F} = \mathcal{F}' = \mathcal{F}''$. □

Corollary 6.5. *Given a FPRM expansion $\mathcal{F}$, $\mathcal{T}$ and $WPST$, where $\mathcal{F} \rightarrow \mathcal{T}$, $VWPST = \{x_{i2}, x_{i2-1}, \cdots, x_{i1+1}, x_{i1}\}$ and all the elements of $WPST$ are "1". If $\mathcal{T}'$ is generated by deletion of $WPST$ from $\mathcal{T}$ and $\mathcal{T}' \rightarrow \mathcal{F}'$,*

$$\mathcal{F} = s\pi\mathcal{F}' \tag{6.6}$$

where $s\pi = x_{i2}x_{i2-1}\cdots x_{i1+1}x_{i1}$ and $0 \leq i1 \leq i2 \leq n-1$.

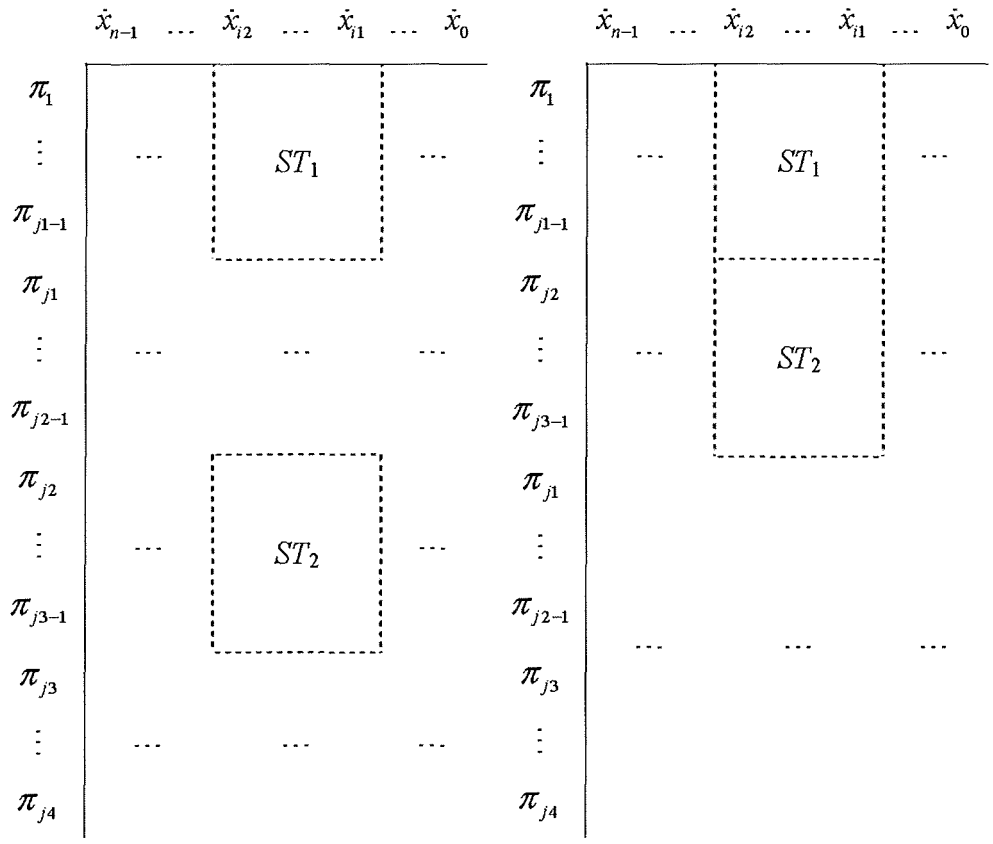


Figure 6.6: Exchange two sub-tables, ST_1 and ST_2 in the horizontal direction. (a) $\mathcal{T}$ before exchange, (b) $\mathcal{T}'$ after exchange.

Proof. Because $\mathcal{F} \rightarrow \mathcal{T}$, $VWPST = \{x_{i2}, x_{i2-1}, \cdots, x_{i1+1}, x_{i1}\}$, $0 \leq i1 \leq i2 \leq n$, and $WPST \subset \mathcal{T}$, the corresponding $\mathcal{T}$ can be drawn as shown in Figure 6.8(a).

Let n-variable function $\mathcal{F} = \oplus \sum_{j=0}^{2^n-1} b_j \pi_j$ be the expansion before extraction and $\mathcal{F}' = \oplus \sum_{j=0}^{2^n-1} b_j' \pi_j'$ be the one after extraction. After extraction of common variables, the number of variables changes but the number of π-terms does not change. Hence $b_j = b_j'$.

For any π_j and π_j', because all the elements of $WPST$ are "1", $\pi_j = (s\pi)\pi_j'$,

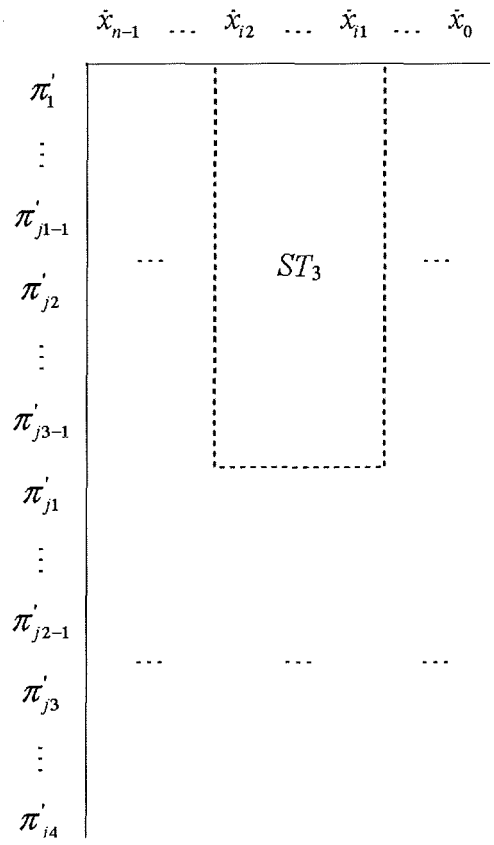


Figure 6.7: Resulting $\mathcal{T}''$ after grouping ST_1 and ST_2 into ST_3.

where $s\pi = x_{i2}x_{i2-1}\cdots x_{i1+1}x_{i1}$. Hence, $\mathcal{F}$ can be rewritten as $\mathcal{F} = s\pi\mathcal{F}'$, namely, $s\pi$ and $\mathcal{F}'$ have the logic "AND" relationship as in (6.6). If $s\pi \rightarrow WPST$, the variables included in the $s\pi$ can be extracted from $\mathcal{F}$, resulting in new $\mathcal{T}'$ generated by deleting $WPST$ from $\mathcal{T}$, as shown in Figure 6.8(b). □

Corollary 6.6. *Given a FPRM expansion $\mathcal{F}$, $\mathcal{T}$ and $WPST$, where $\mathcal{F} \rightarrow \mathcal{T}$, $VWPST = \{x_{i2}, x_{i2-1}, \cdots, x_{i1+1}, x_{i1}\}$ and all the elements of $WPST$ are "0". If $\mathcal{T}'$ is generated by deletion of $WPST$ from $\mathcal{T}$ and $\mathcal{T}' \rightarrow \mathcal{F}'$,*

$$\mathcal{F} = \mathcal{F}' \tag{6.7}$$

where $s\pi = x_{i2}x_{i2-1}\cdots x_{i1+1}x_{i1}$ and $0 \leq i1 \leq i2 \leq n-1$.

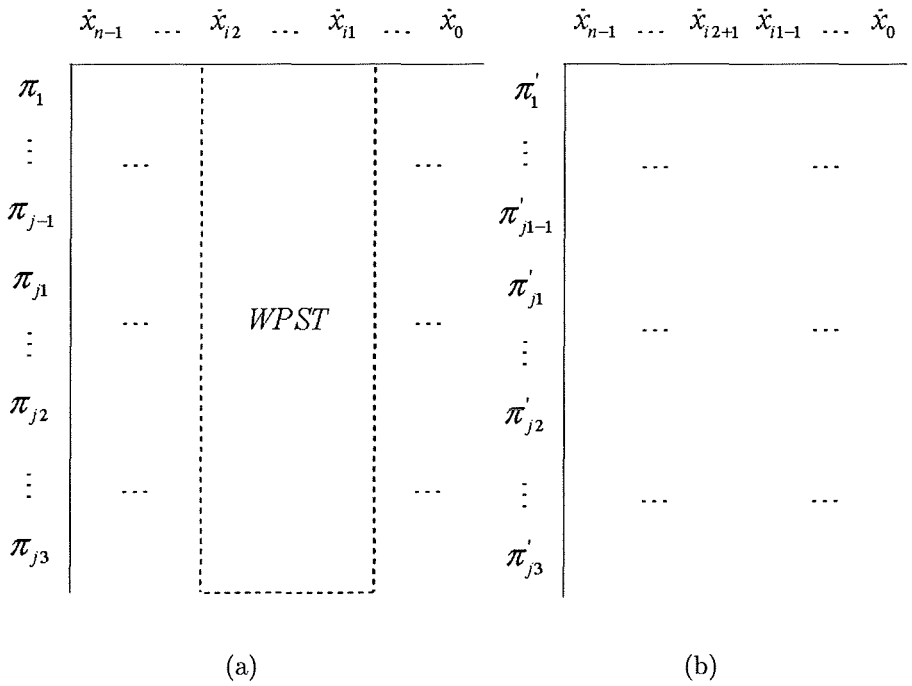


Figure 6.8: Extraction of global common variables. (a) $\mathcal{T}$ before deletion, (b) $\mathcal{T}'$ after deletion.

Proof. Let n-variable function $\mathcal{F} = \oplus \sum_{j=0}^{2^n-1} b_j \pi_j$ be the expansion before extraction and $\mathcal{F}' = \oplus \sum_{j=0}^{2^n-1} b'_j \pi'_j$ be the one after extraction. After extraction of common variables, the number of π-terms does not change. Hence $b_j = b'_j$. $\mathcal{T}'$ is generated from $\mathcal{T}$ by deleting $WPST$, where $VWPST = \{x_{i2}, x_{i2-1}, \cdots, x_{i1+1}, x_{i1}\}$, $0 \leq i1 \leq i2 \leq n$ and all the elements of $WPST$ are "0". According to Definition 6.2, the variables do not appear in the $\mathcal{F}$ if $Q_{si} = 0$, i.e., $VWPST = \{x_{i2}, x_{i2-1}, \cdots, x_{i1+1}, x_{i1}\}$ do not appear either in $\mathcal{F}$ and $\mathcal{F}'$. As a result, $\mathcal{F} = \mathcal{F}'$. □

Example 6.5. Given a 4-variable function in the FPRM expansion of positive polarity $\mathcal{F}(x_3, x_2, x_1, x_0) = x_3 \oplus x_3x_0 \oplus x_3x_1 \oplus x_3x_1x_0$, $VWPST = \{x_3\}$ and all the elements of $WPST$ are "1". Variable x_3 can be extracted from $\mathcal{F}$, as shown in Figure 6.9. Hence, $\mathcal{F}'(x_2, x_1, x_0) = 1 \oplus x_0 \oplus x_1 \oplus x_1x_0$ and $s\pi = x_3$. If

	x_3	x_2	x_1	x_0
π_8	1	0	0	0
π_9	1	0	0	1
π_{10}	1	0	1	0
π_{11}	1	0	1	1

(a)

	x_2	x_1	x_0
π_8'	0	0	0
π_9'	0	0	1
π_{10}'	0	1	0
π_{11}'	0	1	1

(b)

Figure 6.9: On-set table deletion of $WPST$, where $VWPST = \{x_3\}$ and all the elements of $WPST$ are "1". (a) $\mathcal{T}$ before deletion, (b) $\mathcal{T}'$ after deletion.

	x_3	x_2	x_1	x_0
π_8	1	0	0	0
π_9	1	0	0	1
π_{10}	1	0	1	0
π_{11}	1	0	1	1

(a)

	x_3	x_1	x_0
π_8'	1	0	0
π_9'	1	0	1
π_{10}'	1	1	0
π_{11}'	1	1	1

(b)

Figure 6.10: On-set table deletion of $WPST$, where $VWPST = \{x_2\}$ and all the elements of $WPST$ are "0". (a) $\mathcal{T}$ before deletion, (b) $\mathcal{T}'$ after deletion.

the function is expanded, $\mathcal{F}(x_3, x_2, x_1, x_0) = s\pi\mathcal{F}' = x_3(1 \oplus x_0 \oplus x_1 \oplus x_1x_0) = x_3 \oplus x_3x_0 \oplus x_3x_1 \oplus x_3x_1x_0$, the logic functionality remains unchanged.

It can be seen in Figure 6.10(a), $Q_{si} = 0$ for the column of variable x_2 in the $\mathcal{T}$. According to Definition 6.2, x_2 does not appear in the expansion $\mathcal{F}$, where $\mathcal{F} \rightarrow \mathcal{T}$. $\mathcal{T}'$ is generated shown in Figure 6.10(b) after deletion of $VWPST = \{x_2\}$ and all the elements of $WPST$ are "0", where $\mathcal{T}' \rightarrow \mathcal{F}'$.

Although $\mathcal{T}'$ changed, the function does not change. Hence $\mathcal{F} = \mathcal{F}'$.

Definition 6.6. Given a $\mathcal{T}$. Let $ST_1 \subset \mathcal{T}$ and $ST_2 \subset \mathcal{T}$. $ST_1 = ST_2$ if $VST_1 = VST_2$ is satisfied, i.e., the variables included by ST_1 and ST_2 are same and also all the elements of ST_1 and the corresponding elements of ST_2 are bitwise identical.

Corollary 6.7. *Given $\mathcal{F} \rightarrow \mathcal{T}$ and $\mathcal{T}$ consists of two sub-tables, $WVST_1$ and $WVST_2$, where $\mathcal{T} = \{WVST_1, WVST_2\}$. In addition, $SST_1 = SST_2$, where $SST_1 \subset WVST_1$ and $SST_2 \subset WVST_2$. $\mathcal{F}$ can be rewritten as $\mathcal{F} = (s\pi_1 \oplus s\pi_2)\mathcal{F}'$, where $SST_1 \rightarrow \mathcal{F}'$ or $SST_2 \rightarrow \mathcal{F}'$. $s\pi_1$ and $s\pi_2$ are common variables of $WVST_1$ and $WVST_2$ respectively.*

Proof. Let $WVST_1 \rightarrow \mathcal{F}_1$ and $WVST_2 \rightarrow \mathcal{F}_2$. Because $\mathcal{F} \rightarrow \mathcal{T}$ and $\mathcal{T} = \{WVST_1, WVST_2\}$, hence,

$$\mathcal{F} = \mathcal{F}_1 \oplus \mathcal{F}_2 \tag{6.8}$$

Given $SST_1 \rightarrow \mathcal{F}_1'$ and $SST_2 \rightarrow \mathcal{F}_2'$. According to Corollary 6.5,

$$\mathcal{F}_1 = s\pi_1\mathcal{F}_1' \tag{6.9}$$

$$\mathcal{F}_2 = s\pi_2\mathcal{F}_2' \tag{6.10}$$

Equation (6.8) can be rewritten as

$$\mathcal{F} = (s\pi_1\mathcal{F}_1') \oplus (s\pi_2\mathcal{F}_2') \tag{6.11}$$

Because $\mathcal{F}' \rightarrow SST_1$ or $\mathcal{F}' \rightarrow SST_2$ and $SST_1 = SST_2$,

$$\mathcal{F}' = \mathcal{F}_1' = \mathcal{F}_2' \tag{6.12}$$

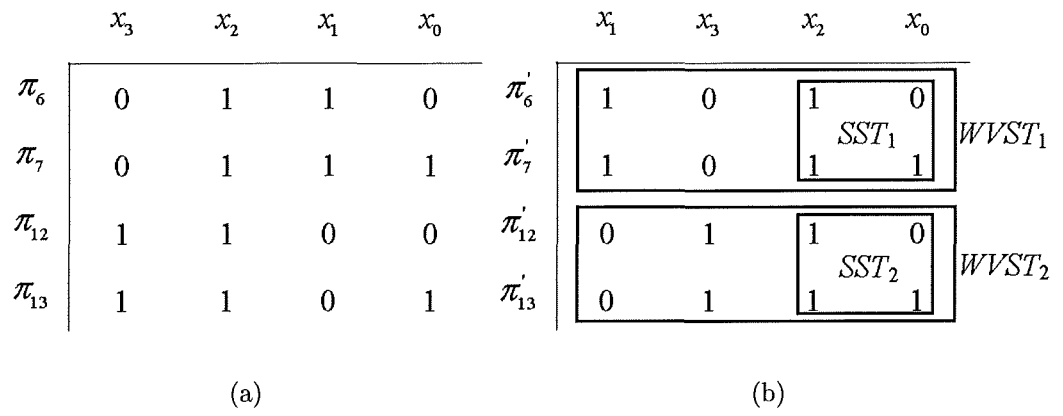


Figure 6.11: Extraction of common sub-table. (a) $\mathcal{T}$ of a 4-variable function in FPRM expansion, (b) The resulting $\mathcal{T}'$ after changing the order of variables.

As a result, $\mathcal{F} = (s\pi_1 \oplus s\pi_2)\mathcal{F}'$. □

Example 6.6. Given a 4-variable function $\mathcal{F}(x_3, x_2, x_1, x_0)$ and its FPRM expansion $\mathcal{F}(x_3, x_2, x_1, x_0) = x_2x_1 \oplus x_2x_1x_0 \oplus x_3x_2 \oplus x_3x_2x_0$, $\mathcal{O} = \{6, 7, 12, 13\}$.

According to $\mathcal{O}$, $\mathcal{T}$ can be obtained as shown in Figure 6.11(a). After exchanging variable order in $\mathcal{T}$, $\mathcal{T}'$ can be obtained as shown in Figure 6.11(b). As it can be seen from Figure 6.11(b), $\mathcal{T}'$ can be divided into two sub-tables, $WVST_1$ and $WVST_2$, where $WVST_1 = \{\pi'_6, \pi'_7\}$ and $WVST_2 = \{\pi'_{12}, \pi'_{13}\}$.

$WVST_1$ can be further divided into sub-table SST_1, where SST_1 is sub-table of $WVST_1$. Similarly, $WVST_2$ can be further divided into sub-table SST_2, where SST_2 is sub-table of $WVST_2$. As it can be seen, $SST_1 = SST_2$ and it includes variables x_2 and x_2x_0. Let $WVST_1 \rightarrow \mathcal{F}_1$, $WVST_2 \rightarrow \mathcal{F}_2$ and $SST_1, SST_2 \rightarrow \mathcal{F}'$. As a result,

$$\mathcal{F} = \mathcal{F}_1 \oplus \mathcal{F}_2 = x_1\mathcal{F}' \oplus x_3\mathcal{F}' = (x_1 \oplus x_3)\mathcal{F}'$$

where $\mathcal{F}' = x_2 \oplus x_2x_0$.

6.3 Extraction of Common Variables

In the process of simplification of the FPRM expansion, the extraction of common variables is of importance. There are two kinds of common variables called global and local common variables, where "global" means the variable exists in all π-terms or does not exist at all in all π-terms while "local" means the variable exists in some of the π-terms.

Definition 6.7. If $WPST$ only includes variable $\dot{x}_i$, the sub-table is notated as $WPST_{\dot{x}i}$. $ST_{\dot{x}i}$ is defined as the sub-table after deletion of $WPST_{\dot{x}i}$ from $\mathcal{T}$, where $WPST_{\dot{x}i} \subset \mathcal{T}$.

According to Corollaries 6.5 and 6.6, the global common variables can be easily obtained. Following is the procedure of extraction of global common variables, assuming R rows and n columns in $\mathcal{T}$ for n variables and R π-terms in the FPRM expansion.

Procedure 6.1. *Extraction of global common variables*

1. Find sub-table $WPST_{\dot{x}i}$ from $\mathcal{T}$ and check whether all the elements of $WPST_{\dot{x}i}$ are "1" for all variables, where $0 \leq i \leq n-1$. Mark the corresponding $WPST_{\dot{x}i}$.

2. Store $\dot{x}i$ as the global common variables.

3. Find sub-table $WPST_{\dot{x}i}$ from $\mathcal{T}$ and check whether all the elements of $WPST_{\dot{x}i}$ are "0" for all variables, where $0 \leq i \leq n-1$. Mark the corresponding $WPST_{\dot{x}i}$.

4. Repeat Steps 1 to 3 for other variables. Count the number of $WPST_{\dot{x}i}$ with all "1"s and with all "0"s respectively. The numbers are notated as G and H for $WPST_{\dot{x}i}$ with all "1"s and with all "0"s respectively.

5. Generate sub-table $\mathcal{T}'$ by deletion of $G+H$ number of $WPST_{\dot{x}i}$ from $\mathcal{T}$, where $\mathcal{T}'$ has R rows and $n-G-H$ columns.

It is however not straightforward to extract local common variables. Following is the procedure of extraction of local common variables, assuming R rows and n columns in $\mathcal{T}$ for n variables and R π-terms in the FPRM expansion.

Procedure 6.2. *Extraction of local common variables*

1. Generate sub-table $ST_{\dot{x}i}$ by deletion of $WPST_{\dot{x}i}$ from $\mathcal{T}$, where $ST_{\dot{x}i}$ has R rows and $n-1$ columns.

2. Categorise π-terms into different class by the number of "1"s in $ST_{\dot{x}i}$. The class that has the greatest number of "1"s is notated as $max(ST_{\dot{x}i})$.

3. Repeat Steps 1 to 2 to obtain n number of $max(ST_{\dot{x}i})$ for each $ST_{\dot{x}i}$.

4. Generate $\mathcal{T}'$ from $\mathcal{T}$ by deletion of E numbers of $WPST_{\dot{x}i}$ in which the $ST_{\dot{x}i}$ has the largest number $max(ST_{\dot{x}i})$, where $E=[n/2]$ and [] is an integer operator. Put the variables $\dot{x}_i$ for the corresponding deleted sub-table $WPST_{\dot{x}i}$ into $\{VAR\}$. As a result, the $\mathcal{T}'$ has R rows and $n-E$ columns.

5. Categorise π-terms into different classes by the number of "1"s in $\mathcal{T}'$. The class that has the greatest number of "1"s is notated as $max(\mathcal{T}')$. Put the π-terms which are in the class with the largest $max(\mathcal{T}')$ into $\{PI\}$.

6. Reorganise $\mathcal{T}$ as follows. Carry out row exchange first so that $\{PI\} \subset WVST_1$ and the rest of π-terms in $WVST_2$. Carry out column exchanges so that $VST_{11} = \{VAR\}$, where $WVST_1 = \{ST_{11}, ST_{12}\}$ and

	x_4	x_3	x_2	x_1	x_0
π_1	0	0	0	0	1
π_6	0	0	1	1	0
π_9	0	1	0	0	1
π_{21}	1	0	1	0	1
π_{23}	1	0	1	1	1
π_{26}	1	1	0	1	0
π_{31}	1	1	1	1	1

Figure 6.12: The $\mathcal{T}$ of a 5-variable FPRM expansion.

$ST_{11}, ST_{12} \subset WVST_1$. All of the elements in ST_{12} should be "1". VST_{12} is local common variables.

Example 6.7. Given a 5-variable FPRM expansion of positive polarity, as shown in (6.13). The onset coefficients are $\mathcal{O} = \{1, 6, 9, 21, 23, 26, 31\}$.

$$\begin{aligned}\mathcal{F}(x_4, x_3, x_2, x_1, x_0) &= x_0 \oplus x_2x_1 \oplus x_3x_0 \oplus x_4x_2x_0 \oplus x_4x_2x_1x_0 \\ &\quad \oplus x_4x_3x_1 \oplus x_4x_3x_2x_1x_0\end{aligned} \tag{6.13}$$

The corresponding $\mathcal{T}$ for the given FPRM expansion is shown in Figure 6.12. Since it is positive polarity expansion, all the variables appear in the true forms, i.e., $\dot{x}_i = x_i$. Because the original on-set table is carried out deletion until the variables are extracted from the original on-set table so as to simplify the FPRM expansion. For purpose of the simple representation, $\mathcal{T}$ is notated as an old on-set table before the deletion of on-set table. $\mathcal{T}'$ is notated as an newly generated one after the deletion of on-set table to distinguish itself from before old on-set table. The notation is the same as for π-terms. π' is

	x_4	x_2	x_1	x_0
$\pi_1^{'}$	0	0	0	1
$\pi_6^{'}$	0	1	1	0
$\pi_9^{'}$	0	0	0	1
$\pi_{21}^{'}$	1	1	0	1
$\pi_{23}^{'}$	1	1	1	1
$\pi_{26}^{'}$	1	0	1	0
$\pi_{31}^{'}$	1	1	1	1

Figure 6.13: Sub-table ST_{x3} after deleting $WPST_{x3}$ from $\mathcal{T}$.

notated as a new π-term to distinguish itself from old π-term after the deletion of table. Because of deletion of the table, $\pi^{'}$ sometimes might become partial π-term (part of a term) but $\pi^{'}$ is used even if it is partial π-term.

Step 1:

Generate sub-table ST_{xi} by deletion of $WPST_{xi}$ from $\mathcal{T}$. If $WPST_{x3}$ is deleted for the corresponding variable x_3, as shown in Figure 6.13, ST_{x3} is generated, where $ST_{x3} = \{WPST_{x4}, WPST_{x2}, WPST_{x1}, WPST_{x0}\}$.

Step 2:

According to the content in each row of ST_{x3}, ST_{x3} can be categorised into 5 classes: $\{\pi_1^{'}, \pi_9^{'}\}$, $\{\pi_{23}^{'}, \pi_{31}^{'}\}$, $\{\pi_6^{'}\}$, $\{\pi_{21}^{'}\}$ and $\{\pi_{26}^{'}\}$. The number of "1"s in each class is 2, 8, 2, 3 and 2 respectively. Hence $max(ST_{x3}) = 8$.

Step 3:

Similarly, $max(ST_{xi})$ for the rest of the variables can be obtained by repeating Steps 1 and 2 as $max(ST_{x0}) = 4$, $max(ST_{x1}) = 6$, $max(ST_{x2}) = 4$ and $max(ST_{x4}) = 4$.

	x_4	x_2	x_0
π_1'	0	0	1
π_6'	0	1	0
π_9'	0	0	1
π_{21}'	1	1	1
π_{23}'	1	1	1
π_{26}'	1	0	0
π_{31}'	1	1	1

Figure 6.14: Sub-table $\mathcal{T}'$ after deleting $WPST_{x1}$ and $WPST_{x3}$.

	x_4	x_2	x_0	x_3	x_1	
π_1'	0	0	1	0	0	
π_6'	0	1	0	0	1	$WVST_2$
π_9'	0	0	1	1	0	
π_{26}'	1	0	0	1	1	
π_{21}'	1	1	1	0	0	
		ST_{12}		ST_{11}		
π_{23}'	1	1	1	0	1	$WVST_1$
π_{31}'	1	1	1	1	1	

Figure 6.15: The resulting $\mathcal{T}'$ after columns and rows are swapped.

Step 4:

$E = [n/2] = 2$. $\{VAR\} = \{x_1, x_3\}$ is recorded. $\mathcal{T}'$ is generated by deleting $WPST_{x1}$ and $WPST_{x3}$. Figure 6.14 shows the resulting $\mathcal{T}'$.

Step 5:

According to the content in each row of $\mathcal{T}'$, $\mathcal{T}'$ can be categorised into 4

classes, which are $\{\pi_1', \pi_9'\}$, $\{\pi_6'\}$, $\{\pi_{26}'\}$ and $\{\pi_{21}', \pi_{23}', \pi_{31}'\}$. $\{\pi_{21}', \pi_{23}', \pi_{31}'\}$ has the greatest number of "1"s. Hence, $\{PI\} = \{\pi_{21}', \pi_{23}', \pi_{31}'\}$.

Step 6:

Reorganise $\mathcal{T}$ by exchanging rows so that $\{\pi_{21}, \pi_{23}, \pi_{31}\} \subset WVST_1$ and the rest of the π-terms in $WVST_2$. As a result, $\mathcal{T}' = \{WVST_1, WVST_2\}$. Reorganise $\mathcal{T}'$ by exchanging columns so that $VST_{11} = \{x_1, x_3\}$, where $WVST_1 = \{ST_{11}, ST_{12}\}$, $ST_{11}, ST_{12} \subset WVST_1$ and all of the elements in ST_{12} should be "1". Hence, $\mathcal{T}' = \{WVST_1, WVST_2\} = \{\{ST_{11}, ST_{12}\}, WVST_2\}$ and $VST_{12} = \{x_4x_2x_0\}$, as shown in Figure 6.15.

If $WVST_2 \rightarrow \mathcal{F}_2$, $ST_{12} \rightarrow s\pi_{12}$ and $ST_{11} \rightarrow \mathcal{F}_{11}$, where $s\pi_{12}$ is local common variables and $s\pi_{12} = x_4x_2x_0$, then

$$\mathcal{F} = s\pi_{12}\mathcal{F}_{11} \oplus \mathcal{F}_2$$

6.4 On-set Table Method for Multi-Level Mixed Polarity RM

Any function of a given FPRM expansion can be represented as an onset table. Using the properties of onset table and the proposed extraction of common variables in Sections 6.2 and 6.3, the onset table can be divided into smaller sub-tables by extracting common variables, which leads to compact MMPRM expansion.

There are two stop criteria of the algorithm for determining the smallest onset truth table.

1. Only one π-term appears in the $\mathcal{T}$.

2. Only one variable appears in each π-term.

The MMPRM expansion is obtained in Procedure 6.3 as follows.

Procedure 6.3. *On-set table method for obtaining the MMPRM expansion.*

1. Obtain $\mathcal{T}$ from a given two-level FPRM expansion $\mathcal{F}$, where $\mathcal{F} \rightarrow \mathcal{T}$.
2. Use Procedure 6.1 to extract global common variables and generate $\mathcal{T}'$ from $\mathcal{T}$. Save the extracted global common variables and update $\mathcal{T}'$ as $\mathcal{T}$.
3. According to Corollary 6.7, extract and save common sub-table from $\mathcal{T}$ if it exists.
4. Use Procedure 6.2 to extract local common variables so that $\mathcal{T}$ becomes $\mathcal{T} = \{WVST_1, WVST_2\} = \{\{ST_{11}, ST_{12}\}, WVST_2\}$, where $WVST_1 \rightarrow \mathcal{F}_1$, $WVST_2 \rightarrow \mathcal{F}_2$, $ST_{12} \rightarrow s\pi_{12}$, $ST_{11} \rightarrow \mathcal{F}_{11}$, $s\pi_{12}$ is local common variables and $\mathcal{F} = s\pi_{12}\mathcal{F}_{11} \oplus \mathcal{F}_2$. Save the extracted local common variables and update ST_{11} as $\mathcal{T}$.
5. Determine if the stop criteria are satisfied. If satisfied, save the expansion. Otherwise repeat Steps 2 to 4 for $\mathcal{T}$.
6. Update $WVST_2$ as $\mathcal{T}$ and repeat Steps 2 to 5 for $\mathcal{T}$.
7. Output the MMPRM expansion $\mathcal{F} = \mathcal{F}_1 \oplus \mathcal{F}_2$.

If the expansion in Example 6.7 is carried out further using the proposed algorithm, the expressions of $\mathcal{F}_1$, $\mathcal{F}_2$ and the compact MMPRM expansion are obtained respectively as

$$\mathcal{F}_1 = x_4x_2x_0(\bar{x}_1 \oplus x_3x_1) \tag{6.14}$$

$$\mathcal{F}_2 = \bar{x}_3x_0 \oplus x_1(x_2 \oplus x_4x_3) \tag{6.15}$$

$$\mathcal{F} = x_4x_2x_0(\bar{x}_1 \oplus x_3x_1) \oplus \bar{x}_3x_0 \oplus x_1(x_2 \oplus x_4x_3) \tag{6.16}$$

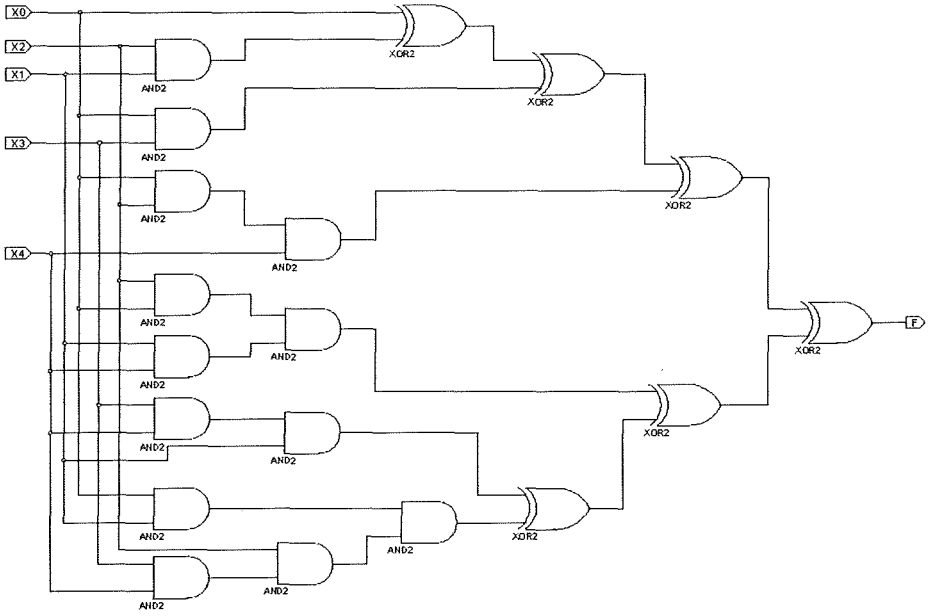


Figure 6.16: The circuit implemented with 2-input AND and XOR gates for PPRM expansion.

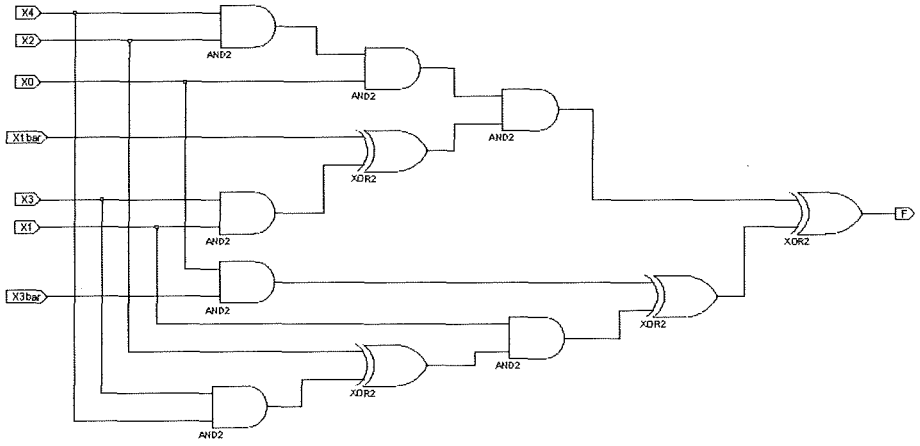


Figure 6.17: The circuit implemented with 2-input AND and XOR gates for MMPRM expansion.

It can be seen that in (6.13), the number of literals is 20. This is reduced to 12 literals in (6.16) representing 40% saving. If 2-input AND and 2-input XOR gates are used to implement expansions, the circuits for PPRM and MMPRM expansions can be obtained as shown in Figure 6.16 and Figure 6.17

respectively. It can be seen that in Figure 6.16, the number of gates in the circuit is 19. This is reduced to 11 gates in Figure 6.16 resulting in 42.1% saving.

6.5 Experimental Results

The proposed algorithm is implemented in C. The results are obtained using a PC with Pentium IV (1.8 GHz). The proposed algorithm is applied to Microelectronics Center of North Carolina (MCNC) benchmark circuits [142]. Performance is measured on 7 MCNC benchmark circuits and three randomly generated circuits. The test circuit size is up to 25 input variables. The number of literals is used to measure the area of the circuit implementation.

Table 6.1 shows the comparison of the number of literals between positive polarity expansion of FPRM expansions and MMPRM expansion obtained from the positive polarity expansion of FPRM expansions. The first column in Table 6.1 is the benchmark circuit name, the second is the input variable number, the third and the fourth list the number of literals, and the last column is the percentage improvement. The improvement is defined as in (6.17).

$$imp = \frac{literals - literals\ in\ MMPRM}{literals} \times 100\% \qquad (6.17)$$

where "literals" stands for the results from [121] while the "literals in MMPRM" stands for the number of literals obtained by the proposed method. It can be seen that the maximum improvement could be up to 80% and the average improvement is 68%.

Table 6.2 shows the comparison of the number of literals between the best polarity expansion of FPRM and MMPRM expansions. The MMPRM is de-

Table 6.1: Comparison of the number of literals between a FPRM expansion under polarity 0 and a MMPRM expansion.

Benchmarks	n	No. of literals under polarity 0 [121]	No. of literals using the proposed method*	*imp* (%)
9sym	9	756	304	60
newill	8	237	70	70
newtag	8	88	27	69
life	9	792	321	59
ryy6	16	624	168	73
sym10	10	1300	528	59
t481	16	108	55	49
test_21	21	135273	27304	80
test_22	22	153654	30262	80
test_25	25	90209	20399	77
average	-	-	-	68

*The MMPRM expansion is obtained from positive polarity.

Table 6.2: Comparison of the number of literals between a FPRM expansion under best polarity and a MMPRM expansion.

Benchmarks	n	No. of literals under best polarity [121]	No. of literals using the proposed method*	*imp* (%)
9sym	9	636	276	57
newill	8	78	24	69
newtag	8	27	15	44
life	9	596	218	63
ryy6	16	464	171	63
sym10	10	1300	528	59
t481	16	40	28	30
test_21	21	135273	27304	80
test_22	22	153654	30262	80
test_25	25	90209	20399	77
average	-	-	-	62

*The MMPRM expansion is obtained from the best polarity expansion of FPRM expansions.

rived from the best FPRM expansion. In this case, the maximum and average improvements are 69% and 62%, respectively.

In terms of the space complexity, the proposed algorithm only needs to store $\mathcal{T}$. Therefore, the space complexity is $O(coefficients*n)$, where $coefficients$ is the number of the onset coefficients and n is the number of input variables for a given function. For the time complexity, the CPU time used to solve a function with 25 input variables is 236 seconds using stated PC. This should not be a major problem for the computer nowadays.

6.6 Summary

In this chapter, a novel onset table based method is proposed to obtain a compact MMPRM form from an FPRM form. This method takes much less memory than previous method [131]. With the efficient extraction of common variables, the onset table is divided into smaller sub-tables. Using the mapping relationship between the $\mathcal{T}$ and the FPRM expansion, the compact MMPRM form can be obtained. The experimental results show a great improvement of literals count can be achieved compared to the published results [121].

The main advantage of the method is that it can search in the 2^n FPRM expansions, i.e. small space, rather than large space but have good result in the MPRM expansion. As a result, significant CPU time is reduced. Although it does not guarantee the best solution in the MPRM expansion, it does produce a very good solution.

Chapter 7

Genetic Algorithms for FPGA Placement

7.1 Introduction

Circuits based on AND/XOR operations have great advantage of easy testability [39]. Applications of Reed-Muller logic to function classification [112], Boolean matching [113], and symmetry detection [114] have also been attempted. However, XOR gate has the disadvantage of low speed and large area consumption compared to AND/OR. As the FPGA technology has made significant progress in recent years, XOR/XNOR gates can be implemented into LUTs, which changes the situation. As a result, XOR/XNOR gates can achieve comparable speed and area as other gates. Generic symmetrical FPGA architecture consists of routing resources and configurable blocks [97], in which routing resources occupy 70-90% of the FPGA area [23], therefore efficient P&R are essential. FPGA placement is categorised to be NP-complete. The aim of the P&R tool is to utilise prefabricated programmable routing switches

and routing channels in an FPGA to achieve 100% successful P&R.

In this chapter, symmetrical (Xilinx-style) FPGA placement algorithms are proposed, which incorporate the idea of using GA to find solutions for FPGA placement. The rest of the chapter is organised as follows. In Section7.2 GA for FPGA placement is given including the representation, crossover and mutation operators. Section 7.3 proposes a FPGA placement algorithm which unifies GA with SA to reduce the CPU time. Experimental results are given in Section 7.4.

7.2 Genetic Algorithm Placement

Hybrid GA (HGA) is a standard GA (SGA) which performs local optimisation in every generation to overcome long computation time and improve fitness of SGA. The HGA is shown in Algorithm 7.1.

The algorithm begins with an initial set of random population. After evaluating fitness of current population, the population is reproduced according to fitness. The fitter the individual, the more chance it has to be selected. Two individuals are randomly selected as parents to generate offsprings by using crossover operator based on high probability of crossover. Mutation operator with low probability rate is carried out. After that, local improvement with low probability rate is applied to randomly selected individuals so that visible improvement can be achieved, resulting in shorter search time. The elitism is employed to retain the good solutions. After a fixed number of generations, the fittest individual, namely the one with highest fitness value, is obtained as the desired solution.

Algorithm 7.1 Hybrid genetic algorithm for FPGA placement.

```
MAX_GENS: maximum number of generations
POP_SIZE: population size
NUM_GENE: number of genes
NUM_BLOCK: number of blocks for each benchmark
NUM_MOVE: number of moves per individual in local improvement
Pcrossover: probability of crossover rate
Pmutation: probability of mutation rate
Plocal: probability of local improvement rate
begin
  generate an initial population
  for generation = 1 to MAX_GENS do
    evaluate population fitness values;
    reproduce population probabilistically based on the individual's fitness
value;
    for i = 1 to POP_SIZE/2 do
      pair two parents randomly;
      crossover based on Pcrossover;
      produce two new offspring;
    end for
    for j = 1 to NUM_GENE do
      mutate offsprings based on Pmutation;
    end for
    for k = 1 to POP_SIZE do
      local improvement based on Plocal and NUM_MOVE;
    end for
    elitism;
  end for
end algorithm
```

7.2.1 Genetic encoding

The genotype of a problem is the representation of an individual in the GA. For our placement problem, the chromosome structure is $(L_1, L_2, L_3, \cdots, L_N)$, where N depends on K and $N = K * K$. K is the size of a symmetrical FPGA. For example, If the size of an FPGA $K = 4$, then $N = 16$. Each L_r can be either positive integer number or "-1", where $0 \leq r \leq N - 1$. The positive number can be in the range of $[0, BLKS]$, where $BLKS$ is the number of

CLBs [134, 135] of the circuit. "-1" represents empty block. The CLB has a corresponding position in the chromosome. The position POS is calculated as

$$POS = (x - 1) * K + (y - 1) \tag{7.1}$$

where POS is the position of gene, (x, y) is the location of the block.

In the above GA procedure, a chromosome pertaining to a possible placement solution is represented as a string with length which equals to the numbers of logic blocks in the symmetrical FPGA. For example, if the size of FPGA is 4 by 4, the numbers of the blocks are 16 and the length of chromosome is 16. The values of string can be either positive integer or "-1". The positive integer represents ID of the block and the value "-1" represents empty block. The position of block in a symmetrical FPGA is numbered according to its X-Y position. The block ID is mapped to chromosome according to its number, e.g. Block 10 at $(4, 4)$ in the FPGA is mapped to position 15 of the chromosome as shown in Figure 7.1.

7.2.2 Selection operator

The GA procedure carries out the genetic selection operator in which individual strings are chosen according to their fitness values. A proportional selection scheme as suggested by Goldberg is employed to select fitter parents which are required for reproduction. There are a number of ways to implement the selection operator. The easiest way is to create a biased roulette wheel where each current chromosome in the population has a roulette wheel slot sized in proportion to its fitness as in [7]. An individual is selected by spinning the roulette wheel and noting the position of the marker. However, the absolute

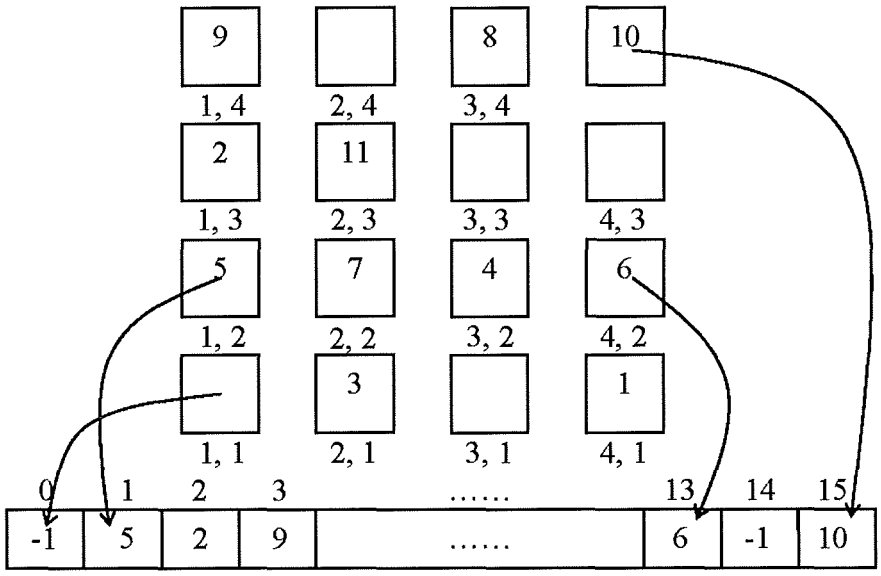


Figure 7.1: Genetic encoding.

difference between an individual's actual sampling probability and its expected value is nonzero, resulting in inefficient reproduction. Hence stochastic universal selection with zero bias [15] is employed in the reproduction process.

7.2.3 Fitness measure

A fitness function is used to evaluate the quality of placement. Its functional form is the sum of all nets in the circuit, as shown in (7.2).

$$P = \sum_{inet=1}^{NET} \left\{ \frac{100}{C(inet)\,[bb_x(inet) + bb_y(inet)]} \right\} \tag{7.2}$$

where NET stands for number of nets. For each net i, $bb_x(i)$ and $bb_y(i)$ denote the horizontal and vertical spans of its bounding box respectively. $C(inet)$, which is adapted from [19], compensates for the fact that the bounding box wire length model underestimates the wiring necessary to connect nets with more than three terminals. Its value depends on the number of terminals of

the net *inet*. It should be noted that high fitness value indicates a placement with shorter wire length, hence a better solution.

7.2.4 Crossover and mutation operators

Crossover is the main genetic operator. It operates on two individuals and generates two offsprings. It is an inheritance mechanism where the offspring inherits some of the characteristics of the parents. The operation consists of choosing a random cut point and generating the offsprings from two parents. Unfortunately, the elements to the left of crossover cut point in one parent appear on the right of the second parent, which results in element duplication in one offspring. This duplication does not represent a feasible placement solution. Modification of crossover to avoid duplication has to be carried out.

The modification is implemented as follows. Choose a random cut point and copy the entire segments following the cut point in parent 2 to the offspring. Next, the left segment of parent 1 is scanned from the left most, gene by gene, to the cut point. If a gene does not appear in the offspring then it is copied to the offspring. However, if it already exists in the offspring, then its position in parent 2 is determined and a gene from parent 1 in the determined position is copied. If the determined gene still exists in the offspring, determine the position in parent 2 as before until the gene does not exist in the offspring. One particular case is the gene "-1" which means the block is empty. If there is no empty block in parent 2, the empty gene is copied to the offspring in the same way as a gene does not appear in the offspring. However, if empty blocks do exist, it will randomly select any one of the empty positions in parent 2 and the gene from parent 1 in the selected position is then copied to the offspring. The selected empty position is marked to avoid being used again.

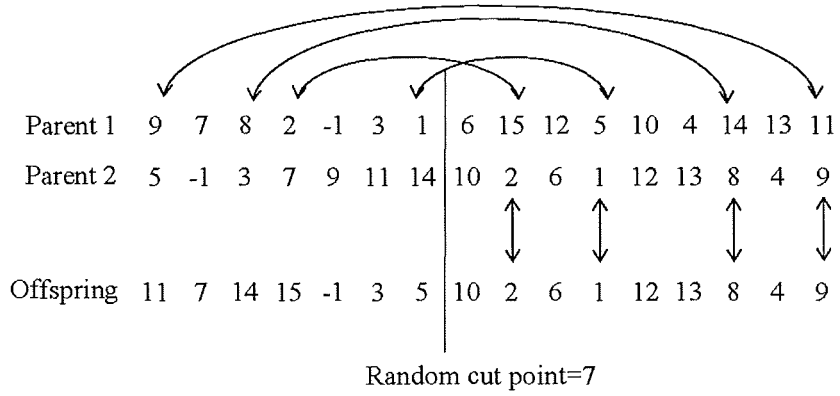


Figure 7.2: Modified crossover.

An example is shown in Figure 7.2. One possible placement, is encoded as a string of 16 bits, namely, {9, 7, 8, 2, -1, 3, 1, 6, 15, 12, 5, 10, 4, 14, 13, 11}. The other parent is {5, -1, 3, 7, 9, 11, 14, 10, 2, 6, 1, 12, 13, 8, 4, 9}. The random cut point is 7. The segment following the cut point in parent 2 is copied to offspring. Then the left segment of parent 1 is scanned from the left most. Because gene 9 is already in the offspring, position of gene 9 is determined in parent 2 so that gene 11 is found in parent 1. Gene 11 is not in the offspring so that it is copied to the offspring with the same position as in parent 1. Next gene is 7 which is not in the offspring so that it is copied to the offspring in the corresponding position. Again, gene 8 is already in the offspring. Gene 14 is copied to the offspring in the same way as was done for gene 11. The other genes in the left segment of parent 1 is processed in the same way. As a result, one offspring is generated as {11, 7, 14, 15, -1, 3, 5, 10, 2, 6, 1, 12, 13, 8, 4, 9}.

Mutation produces incremental random changes in the offspring generated by the crossover to overcome early converge to a local optimum. In the placement, the mutation is pair-wise interchange, namely, two genes of the chro-

mosome are randomly selected according to probability of mutation rate and their positions swapped.

7.2.5 Local improvement

After reproduction, crossover and mutation are performed, a local improvement is applied to the selected offspring in the current population based on the probability of local improvement rate. The local improvement is performed in every generation and keeps switching the position of the blocks in the symmetrical FPGA a number of times for randomly selected individual in order to improve the fitness of this particular individual. Some good schema of this individual is more likely to be selected and passed to next generation. The main goal of this process is to get some visible improvement in the offspring rather than obtaining a local optimum value. So the improvement rate is kept as low as possible and movement per individual is also kept to small value. As a result, the time for convergence is reduced significantly.

7.3 GA with SA Placement Algorithm

Although HGA performs better than SGA, excessive CPU time is consumed during the late process of the GA. To reduce the CPU time consumed in the late process of the GA, unified GA and SA algorithm that has two stages is proposed. The first stage is called global search stage, which is performed by GA. The second stage is called local search stage, which is performed by SA. Our algorithm starts with GA and works on a population of individuals by using reproduction, crossover, mutation and elitism operators with local improvement in order to obtain good solution. After a number of generations, the

solution is improved and has already jumped out of the local optimum. Then SA will take over to "fine-tune" the solution by swapping two nearby blocks at the low temperature. The pseudo code of unified GA with SA (GASA) is shown in Algorithm 7.2.

7.3.1 Fitness function

The quality of placement is evaluated by fitness function. The higher the fitness value, the better the placement. Since a benchmark circuit will have hundreds of nets or even more, the measure is judged by average fitness of all nets not just by partial ones, as follows

$$P = maxcost - \sum_{inet=1}^{NET} C(inet)\left[bb_x(inet) + bb_y(inet)\right] \tag{7.3}$$

$$maxcost = NET * K^2 \tag{7.4}$$

where $maxcost$ stands for the worst case of cost for placement, NET stands for the total numbers of nets, and K stands for the size of symmetrical FPGA. For each net $inet$, $bb_x(inet)$ and $bb_y(inet)$ denote the horizontal and vertical spans of its bounding box respectively. $C(inet)$ compensates for the fact that the bounding box wire length model underestimates the wiring necessary to connect nets with more than three terminals. Its value depends on the number of terminals of the net $inet$. The value of $C(inet)$ is 1 for nets with 3 or fewer terminals and slowly increases to 2.79 for nets with 50 or more terminals [33].

Algorithm 7.2 Genetic algorithm with simulated annealing for symmetrical FPGA placement.

```
MAX_GENS: maximum number of generations
POP_SIZE: population size
NUM_GENE: number of genes
Pcrossover: probability of crossover rate
Pmutation: probability of mutation rate
Plocal: probability of local improvement rate
Preserve: the percent of population are reserved in the generation
RANDOM: random number between 0 and 1
Pnew: new placement
Pold: old placement
T: temperature
e: constant of 2.732
begin
  initialise_population ();
  while (generation < MAX_GENS) do
    evaluate_population_fitness ();
    reproduce_population (Preserve);
    for index = 1 to POP_SIZE/2 do
      crossover (Pcrossover);
    end for
    for index = 1 to NUM_GENES do
      mutate (Pmutation);
    end for
    for index = 1 to POP_SIZE do
      local_improvement (Plocal);
    end for
    elitism();
  end while
  select_the_best_one ();
  T = set_temperature ();
  while (exit_criteria () == FALSE) do
    while (inner_criteria () == FALSE) do
      P_new = generate_movement ();
      ΔC = C(P_new) − C(P_old);
      RANDOM = generate_number ();
      if (RANDOM < e^(−ΔC/T))
        P_new = P_old;
    end while
  end while
end algorithm
```

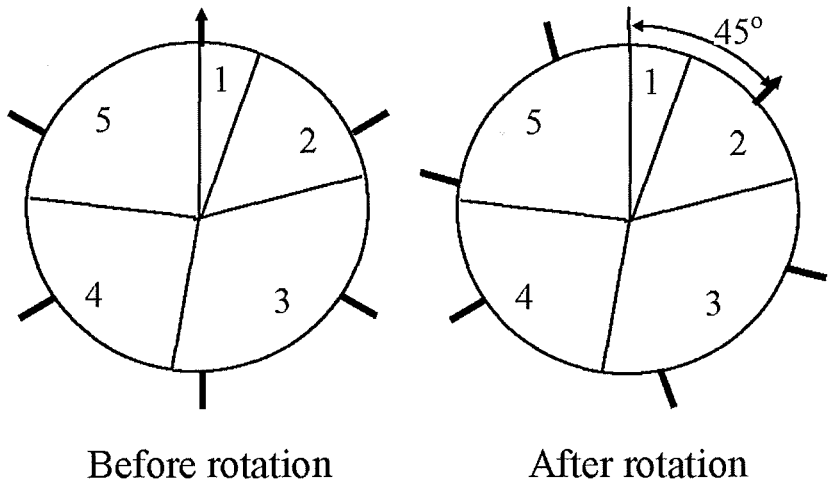


Figure 7.3: Selection pie and the rotation of markers.

7.3.2 Reproduction operator

In the algorithm, the population is initialised and evaluated according to fitness function. The fitness values of population are sorted in increasing order according to the fitness of individual. A small number of individuals of population with higher fitness value in the current generation are intact and remain in the population to the next generation. The rest works as follows.

The fitness of each individual is considered as a slot of a sized pie. Equidistant markers are placed around the pie, where ψ is the number of markers and equal to the number of individuals that do not remain in the population to the next generation. Figure 7.3 shows the sized pie and $\psi = 6$ equidistant markers around the pie. The sum of fitness value of individuals that do not remain in the population to the next generation corresponds to the whole size. A random number is generated. This number corresponds to the rotation of the markers. If the equidistant markers are inside the slot, the corresponding individuals are selected. As a result, individuals are simultaneously selected.

Although the selection procedure is random, the chance of being selected for each individual is directly proportional to its fitness.

For example, if there are 6 markers and the random number is 0.125, the 6 equidistant markers will rotate 45 degrees simultaneously. The selected individuals are therefore 2, 3, 3, 4, 5 and 5, as shown in Figure 7.3.

7.3.3 Initial temperature and update scheme

Once GA has done the global search in the first stage, SA will take over to do the fine tuning. The initial temperature T in the second local search stage is important to the overall performance of the algorithm. If the initial temperature is set too high, i.e., a large number of movements are allowed, the SA might not do the local search but tries to do the global search instead. As a result, it ruins the good global solution obtained by GA. If initial temperature is set too low, i.e., nearly no movements are allowed, SA can hardly do the local search. It is therefore that the initial temperature T is set at the low temperature.

As the process of SA continues, the temperature gradually drops, which limits the movements of blocks. New temperature is computed as

$$T_{new} = \beta T_{old} \tag{7.5}$$

where the value of β depends on the value of α. α is the percentage of trial movements that have been accepted. Table 7.1 shows the respective values of α and β. For example, if $T_{old} = 0.8$ and $\alpha = 0.1$, thus $\beta = 0.8$ and $T_{new} = 0.64$.

Table 7.1: Temperature update scheme.

α	β
$\alpha \geq 0.3$	0.6
$0.15 < \alpha < 0.3$	0.95
$0.03 \leq \alpha \leq 0.15$	0.8
$\alpha < 0.05$	0.6

7.4 Experimental Results

In this section, experimental results obtained with an implementation of SGA, HGA and GASA to symmetrical FPGA are reported. Each CLB has a 4-input LUT, which is used to implemented combinational logic only, one D flip-flop and one multiplexer, which is used to select combinational or sequential logic. There are horizontal channel and vertical channel. Each routing channel is assumed to have a fixed number of channel tracks. At every intersection of a horizontal channel and vertical channel, there is a programmable switch. It configures the wire segments between CLBs and CLBs and IOBs. Each wire segment spans the distance of one CLB. Two IOBs fit in the space of each CLB along the periphery of the symmetrical FPGA. The implementations of SGA and HGA are written in the C programming language. The proposed algorithm was applied to MCNC benchmark circuits [142]. Performance is measured using 9 MCNC benchmark circuits. Table 7.2 lists the main characteristics of these benchmark circuits.

The parameter values are selected following some experiments and based on previous experience. Pcrossover=0.6, Pmutation=0.005, POP_SIZE=50. Since SGA does not involve local improvement mutation operator, the parameter values are Plocal=0 and MAX_GENS=1000. The parameter values are set for HGA as Plocal=0.05, NUM_MOVE=10*NUM_BLOCK and

Table 7.2: Characteristics of MCNC benchmark circuits.

Benchmarks	No. of blocks	No. of nets	No. of CLBs	No. of I/Os
9symml	107	106	97	9/1
alu2	213	207	197	10/6
apex7	188	151	102	49/37
e64	404	339	274	65/65
example2	289	223	138	85/66
k2	609	564	519	45/45
term1	132	122	88	34/10
too-lrg	228	225	187	38/3
vda	374	308	291	17/39

MAX_GENS=200. POP_SIZE, MAX_GENS, Pcrossover, Pmutation and Plocal stand for population size, the maximum number of generations, probability of crossover rate, probability of mutation rate and probability of local improvement rate, respectively. NUM_MOVE stands for the number of block movements in the local improvement. For example, NUM_MOVE =1070 for circuit 9symml.

Fitness value of benchmark circuit 9symml in each generation obtained by SGA and HGA are illustrated in Figure 7.4 and Figure 7.5 respectively for demonstration. The best fitness value and average fitness value of each generation are shown in the two figures. In either of the two figures, the fitness value becomes fitter and fitter as the number of generations increases. Furthermore, observing the fitness value curve in Figure 7.4 and the fitness value curve in Figure 7.5, the fitness values from HGA are fitter than those obtained by SGA and HGA needs less number of generations than SGA does.

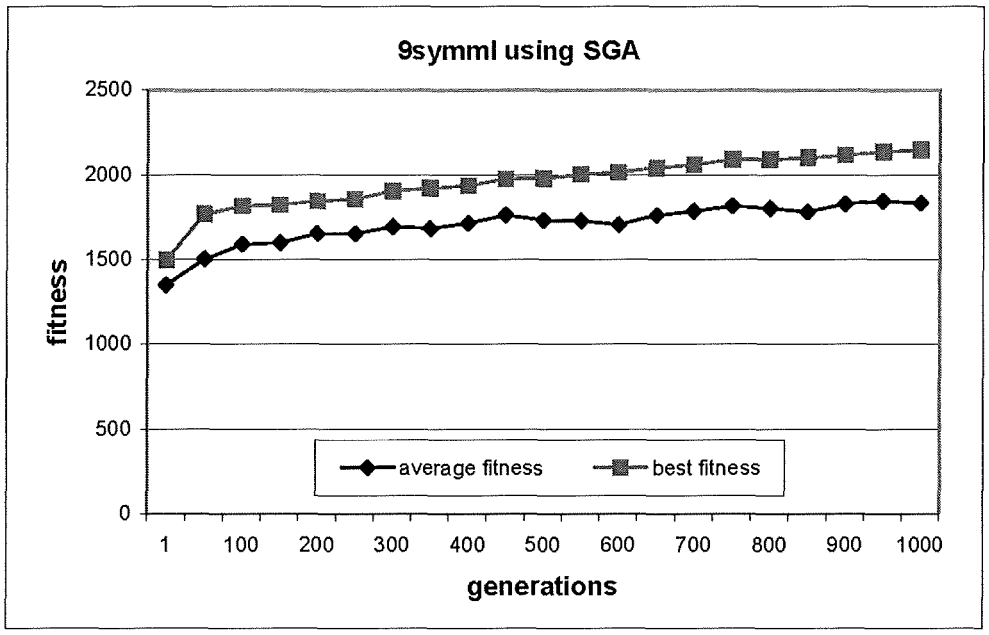


Figure 7.4: Fitness value of 9symml using SGA.

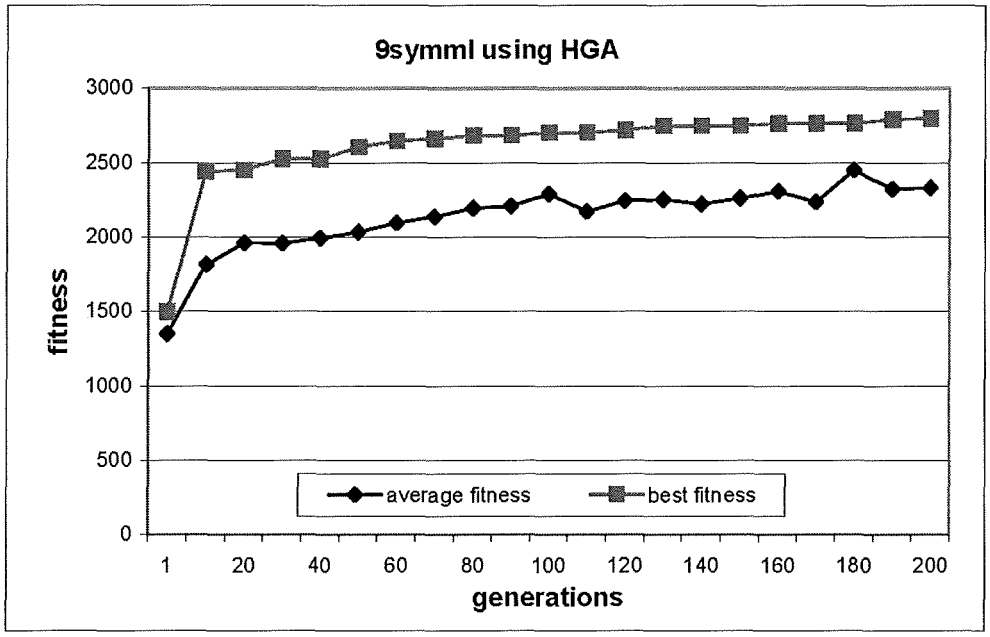


Figure 7.5: Fitness value of 9symml using HGA.

Table 7.3: Comparison of fitness.

Benchmarks	init. avg. fitness f	fitness of SGA f	imprvmnt. cmpr. to init. fitness (%)	fitness of HGA f	imprvmnt. cmpr. to SGA (%)
9symml	1349	2147	59	2799	30
alu2	1732	2921	69	4159	42
apex7	1495	3299	121	4364	32
e64	2246	3638	62	6391	76
example2	1528	3626	137	6746	86
k2	3062	4749	55	8261	74
term1	1323	2375	80	2944	24
too-lrg	1957	3239	66	4557	41
vda	2033	3258	60	5293	62
average	-	-	79	-	52

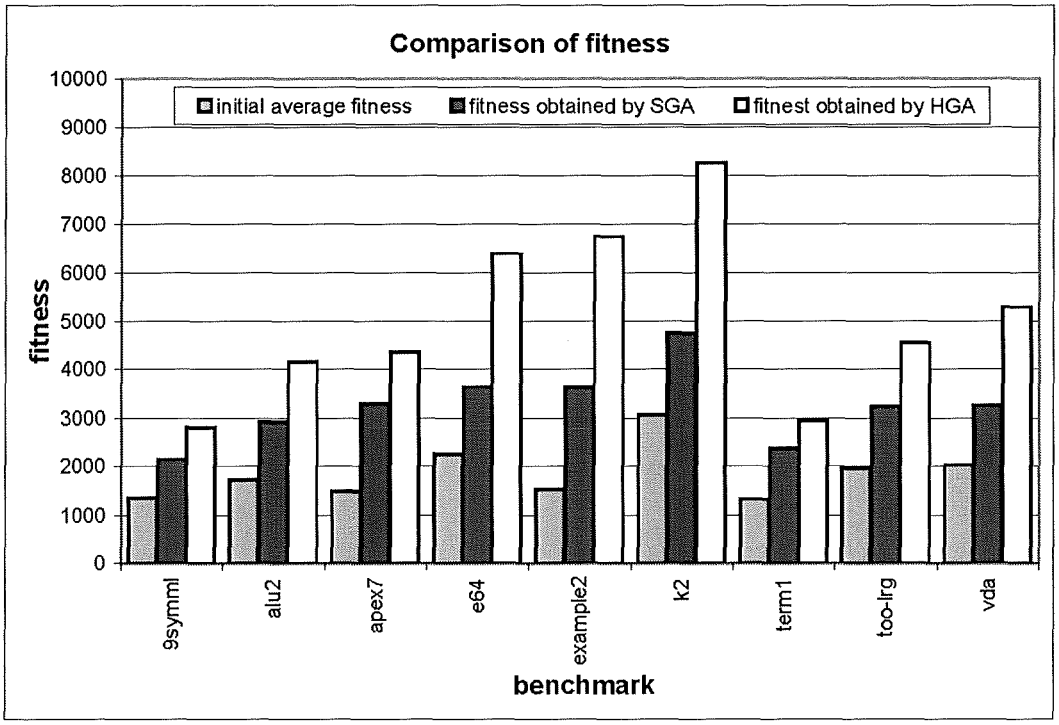


Figure 7.6: Comparison of fitness based on Table 7.3.

The full comparison results of 9 tested benchmark circuits are summarised in Table 7.3. As can be seen in Table 7.3, the fitness is improved on average by 79% in 1000 generations compared to initial random cost of placement. Further 52% improvement just in 200 generations is gained when HGA is employed compared to SGA. Figure 7.6 illustrates initial fitness before optimisation, fitness obtained by SGA and percentage improvement compared to initial fitness and fitness obtained by HGA and percentage improvement compared to SGA for 9 MCNC benchmark circuits respectively.

The placement from VPR Placer (VPlace) [18,19] and our SGA and HGA placements are routed by VPR Router (VRouter) [18,19]. The resulting channel tracks are compared. The main steps are illustrated by the flow chart in Figure 7.7. All parameter values for the routing are set equal for the purpose of fair comparison.

As expected, a poor placement will result in more tracks needed in the final routing for the same circuit. Therefore, we make practical assumption that the less the channel tracks needed for final routing, the better the placement is. As a result, the number of channel tracks is counted for comparison. Figure 7.8 shows the final routing of benchmark circuit 9symml, which needs 5 channel tracks.

The comparison of channel tracks needed by our algorithms with the state-of-the-art results from VPR [18,19] of 9 tested benchmark circuits are shown in Table 7.4.

The implementation of GASA algorithm was written in C programming language as well. The population size used by the GA is controversy. A smaller population size increases the efficiency of the GA and makes it competitive with other heuristic algorithms in terms of CPU time. On the other hand, a larger

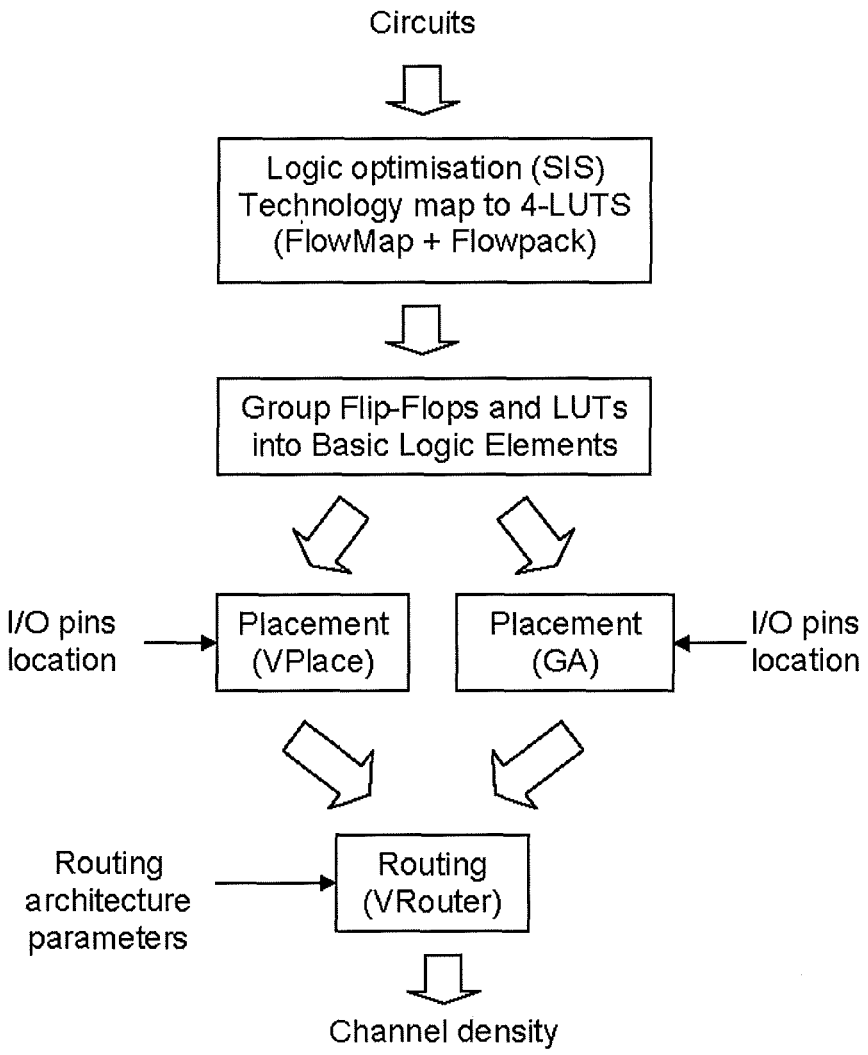


Figure 7.7: Comparison of CAD flow chart.

population size is able to reduce the errors associated with the selection of the parents for propagation. It is found that the diversity of the population is maintained when small population size of 20 is used. Due to small population size, only one individual of population with the highest fitness value in the current generation is intact and remains in the population to the next generation. In other word, 5 percent of population remains in the population to the next generation. The following fixed parameter values are selected and found suit-

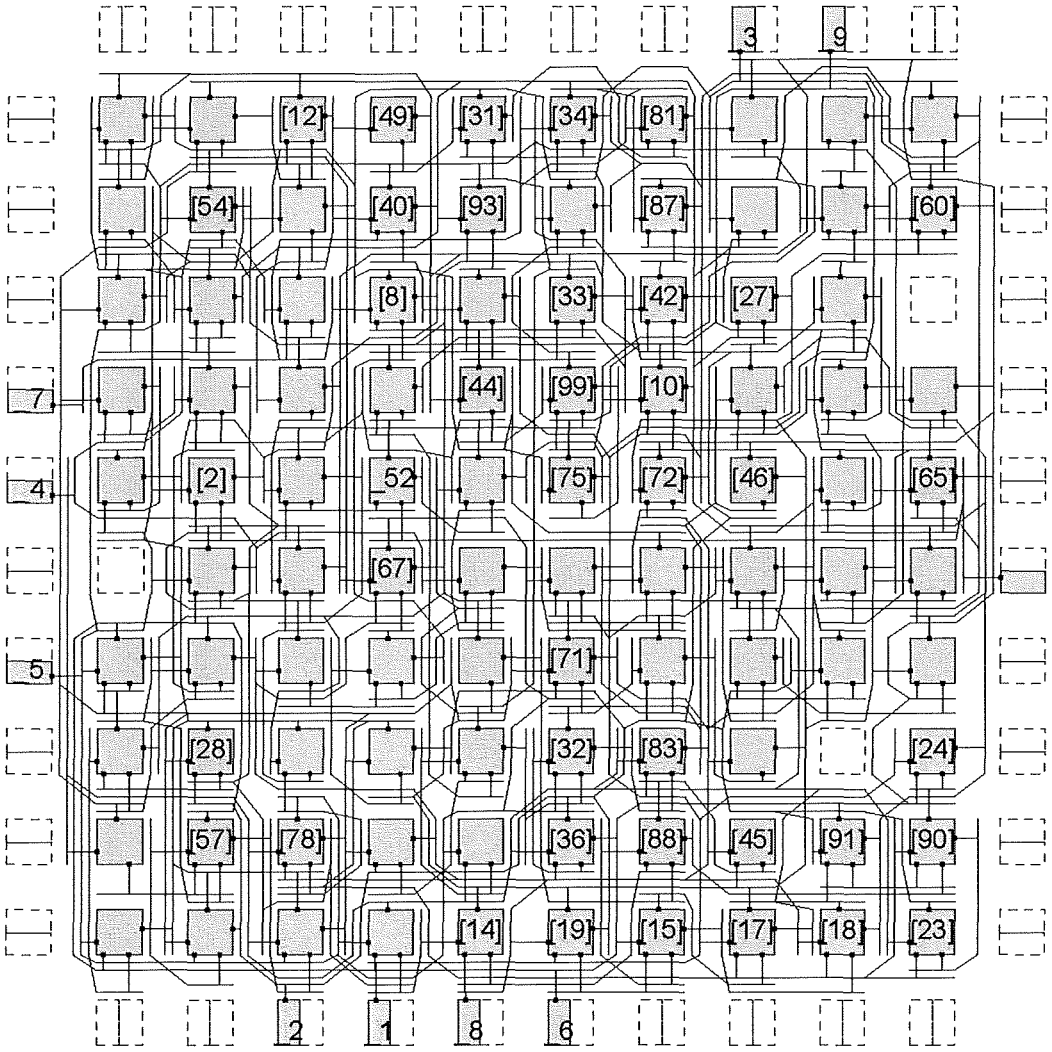


Figure 7.8: Final routing of 9symml using HGA placement with 5 channel tracks.

able for the tested benchmark circuits following some experiments and based on previous experience. POP_SIZE=20, Pcrossover=0.4, Pmutation=0.01, Plocal=0.3, Preserve=0.05 and MAX_GENS=50, where POP_SIZE, Pcrossover, Pmutation, Plocal, Preserve and MAX_GENS stand for population size, probability of crossover rate, probability of mutation rate, probability of local improvement rate, the percent of reserved population and the maximum number of generations, respectively. If the improvement does not gain in the GA for 5

Table 7.4: Comparison of channel tracks of VPR, SGA and HGA.

Benchmarks	FPGA size	VPlace [18,19]	SGA	HGA
9symml	$10 * 10$	5	6	5
alu2	$15 * 15$	6	10	7
apex7	$11 * 11$	5	8	5
e64	$17 * 17$	8	17	8
example2	$19 * 19$	5	8	5
k2	$23 * 23$	9	20	11
term1	$10 * 10$	5	7	5
too-lrg	$14 * 14$	7	12	8
vda	$18 * 18$	8	14	9
total	-	58	102	63

Table 7.5: Comparison results of CPU time and routing channel tracks between GA and GASA.

	GA		GASA	
Benchmarks	CPU (s)	No. of tracks	CPU (s)	No. of tracks
9symml	25.74	5	22.86	5
alu2	91.76	6	74.27	6
apex7	38.39	5	38.11	5
e64	163.70	8	155.21	8
example2	107.57	5	95.23	5
k2	461.59	10	364.77	9
term1	28.06	5	26.35	5
too-lrg	82.51	7	74.37	7
vda	179.17	8	148.33	8
total	1178.49	59	999.5	58

generations or the number of generations is greater than the maximum number of generations, SA will start to work on individual instead of entire population.

The proposed GASA is compared to GA in terms of CPU time. Table 7.5 shows that GASA consumes less CPU time than GA in all cases. To further compare the quality of placement between GA and GASA, same router is used to route placement solutions generated by GA and GASA. Placements are

Table 7.6: Comparison results of placement cost between VPlace and GASA.

	VPlace [18,19]		GASA	
Benchmarks	cost	No. of tracks	cost	No. of tracks
9symml	690	5	693	5
alu2	1670	6	1678	6
apex7	785	5	785	5
e64	2853	8	2849	8
example2	1348	5	1345	5
k2	5874	9	5873	9
term1	700	5	700	5
too-lrg	1750	7	1748	7
vda	3067	8	3067	8
total	18737	58	18738	58

routed on the smallest possible size of a symmetrical FPGA. The number of routing channel tracks is used to measure the performance. If a circuit can be placed and routed in an FPGA with fewer channel tracks, the area of an FPGA will be smaller and wire length and critical path of the circuit will be reduced. Thus the fewer channel tracks the better. For the purpose of fair comparison, parameters of routing tool from VRouter [18] are set to same values. The numbers of tracks obtained by GA and GASA are shown in Table 7.5.

The GASA is further compared to the state-of-art VPlace in terms of placement costs, as shown in Table 7.6. The cost is defined according to the cost function in [18,19]. As it can be seen in Table 7.6, GASA outperforms VPlace in 4 benchmark circuits, such as e64, example2, k2 and too-lrg.

7.5 Summary

In this chapter, SGA and HGA for symmetrical (Xilinx-style) FPGA placement are presented in the first part. The experiment results verify that the proposed SGA is effective for small size circuits. The fitness improved by 79% on average compared to fitness without optimization for 9 MCNC benchmark circuits, though a large number of generations is required. However, the proposed HGA can overcome this problem. It can obtain further improvement of 52% on fitness on average compared to the results obtained by SGA and significantly reduce the numbers of generations for convergence.

Our results are also compared to the state-of-the-art results from VPR. VPR needs 58 channel tracks to route tested benchmarks while SGA needs 102 channel tracks and HGA needs 63 channel tracks respectively.

In the second part, a two-step unified GASA method for symmetrical FPGA placement is presented. The experimental results show that the proposed GASA is effective in improving the quality of placement for the tested MCNC benchmark circuits. It consumes less CPU time than GA. Furthermore, the proposed placement algorithm could achieve as good performance as the state-of-the-art placement tool VPlace in terms of placement cost required for all benchmark circuits.

Chapter 8

Conclusions and Future Work

The objective of the research is to develop various CAD methods and algorithms for the synthesis and optimisation of logic function in the Reed-Muller forms, which are based on AND/XOR and OR/XNOR operations respectively. Additionally, FPGA placement is studied and algorithms are developed using GA and GA with SA. The main contributions can be summarised as follows.

1. In Chapter 3, an efficient way of generation of transformation matrix between CPOS and fixed polarity COC expansions is introduced. Based on the transformation matrix, two map techniques, map folding and transformation techniques, are presented for the generation of the fixed polarity COC expansion of any polarity. Map folding technique generates the expansion of any polarity by folding map of the positive polarity COC expansion. For some cases, the number of folding times becomes high. To overcome this, map transformation technique is proposed which generates the expansion of any polarity directly from the coefficients map. However, when the number of variables is greater than 4 drawing circles becomes inconvenient. In addition, it was observed that in most

cases the number of on-set CSOP minterms is much less than on-set CPOS maxterms. Hence, map folding techniques for conversion of fixed polarity COC expansion of any polarity based on on-set minterms of CSOP expansion is also proposed. Minterm method is further discussed in the Chapters 4 and 5.

2. Map techniques is straightforward but it can only be used for up to 6 variables. In Chapter 4, generalised on-set coefficients method is first proposed based on the bitwise implementation for large functions. Multi-segment algorithm is then proposed, which divides coefficients into several segments in order to achieve conversion between CPOS and fixed polarity COC expansions more efficiently. The proposed algorithms not only overcome the limitation of map methods in Chapter 3 and but also can be used for the bidirectional conversion between CPOS and fixed polarity COC expansions of any polarity. With the introduced concept of CPOS polarity, the fixed polarity COC expansions of any polarity is generated directly from on-set CPOS maxterms. It avoids generating positive expansion first before generating other polarity expansions like the procedure in map folding technique. Minterm method in Chapter 3 is generalised for large functions. Experimental results show that the proposed multi-segment algorithm is very efficient in terms of time and space for large functions, especially when minterm method is used. The average improvement is 30.64% for the 30 tested benchmarks compared to maxterm multi-segment method. The time and space complexity are $2^{1.5n}$ and 2^n respectively, where n is the number of input variables. The maxterm and minterm multi-segment methods took less than 0.22 seconds and 0.06 respectively for the tested benchmark circuits if the number

of input variables is less than 17. Both minterm and maxterm multi-segment methods outperform significantly published results [34, 51].

3. Chapter 5 proposes two conversion algorithms based on tabular technique, that is serial and parallel tabular techniques. Space complexity of the STT and PTT are both $O(2^n)$. STT deals with variable one at a time in sequence. PTT generates product terms at the same time. Although PTT produces products in parallel, if the number of product terms increases, the overhead of computation in each product term makes PTT under-performs compared to STT, in which STT adapts every efficient way of bitwise implementation. Time complexity of the STT and PTT are $O(n2^n)$ and $O(coefficients2^o)$ respectively, where $coefficients$ and n are the number of product terms and the number of input variables respectively and o is the number of "1"s in each on-set CPOS maxterm or CSOP minterm, depending on the method used. Experimental results show that both proposed STT and PTT achieve better performance than multi-segment method in Chapter 4 and other methods [34, 51].

4. Any n-variable function can be expressed in two-level FPRM forms. The on-set coefficients of FPRM forms can be represented in $\mathcal{T}$. In Chapter 6, on-set table method for obtaining MMPRM expansions is proposed. It deals with on-set coefficients only, resulting in significantly reduced memory. The space complexity is $O(coefficients * n)$, where $coefficients$ is the number of product terms and n is the number of input variables. By extracting common variables in the on-set table, the on-set table becomes smaller and smaller, the logical functionality of the circuit remains unchanged. As a result more compact expansion is obtained. Experimental

results show that the proposed method can achieve on average 68% and 55% area improvement compared to those under positive polarity and best polarity of FPRM expansions, respectively. By searching 2^n polarities of FPRM expansions, a good polarity out of the $2^{n2^{n-1}}$ polarities of MPRM can be obtained in reasonable time.

5. Symmetrical FPGA placement by using GAs is studied in Chapter 7. Genotype using integer representation is proposed for Symmetrical FPGA placement. With the proposed fitness function, selection, crossover and mutation operators, GA Placement algorithm is developed. However, it still spends long time in searching good solutions. To overcome the limitation of GA in large CPU consumption, Chapter 7 also proposes a unified GA and SA for Symmetrical FPGA placement based on two-step model. The unified GA and SA algorithm has two stages. The first stage is called global search stage, which is performed by GA. And the second stage is called local search stage, which is performed by SA. The algorithm achieves less CPU time than GA but without degrading the quality of the placement. Experimental results show that both GA and GA with SA placement could achieve as good performance as VPlace [18] in terms of final routing channel tracks.

The above works can be further carried out as follows.

1. The conversion methods in Chapters 4 and 5 for DFRM forms optimisation can be further generalised to incompletely specified Boolean functions.

2. Evolutionary computation algorithm, Particle Swarm Optimisation (PSO) [68], proved to be effective to many applications such as FPGA place-

ment [64] and Evolvable digital circuits [63]. It can be used for finding best fixed polarity and mixed polarity in RM and DFRM expansions.

3. The multi-segment methods, STT, PTT and on-set table based MMPRM optimisation method in Chapters 4 to 6 can be generalised for multiple output Boolean functions by using redundancy removal method as in [119].

4. The proposed GA and GA with SA pay penalty in CPU time for obtaining good results. PSO can however be applied to FPGA placement [64, 116] to reduce CPU time consumption. In addition, GA with SA placement, adapted two-step model, can be further carried out in other more complex models to achieve even better performance in terms of CPU time.

Publications

The following list shows the papers published or submitted during the research.

1. Yang, M., and Almaini, A.E.A., "Hybrid Genetic Algorithm for Xilinx-style FPGA Placement", *Proceedings of the 1st International Conference on ECAD/ECAE*, University of Durham, England, UK, pp.95-100, November 2004.

2. Yang, M., Almaini, A.E.A., Wang, L., and Wang, P.J., "An Evolutionary Approach for Symmetrical Field Programmable Gate Array Placement", *Proceedings of the 1st IEEE PhD Research in Microelectronics and Electronics* (*PRIME* '05), EPFL, Lausanne, Switzerland, pp.169-172, July 2005.

3. Yang, M., Almaini, A.E.A., Wang, L., and Wang, P.J., "FPGA Placement Using Genetic Algorithm with Simulated Annealing", *Proceedings of the 6th IEEE International Conference on ASIC* (*ASICON* '05), Shanghai, China, pp.808-811, October 2005.

4. Yang, M., Wang, P.J., Chen, X., and Almaini, A.E.A., "Fast Tabular Based Conversion Method for Canonical OR-Coincidence", *Proceedings of the 2005 IEEE International Conference on computer as a tool* (*EUROCON* '05), Belgrade, Serbia, pp.507-510, November 2005.

5. Yang, M., and Almaini, A.E.A., "FPGA Placement Optimization by Two-step Unified Genetic Algorithm and Simulated Annealing Algorithm", (accepted and in press) *Journal of Electronics (China)*.

6. Wang, P.J., Liu, Y., Yang, M., and Almaini, A.E.A, "Five Valued Circuit Quantitative Theory and Design of Five-valued Twisted Ring Counter", *Proceedings of the 6th IEEE International conference on ASIC (ASICON '05)*, Shanghai, China, pp.354-357, October 2005.

7. Yang, M., Almaini, A.E.A., and Wang, L., "Multi-segment Conversion Method between Large Boolean Functions and Canonical OR-Coincidence Expansions", *IEE Proceedings on Circuits, Devices and Systems* (submitted).

8. Yang, M., Almaini, A.E.A., and Wang, L., "Map and Tabular Techniques for Dual Forms of Reed-Muller Expansions Conversion", *Integration, the VLSI Journal* (submitted).

References

[1] Alexander, M.J., Cohoon, J.P., Ganley, J.L., and Robins, G., "Performance-oriented Placement and Routing for Field Programmable Gate Arrays", *Proceedings of the 1995 European Design Automation Conference* (*EURO-DAC* '95), Brighton, England, UK, pp.80-85, September 1995.

[2] Alexander, M.J., and Robins, G., "New Performance-driven FPGA Routing Algorithm", *IEEE Transactions on Computer Aided Design*, Vol.15, No.12, pp.1505-1517, December 1996.

[3] Ali, B., Kalganova, T., and Almaini, A.E.A., "Extrinsic Evolution of Finite State Machines", *Proceedings of the 5th International Conference on Adaptive Computing in Design and Manufacture* (*ACDM* '2002), University of Exeter, Devon, UK, pp.157-168, April 2002.

[4] Ali, B., Kalganova, T., and Almaini, A.E.A., "Evolutionary Algorithms and Their Use in the Design of Sequential Logic Circuits", *Genetic Programming and Evolvable Machines*, Vol.5, No.1, pp.11-29, 2004.

[5] Almaini, A.E.A., Thomson, P., and Hanson, D., "Tabular Techniques for Reed-Muller Logic", *International Journal of Electronics*, Vol.70, No.1, pp.23-34, 1991.

[6] Almaini, A.E.A., Electronic Logic Systems, 3rd ed., Prentice-Hall International, London, UK, 1994.

[7] Almaini, A.E.A., and Zhuang, N., and Bourset, F., "Minimisation of Multioutput Reed-Muller Binary Decision Diagrams Using Hybrid Genetic Algorithm", *IEE Electronics Letters*, Vol.31, No.20, pp.1722-1723, 1995.

[8] Almaini, A.E.A., Billina, S., Miller, J., and Thomson, P., "State Assignment of Finite State Machines Using a Genetic Algorithm", *IEE Proceedings on Computers and Digital Techniques*, Vol.142, No.4, pp.279-286, 1995.

[9] Almaini, A.E.A., and Zhuang, N., "Using Genetic Algorithms for the Variable Ordering of Reed-Muller Binary Decision Diagrams", *Microelectronics Journal*, Vol.24, No.4, pp.471-480, 1995.

[10] Almaini, A.E.A., and McKenzie, L., "Tabular Techniques for Generating Kronecker Expansions", *IEE Proceedings on Computers and Digital Techniques*, Vol.143, No.4, pp.205-212, 1996.

[11] Almaini, A.E.A., and Ping, S., "Algorithm for Reed-Muller Expansions of Boolean Functions and Optimization of Fixed Polarities", *Proceedings of the 4th IEEE International Conference on Electronics, Circuits and Systems* (*ICECS* '97), Cairo, Egypt, pp.148-153, December 1997.

[12] Almaini, A.E.A., and Zhuang, N., "Variable Ordering of BDDs for Multioutput Boolean Functions Using Evolutionary Techniques", *Proceedings of the 4th IEEE International Conference on Electronics, Circuits and Systems* (*ICECS* '97), Cairo, Egypt, pp.1239-1244, December 1997.

[13] Almaini, A.E.A., "A Semicustom IC for Generating Optimum Generalized Reed-Muller Expansions", *Microelectronics Journal*, Vol.28, No.2, pp.129-142, 1997.

[14] Ashar, P., Devadas, S., Newton, A.R., Sequential Logic Synthesis, Kluwer Academic Publishers, Boston, MA, USA, 1992.

[15] Baker, J.E., "Reducing Bias and Inefficiency in the Selection Algorithm", *Proceedings of the 2nd International Conference on GAs and Their Application*, Hillsdale, NJ, USA, pp.14-21, July 1987.

[16] Becker, B., and Drechsler, R., "Exact Minimisation of Kronecker Expressions for Symmetric Functions", *IEE Proceedings on Computers and Digital Techniques*, Vol.143, No.6, pp.349-354, 1996.

[17] Besslich, Ph.W., "Efficient Computer Method for ExOR Logic Design", *IEE Proceedings on Computers and Digital Techniques*, Vol.130, Pt. E, No.6, pp.203-206, 1983.

[18] Betz, V., and Rose, J., "VPR: A New Packing, Placement and Routing Tool for FPGA Research", *Proceedings of the 7th International Workshop on Field Programmable Logic and Applications* (*FPL* '97), London, UK, pp.213-222, September 1997.

[19] Betz, V., Rose, J., and Marquardt, A., Architecture and CAD for Deep-Submicron FPGAs, Kluwer Academic Publishers, Norwell, Massachusetts, USA, 1999.

[20] Brayton, R.K., and McMullen, C.T., "The Decomposition and Factorization of Boolean Expression", *Proceedings of the 1982 IEEE International*

Symposium on Circuit and Systems (*ISCAS* '82), Rome, Italy, pp.49-54, May 1982.

[21] Brayton, R.K., and Sangiovanni-Vincentelli, A.L., McMullen, C.T., and Hachtel, G.D., Logic Minimization Algorithms for VLSI Synthesis, Kluwer Academic Publishers, Boston, MA, USA, 1984.

[22] Brayton, R.K., and Rudell, R., Sangiovanni-Vincentelli, A.L., and Wang, A., "MIS: A Multiple-level Logic Optimization System", *IEEE Transactions on Computer Aided Design*, Vol.CAD-6, No.6, pp.1062-1081, 1987.

[23] Brown, S., Francis, R.J., Rose, J., and Vranesic, Z.G., Field Programmable Gate Arrays, Kluwer Academic Publishers, Norwell, MA, USA, 1992.

[24] Brown, S., and Rose, J. "A Detailed Router for Field Programmable Gate Arrays", *IEEE Transactions on Computer Aided Design*, Vol.11, No.5, pp.620-628, 1992.

[25] Bui, T., and Moon, B., "A Fast and Stable Hybrid Genetic Algorithm for the Ratio-cut Partitioning Problem on Hypergraphs", *Proceedings of the 31st ACM/IEEE Design Automation Conference* (*DAC* '94), San Diego, California, USA, pp.664-669, June 1994.

[26] Bystrov, A., and Almaini, A.E.A., "Testability and Test Compaction for Decision Diagram Circuits", IEE *Proceedings on Circuits Devices and Systems*, Vol.146, No.4, pp.153-158, 1999.

[27] Carter, W.S., Duong, K., Freeman, R.H., and Hsieh, H., Ja, J.Y., Mahoney, J.E., Ngo, L.T., Sze, S.L., "A User Programmable Reconfigurable

Logic Array", *Proceedings of the 1986 IEEE Custom Integrated Circuits Conference* (*CICC* '86), Rochester, NY, USA, pp.233-235, May 1986.

[28] Chang, Y.W., Thakur, S., Zhu, K., and Wong, D.F., "A New Global Routing Algorithm for FPGAs", *Proceedings of the 1994 IEEE/ACM International Conference on Computer Aided Design* (*ICCAD* '94), San Jose, California, USA, pp.356-361, November 1994.

[29] Chang, Y.W., Wong, D.F., and Wong, C.K., "FPGA Global Routing Based on A New Congestion Metric", *Proceedings of the 1995 IEEE International Conference on Computer Design: VLSI in Computers and Processors* (*ICCD* '95), Austin, Texas, USA, pp.372-378, October 1995.

[30] Chang, Y.W., and Chang, Y.T., "An Architecture-driven Metric for Simultaneous Placement and Global Routing for FPGAs", *Proceedings of the 37th ACM/IEEE Design Automation Conference* (*DAC* '2000), Los Angeles, California, USA, pp.567-572, June 2000.

[31] Chang, Y.W., Zhu, K., and Wong, D.F., "Timing-driven Routing for Symmetrical-Array-Based FPGAs", *ACM Transactions on Design Automation of Electronic Systems*, Vol.5, No.3, pp.433-450, July 2000.

[32] Chen, K.C., Cong, J., Ding, Y., Kahng, A.B., and Trajmar, P., "DAG-MAP: Graph-based FPGA Technology Mapping for Delay Optimization", *IEEE Design and Test of Computers*, Vol.9, No.3, pp.7-20, September 1992.

[33] Cheng, C. "RISA: Accurate and Efficient Placement Routability Modeling", *Proceedings of the 1994 IEEE/ACM International Conference*

on Computer Aided Design (*ICCAD* '94), San Jose, California, USA, pp.690-695, November 1994.

[34] Cheng, J., Chen, X., Faraj, K. M. and Almaini, A.E.A., "Expansion of Logical Function in the OR-coincidence System and the Transform between It and Maxterm Expansion", *IEE Proceedings on Computers and Digital Techniques*, Vol.150, No.6, pp.397-402, 2003.

[35] Cong, J., and Ding, Y., "An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs", *Proceedings of the 1992 IEEE/ACM International Conference on Computer Aided Design* (*ICCAD* '92), pp.48-53, November 1992.

[36] Cong, J., and Ding, Y., "FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs", *IEEE Transactions on Computer Aided Design*, Vol.3, No.1, pp.1-12, 1994.

[37] Cong, J., and Xu, S., "Delay-optimal Technology Mapping for FPGAs with Heterogeneous LUTs". *Proceedings of the 35th ACM/IEEE Design Automation Conference* (*DAC* '98), San Francisco, California, USA, pp.704-707, June 1998.

[38] Cong, J., Hwang, Y.Y., and Xu., S., "Technology Mapping for FPGAs with Non-uniform Pin Delays and Fast Interconnections". *Proceedings of the 36th ACM/IEEE Design Automation Conference* (*DAC* '99), New Orleans, Louisiana, pp.373-378, June 1999.

[39] Damarla, T., and Karpovsky, M., "Detection of Stuck-at and Bridging Faults in Reed-Muller Canonical (RMC) Networks", *IEE Proceedings on*

Computers and Digital Techniques, Pt.E., Vol.136, No.5, pp.430-433, 1989.

[40] Debnath, D., and Sasao, T., "GRMIN2: A Heuristic Simplification Algorithm for Generalised Reed-Muller Expressions", *IEE Proceedings on Computers and Digital Techniques*, Vol.143, No.6, 376-384, 1996.

[41] Drechsler, R., Becker, B., and Drechsler, N., "Genetic Algorithm for Minimisation of Fixed Polarity Reed-Muller Expressions", *IEE Proceedings on Computers and Digital Techniques*, Vol.147, No.5, pp.349-353, 2000.

[42] Drechsler, R., Theobald, M., and Becker, B., "Fast OFDD-based Minimisation of Fixed Polarity Reed-Muller Expressions", *IEEE Transactions on Computers*, Vol.45, No.11, pp.1294-1299, 1996.

[43] Drechsler, R., and Becker, B., "Relation between OFDDs and FPRMs", *IEE Electronics Letters*, Vol.32, No.21, pp.1975-1976, 1996.

[44] Du, P., Grewal, G., Areibi, S., and Banerji, D., "A Fast Hierarchical Approach to FPGA Placement", *Proceedings of the 2004 International Conference on Embedded Systems and Applications* (*ESA* '04), Las Vegas, Nevada, USA, pp.497-503, June 2004.

[45] Du, P., Grewal, G., Areibi, S., and Banerji, D., "A Fast Adaptive Heuristic for FPGA Placement", *Proceedings of the 2nd IEEE Northwest Workshop on Circuits and Systems*, Montreal, Canada, pp.373-376, June 2004.

[46] Esbensen, H., "A Genetic Algorithm for Macro Cell Placement", *Proceedings of the 1992 European Design Automation Conference* (*EURO-DAC* '92), Hamburg, Germany, pp.52-57, September 1992.

[47] Esbensen, H. and Mazumder, P., "SAGA: Unification Of Genetic Algorithm with Simulated Annealing and Its Application to Macro-Cell Placement", *Proceedings of the 7th IEEE International Conference on VLSI Design* (*VLSI Design* '94), Calcutta, India, pp. 211-214, January 1994.

[48] Esbensen, H., "A Macro-Cell Global Router Based on Two Genetic Algorithms", *Proceedings of the 1994 European Design Automation Conference* (*EURO-DAC* '94), Grenoble, France, pp.428-433, September 1994.

[49] Falkowski, B.J., and Chang, C.H., "An Exact Minimizer of Fixed Polarity Reed-Muller Expansions", *International Journal of Electronics*, Vol.79, No.4, pp.389-409, 1995.

[50] Falkowski, B.J., and Perkowski, M.A., "One More Way to Calculate Generalized Reed-Muller Expansions of Boolean Functions", *International Journal of Electronics*, Vol.71, No.3, pp.385-396, 1991.

[51] Faraj, K.M., Combinational Logic Synthesis Based on the Dual Form of Reed-Muller Representation, Ph.D. Thesis, Napier University, Edinburgh, UK, 2005.

[52] Francis, R.J., Rose, J., and Chung, K., "Chortle: A Technology Mapping Program for Lookup Table-Based FPGAs", *Proceedings of the 27th ACM/IEEE Design Automation Conference* (*DAC* '90), Orlando, Florida, USA, pp.613-619, June 1990.

[53] Francis, R.J., Rose, J., and Vranesic, Z., "Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs", *Proceedings of the 28th*

ACM/IEEE Design Automation Conference (*DAC* '91), San Francisco, California, USA, pp.227-233, June 1991.

[54] Francis, R.J., "A Tutorial on Logic Synthesis for Lookup-Table Based FPGAs", *Proceedings of the 1992 IEEE/ACM International Conference on Computer Aided Design* (*ICCAD* '92), Santa Clara, California, USA, pp.40-47, November 1992.

[55] Gockel, N., Pudelko, G.,Drechsler, R., and Becker, B., "A Hybrid Genetic Algorithm for the Channel Routing Problem", *Proceedings of the the 1996 IEEE International Symposium on Circuits and Systems "Connecting the World"* (*ISCAS* '96), Atlanta, GA, USA, pp.675-678, May 1996.

[56] Goldberg, D.E., Genetic Algorithms in Search, Optimization, and Machine Learning, Addison Wesley Longman Publishing, Boston, MA, USA, 1989.

[57] Goni, B.M., and Arslan, T., "An Evolutionary 3D Over-the-cell Router", *Proceedings of the 12th IEEE International ASIC/SOC Conference*, Washington, DC, USA, pp.206 -209, September 1999.

[58] Goni, B. M., Arslan, T., and Turton, B., "Power Driven Routing Using a Genetic Algorithm", *Proceedings of the Joint Conference of the 3rd World Multiconference on Systemics, Cybernetics and Informatics* (*SCI* '99) *and 5th International Conference on Information Systems Analysis and Synthesis* (*ISAS* '99), Orlando, Florida, pp.444-448, August, 1999.

[59] Goni, B.M., Arslan, T., and Turton, B., "A Genetic Algorithm for Over-the-cell and Channel Area Optimization", *Proceedings of the 2000*

Congress on Evolutionary Computation (*CEC* '00), La Jolla, CA, USA, Vol.1, pp.586-592, July 2000.

[60] Green, D.H., "Families of Reed-Muller Canonical Forms", *International Journal of Electronics*, Vol.70, No.2, pp.259-280, 1991.

[61] Green, D.H., "Dual Forms of Reed-Muller Expansions", *IEE Proceedings on Computers and Digital Techniques*, Vol.141, No.3, pp.184-192, 1994.

[62] Gregory, D., Bartlett, K., Geus, A., and Hachtel, Gary, "Socrates: A System for Automatically Synthesizing and Optimizing Combinational Logic", *Proceedings of the 23rd ACM/IEEE Design Automation Conference* (*DAC* '86), Las Vegas, Nevada, pp.79-85, June 1986.

[63] Gudise, V.G., and Venayagamoorthy, G.K., "Evolving Digital Circuits Using Particle Swarm", *Proceedings of the 2003 IEEE International Joint Conference on Neural Networks* (*IJCNN* '03), Portland, Oregon, USA, pp.468-472, July 2003.

[64] Gudise, V.G., and Venayagamoorthy, G.K., "FPGA Placement and Routing Using Particle Swarm Optimization", *Proceedings of the 2004 IEEE Annual Symposium on VLSI Emerging Trends in VLSI Systems Design* (*ISVLSI* '04), Lafayette, Louisiana, USA, pp.307-308, February 2004.

[65] Habib, M.K., "Boolean Matrix Representation for the Conversion of Minterms to Reed-Muller Coefficients and the Minimization of Exclusive-OR Switching Functions", *International Journal of Electronics*, Vol.68, No.4, pp.493-506, 1990.

[66] Harking, B., "Efficient Algorithm for Canonical Reed-Muller Expansions of Boolean Functions", *IEE Proceedings on Computers and Digital Techniques*, Vol.137, Pt.E. No.5, pp.366-370, 1991.

[67] Kaviani, A., and Brown, S., "Technology Mapping Issues for an FPGA with Lookup Tables and PLA-like Blocks", *Proceedings of the 2000 ACM/SIGDA International Symposium on FPGAs (FPGA '00)*, Monterey, California, USA, pp.60-66, February 2000.

[68] Kennedy, J., Eberhart, R.C., and Shi, Y., Swarm Intelligence, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[69] Khan, M.M.H.A., and Alam, M.S., "Mapping of On-set Fixed Polarity Reed-Muller Coefficients from Minterms and the Minimization of Fixed Polarity Reed-Muller Expressions", *International Journal of Electronics*, Vol.83, No.2, pp.235-247, 1997.

[70] Khan, M.M.H.A., and Alam, M.S., "Mapping of On-set Fixed polarity Reed-Muller Coefficients from On-set Canonical Sum of Products Coefficients and the Minimization of Pseudo Reed-Muller Expressions", *International Journal of Electronics*, Vol.86, No.3, pp.255-268, 1999.

[71] Koakutsu, S., Kang, M., and Dai, W.W.-M., "Genetic Simulated Annealing and Application to Non-slicing Floorplan Design", *Proceedings of the 5th ACM/SIGDA Physical Design Workshop (PDW '96)*, Reston Sheraton, Virginia, USA, pp.134-141, April 1996.

[72] Korupolu, M.R., Lee, K.K., and Wong, D.F., "Exact Tree-Based FPGA Technology Mapping for Logic Blocks with Independent LUTs", *Proceed-*

ings of the 35th ACM/IEEE Design Automation Conference (*DAC* '98), San Francisco, California, USA, pp.708-711, June 1998.

[73] Lemieux, G., and Brown, S.D., "A Detailed Router for Allocating Wire Segments in FPGAs", *Proceedings of the 4th ACM/SIGDA Physical Design Workshop* (*PDW* '93), Lake Arrowhead, California, USA, pp.215-226, April 1993.

[74] Lemieux, G., Brown, S.D., and Vranesic, D., "On Two-Step Routing for FPGA", *Proceedings of the 1997 ACM/SIGDA International Symposium on Physical Design* (*ISPD* '97), Napa Valley, California, USA, pp.60-66, April 1997.

[75] Lienig, J., and Thulasiraman, K., "A New Genetic Algorithm for the Channel Routing Problem", *Proceedings of the 7th International Conference on VLSI Design* (*VLSI Design* '94), Calcutta, India, pp.133-136, January 1994.

[76] Lienig, J., and Thulasiraman, K., "A Genetic Algorithm for Channel Routing in VLSI Circuits", *Evolutionary Computation*, Vol.1, No.4, pp.133-136, 1994.

[77] Lienig, J., "A Parallel Genetic Algorithm for Performance-driven VLSI Routing", *IEEE Transactions on Evolutionary Computation*, Vol.1, No.1, pp.29-39, 1994.

[78] Lu, G., and Areibi, S., "An Island Based GA Implementation for VLSI Standard Cell Placement", *Proceedings of the 2004 Genetic and Evolutionary Computation Conference* (*GECCO* '04), Seattle, Washington, USA, pp. 1138-1150, June 2004.

[79] Lui, P.K., and Muzio, J., "Boolean Matrix Transforms for the Parity Spectrum and the Minimization of Modulo-2 Canonical Expansions", *IEE Proceedings on Computers and Digital Techniques*, Vol.138, Pt.E. No.6, pp.411-417, 1991.

[80] Lui, P.K., and Muzio, J., "Boolean Matrix Transforms for the Minimization of Modulo-2 Canonical Expressions", *IEEE Transactions on Computers*, Vol.41, No.3, pp.342-347, 1992.

[81] Marquardt, A., Betz, V., and Rose, J., "Using Cluster-based Logic Blocks to Improve FPGA Speed and Density", *Proceedings of the 1999 ACM/SIGDA International Symposium on FPGAs (FPGA* '99), Monterey, California, USA, pp.37-46, February 1999.

[82] Marquardt, A., Betz, V., and Rose, J., "Speed and Area Tradeoffs in Cluster-Based FPGA Architectures", *IEEE Transactions on VLSI Systems*, Vol.8, No.1, pp.84-93, 2000.

[83] Marquardt, A. Betz, V. and Rose, J., "Timing-driven Placement for FPGAs", *Proceedings of the 2000 ACM/SIGDA International Symposium on FPGAs (FPGA* '00), Monterey, California, USA, pp.203-213, February 2000.

[84] Mazumder, P., and Rudnick, E. M., "Genetic Algorithms for VLSI Design, Layout and Test Automation", Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999.

[85] McCulloch, S., Auction-based Routing for Field-Programmable Gate Arrays, Ph.D. Thesis, University of Virginia, USA, May 2002.

[86] McKenzie, L., Almaini, A.E.A., Miller, J.F., and Thomson, P., "Optimization of Reed-Muller Logic Functions", *International Journal of Electronics*, Vol.75, No.3, pp.451-466, 1993.

[87] McKenzie, L., and Almaini, A.E.A., "Generating Kronecker Expansions from Reduced Boolean Forms Using Tabular Methods", *International Journal of Electronics*, Vol.82, No.4, pp. 313-325, 1997.

[88] Miller, J.F., Luchian, H., Bradbeer, P.V.G., and Barclay, P.J., "Using a Genetic Algorithm for Optimizing Fixed Polarity Reed-Muller Expansions of Boolean Functions", *International Journal of Electronics*, Vol.76, No.4, pp.601-609, 1994.

[89] Miller, J.F., and Thomson, P., "Highly Efficient Exhaustive Search Algorithm for Optimizing Canonical Reed-Muller Expansions of Boolean Functions", *International Journal of Electronics*, Vol.76, No.1, pp.37-56, 1994.

[90] Muller, D.E., "Application of Boolean Algebra to Switching Circuits Design and to Error Detection", *IRE Transactions on Electronic Computers*, Vol.EC-3, 6-12, 1954.

[91] Rahmani, A.T., and Ono, N., "A Genetic Algorithm for Channel Routing Problem", *Proceedings of the 5th International Conference on GAs and their Application*, Urbana-Champaign, IL, USA, pp.494-498, June 1993.

[92] Prahlada Rao, B.B., Patnaik, L.M., and Hansdah, R.C., "Parallel Genetic Algorithm for Channel Routing Problem", *Proceedings of the 3rd IEEE Great Lake Symposium on VLSI Design*, Kalamazoo, Michigan, USA, pp.69-70, March 1993.

[93] Prahlada Rao, B.B., Patnaik, L.M., and Hansdah, R.C., "A Genetic Algorithm for Channel Routing Using Inter-cluster Mutation", *Proceedings of the 1st IEEE Conference on World Congress on Computation Intelligence*, Orlando, Florida, USA, pp.97-103, June 1994.

[94] Prahlada Rao, B.B., Patnaik, L.M., and Hansdah, R.C., "An Extended Evolutionary Programming Algorithm for VLSI Channel Routing", *Proceedings of the 4th Annual Conference Evolutionary Programming*, San Diego, CA, USA, pp.521-544, March 1995.

[95] Purwar, S., "An Efficient Method of Computing Generalised Reed-Muller Expansions from Binary Decision Diagram", *IEEE Transactions on Computers*, Vol.40, No.11, pp.1298-1301, 1996.

[96] Reed, I.S., "A Class of Multiple-Error-Correcting Codes and the Decoding Scheme", *IRE Transactions Information Theory*, Vol.PGIT-4, pp.38-49, 1954.

[97] Rose, J., El Gamal, A., and Sangiovanni-Vincentelli, A.L., "Architecture of FPGAs", *Proceedings of IEEE*, Vol.81, No.7, pp.1013-1029, 1993.

[98] Roy, S., Belkhale, K., and Banerjee, P., "An α-approximate Algorithm for Delay-Constraint Technology Mapping", *Proceedings of the 36th ACM/IEEE Design Automation Conference (DAC '99)*, New Orleans, Louisiana, USA, pp.367-372, June 1999.

[99] Sarrafzadeh, M., and Wong, C.K., An Introduction to VLSI Physical Design, McGraw-Hill, New York, USA, 1996.

[100] Sasao, T., and Besslich, P., "On the Complexity of MOD-2 Sum PLA's", *IEEE Transactions on Computers*, Vol.39, No.2, 262-266, 1990.

[101] Sasao, T., "Logic Synthesis with EXOR Gates", Logic Synthesis and Optimization, Kluwer Academic Publishers, Norwell, MA, USA, pp.259-285, 1993.

[102] Sasao, T., "AND-EXOR Expressions and Their Optimization", Logic Synthesis and Optimization, Kluwer Academic Publishers, Norwell, MA, USA, pp.287-312, 1993.

[103] Schnecke, V., and Vornberger, O., "Hybrid Genetic Algorithms for Constrained Placement Problems", *IEEE Transactions on Evolutionary Computation*, Vol.1, No.4, pp.266-277, 1994.

[104] Sechen, C., and Sangiovanni-Vincentelli, A.L., "Timber Wolf 3.2: A New Standard Cell Placement and Routing Package", *Proceedings of the 23rd ACM/IEEE Design Automation Conference (DAC '86)*, Las Vegas, Nevada, USA, pp.432-439, June, 1986.

[105] Shahookar, K., and Mazumder, P., "A Genetic Approach to Standard Cell Placement with Meta-genetic Parameter Optimization", *IEEE Transactions on Computer Aided Design*, Vol.9, No.5, pp.500-511, 1990.

[106] Shahookar, K., and Mazumder, P., "VLSI Placement Techniques", *ACM Computing Surveys*, Vol.23, No.2, pp.143-220, 1991.

[107] Smith, M., Application-Specific Integrated Circuits, Addison-Wesley Longman, USA, 1997.

[108] Tan, E.C., and Yang, H., "Fast Tabular Technique for Fixed Polarity Reed-Muller Logic with Inherent Parallel Processes", *International Journal of Electronics*, Vol.85, No.4, pp.511-520, 1998.

[109] Tan, E.C., and Yang, H., "Optimization of Fixed Polarity Reed-Muller Circuits Using Dual-polarity Property", *Circuits Systems Signal Process*, Vol.19, No.6, pp.535-548, 2000.

[110] Thakur, S., and Chang, Y., "Algorithms for an FPGA Switch Module Routing Problem with Application to Global Routing", *IEEE Transactions on Computer Aided Design*, Vol.16, No.1, pp.32-46, 1997.

[111] Thomson, P., and Miller, J.F., "Symbolic Method for Simplifying AND-EXOR Representations of Boolean Functions Using a Binary Decision Technique and a Genetic Algorithm", *IEE Proceedings on Computers and Digital Techniques*, Vol.143, No.2, pp.151-155, 1996.

[112] Tsai, C.C., and Marek-Sadowska, M., "Boolean Functions Classification via Fixed Polarity Reed-Muller Forms", *IEEE Transactions on Computers*, Vol.46, No.2, pp.173-186, 1997.

[113] Tsai, C.C., and Marek-Sadowska, M., "Boolean Matching Using Generalized Reed-Muller Forms", *Proceedings of the 31st ACM/IEEE Design Automation Conference (DAC '94)*, San Diego, USA, pp.339-344, June 1994.

[114] Tsai, C.C., and Marek-Sadowska, M., "Generalized Reed-Muller Forms as a Tool to Detect Symmetries", *IEEE Transactions on Computer*, Vol.45, No.1, pp.33-40, 1996.

[115] Tsai, C.C., and Marek-Sadowska, M., "Minimisation of Fixed-Polarity AND/XOR Canonical Networks", *IEE Proceedings on Computers and Digital Techniques*, Vol.141, No.6, 369-374, 1994.

[116] Venayagamoorthy, G.K., and Gudise, V.G., "Swarm Intelligence for Digital Circuits Implementation on Field Programmable Gate Arrays Platforms", *Proceeding of the 2004 NASA/DoD Conference on Evolvable Hardware (EH '04)*, Seattle, Washington, USA, pp.83-86, June 2004.

[117] Villa, T., Kam, T., Brayton, R.K., and Sangiovanni-Vincentelli, A.L., Synthesis of Finite State Machines: Logic Optimization, Kluwer Academic Publishers, Boston, MA, USA, 1997.

[118] Wang, L., and Almaini, A.E.A., and Bystrov, A., "Efficient Polarity Conversion for Large Boolean Functions", *IEE Proceedings on Computers and Digital Techniques*, Vol.146, No.4, pp.197-204, 1999.

[119] Wang, L., and Almaini, A.E.A., "Fast Conversion Algorithm for Very Large Boolean Functions, *IEE Electronics Letters*, Vol.36, No.16, 1370-1371, 2000.

[120] Wang, L., and Almaini, A.E.A., "Optimisation of Reed-Muller PLA Implementations", *IEE Proceeding on Circuits Devices and Systems*, Vol.149, No.2, pp.119-128, 2002.

[121] Wang, L., and Almaini, A.E.A., "Exact Minimisation of Large Multiple Output FPRM Functions", *IEE Proceedings on Computers and Digital Techniques*, Vol.149, No.4, pp.203-212, 2002.

[122] Wang, L., and Almaini, A.E.A. "Multilevel Logic Simplification Based on Containment Recursive Paradigm", *IEE Proceedings on Computers and Digital Techniques*, Vol.150, No.4. pp.218-226, 2003.

[123] Wang, P.J., Liu, Y., Yang, M., and Almaini, A.E.A, "Five Valued Circuit Quantitative Theory and Design of Five-valued Twisted Ring Counter",

Proceedings of the 6th IEEE International Conference on ASIC (*ASICON* '05), Shanghai, China, pp.354-357, October 2005.

[124] Woo, N.S., "A Heuristic Method for FPGA Technology Mapping Based on Edge Visibility", *Proceedings of the 28th ACM/IEEE Design Automation Conference* (*DAC* '91), San Francisco, California, USA, pp. 248-251, June 1991.

[125] Wu, X., Chen, X., and Hurst,S.L. "Mapping of Reed-Muller Coefficients and the Minimization of Exclusive-OR Switching Functions", *IEE Proceedings on Computers and Digital Techniques*, Vol.129, Pt. E, No.1, pp.15-20, 1982.

[126] Xia, Y., Almaini, A.E.A., "Genetic Algorithms Based State Assignment for Power and Area Optimisation", *IEE Proceedings on Computers and Digital Techniques*, Vol.149, No.4, pp.128-133, 2002.

[127] Xia, Y., Almaini, A.E.A., "Best Polarity for Low Power XOR Gate Decomposition", *Proceedings of Euromicro Symposium on Digital System Design* (*DSD* '2002), Dortmund, Germany, pp.53-59, September 2002.

[128] Xia, Y., Ali, B., and Almaini, A.E.A., "Area and Power Optimisation of FPRM Function Based Circuits", *Proceedings of IEEE International Symposium on Circuits and Systems* (*ISCAS* '03), Bangkok, Thailand, pp.329-332, May 2003.

[129] Xia, Y., Almaini, A.E.A., and Wu, X., "Power Optimisation of Finite State Machines Based on Genetic Algorithms", *Journal of Electronics* (*China*), P.R. China, Vol. 20, No.3, pp.194-201, 2003.

[130] Xia, Y., Wu, X. and Almaini, A.E.A., "Power Minimization of FPRM Functions Based on Polarity Conversion", *Journal of Computer Science and Technology*, Vol.18, No.3, pp.325-331, 2003.

[131] Xia, Y., Wang, L., Zhou, Z., Ye, X., Hu, J. and Almaini, A.E.A., "Novel Synthesis and Optimization of Multi-level Mixed Polarity Reed-Muller Functions", *Journal of Computer Science and Technology*, Vol.20, No.6, pp.895-900, 2005.

[132] Xia, Y., Ye, X., Wang, L., Tap, J., and Almaini, A.E.A., "A Novel Low Power FSM Partition Approach and Its Implementation", *Proceedings of IEEE Conference on NORCHIP*, Oulu, Finland, pp. 102-105, November 2005.

[133] Xia, Y., Ye, X., Wang, L., Zou, Z., Almaini, A.E.A. , "Novel Synthesis Method of Mixed Polarity Reed-Muller Functions", *Proceedings of the 3rd IASTED Conference on Circuits Signals and Systems*, CA, USA , pp.148-153, October 2005.

[134] Xilinx XC4000E(XL) FPGAs, Data Sheet, Version 1.6, May 1999.

[135] Xilinx Virtex-4 Family, Data Sheet, Version 1.1, September 2004.

[136] Yang, H., and Tan, E.C., "Optimization of Multi-output Fixed Polarity Reed-Muller Circuits Using the Genetic Algorithm", *International Journal of Electronics*, Vol.86, No.6, pp.663-670, 1999.

[137] Yang, M., and Almaini, A.E.A., "Hybrid Genetic Algorithm for Xilinx-style FPGA Placement", *Proceedings of the 1st International Conference on ECAD/ECAE*, University of Durham, England, UK, pp. 95-100, November 2004.

[138] Yang, M., Almaini, A.E.A., Wang, L., and Wang, P.J., "An Evolutionary Approach for Symmetrical Field Programmable Gate Array Placement", *Proceedings of the 1st IEEE PhD Research in Microelectronics and Electronics* (*PRIME* '05), EPFL, Lausanne, Switzerland, pp.169-172, July 2005.

[139] Yang, M., Almaini, A.E.A., Wang, L., and Wang, P.J., "FPGA Placement Using Genetic Algorithm with Simulated Annealing", *Proceedings of the 6th IEEE International Conference on ASIC* (*ASICON* '05), Shanghai, China, pp.808-811, October 2005.

[140] Yang, M., Wang, P.J., Chen, X., and Almaini, A.E.A., "Fast Tabular Based Conversion Method for Canonical OR-Coincidence", *Proceedings of the 2005 IEEE International Conference on Computer as a Tool* (*EUROCON* '05), Belgrade, Serbia, pp.507-510, November 2005.

[141] Yang, M., Almaini, A.E.A., and Wang, P.J.,"FPGA Placement Optimization by Two-step Unified Genetic Algorithm and Simulated Annealing Algorithm", (accepted and in press) *Journal of Electronics* (*China*).

[142] Yang, S., Logic Synthesis and Optimization Benchmark User Guide Version 3.0, Technical Report, 1991.

[143] Yao, X., "Optimization by Genetic Annealing", *Proceedings of the 2nd Australian Conference on Neural Networks* (*ACNN* '91), Sydney, Australia, pp.94-97, February 1991.

Appendix A

An Example of a Circuit Optimisation

As it is known, some circuits might perform better in RM forms if they can not be optimised well in standard Boolean domain. In some cases, the circuits can be better simplified in OR/XNOR forms. Following is an example to show that the 3-variable function $f(x_2, x_1, x_0)$ can be better optimised in the DFRM expansions instead of either standard Boolean or RM expansions.

Let 3-variable Boolean function be CSOP expansion $f(x_2, x_1, x_0) = \bar{x}_2\bar{x}_1\bar{x}_0 + \bar{x}_2\bar{x}_1x_0 + \bar{x}_2x_1x_0 + x_2\bar{x}_1x_0 + x_2x_1\bar{x}_0 + x_2x_1x_0$. It can be simplified as $f(x_2, x_1, x_0) = x_0 + \bar{x}_2\bar{x}_1 + x_2x_1$. As it can be seen, the number of product terms and literals are 3 and 5 if using AND/OR forms, respectively. The following 2 sets of equations, (A.9) to (A.8) and (A.9) to (A.16) are respective polarity expansions in RM and DFRM expansions. The least number of product terms is 3 if implemented with AND/XOR forms. However, the least number of product terms is only 2 in (A.9) for polarity 0 and (A.15) for polarity 6 if implemented with OR/XNOR forms. In terms of number of literals, if using AND/XOR

forms, the least number of literals is 4 in (A.2) for polarity 1 and (A.8) for polarity 7. The least number of literals is 4 in (A.9) for polarity 0 and (A.15) for polarity 6 if using OR/XNOR forms. In this case, the circuit implemented in OR/XNOR forms outperforms other implementations.

$$f(x_2, x_1, x_0) = 1 \oplus x_1 \oplus x_1x_0 \oplus x_2 \oplus x_2x_0 \quad (A.1)$$

$$f(x_2, x_1, \bar{x}_0) = 1 \oplus x_1\bar{x}_0 \oplus x_2\bar{x}_0 \quad (A.2)$$

$$f(x_2, \bar{x}_1, x_0) = x_0 \oplus \bar{x}_1 \oplus \bar{x}_1x_0 \oplus x_2 \oplus x_2x_0 \quad (A.3)$$

$$f(x_2, \bar{x}_1, \bar{x}_0) = 1 \oplus \bar{x}_0 \oplus \bar{x}_1\bar{x}_0 \oplus x_2\bar{x}_0 \quad (A.4)$$

$$f(\bar{x}_2, x_1, x_0) = x_0 \oplus x_1 \oplus x_1x_0 \oplus \bar{x}_2 \oplus \bar{x}_2x_0 \quad (A.5)$$

$$f(\bar{x}_2, x_1, \bar{x}_0) = 1 \oplus \bar{x}_0 \oplus x_1\bar{x}_0 \oplus \bar{x}_2\bar{x}_0 \quad (A.6)$$

$$f(\bar{x}_2, \bar{x}_1, x_0) = 1 \oplus \bar{x}_1 \oplus \bar{x}_1x_0 \oplus \bar{x}_2 \oplus \bar{x}_2x_0 \quad (A.7)$$

$$f(\bar{x}_2, \bar{x}_1, \bar{x}_0) = 1 \oplus \bar{x}_1\bar{x}_0 \oplus \bar{x}_2\bar{x}_0 \quad (A.8)$$

$$f(x_2, x_1, x_0) = (x_2 + x_0) \odot (x_1 + x_0) \quad (A.9)$$

$$f(x_2, x_1, \bar{x}_0) = (x_2 + \bar{x}_0) \odot x_2 \odot (x_1 + \bar{x}_0) \odot x_1 \quad (A.10)$$

$$f(x_2, \bar{x}_1, x_0) = (x_2 + x_0) \odot (\bar{x}_1 + x_0) \odot x_0 \quad (A.11)$$

$$f(x_2, \bar{x}_1, \bar{x}_0) = (x_2 + \bar{x}_0) \odot x_2 \odot (\bar{x}_1 + \bar{x}_0) \odot \bar{x}_1 \odot \bar{x}_0 \odot 0 \quad (A.12)$$

$$f(\bar{x}_2, x_1, x_0) = (\bar{x}_2 + x_0) \odot (x_1 + x_0) \odot x_0 \quad (A.13)$$

$$f(\bar{x}_2, x_1, \bar{x}_0) = (\bar{x}_2 + \bar{x}_0) \odot \bar{x}_2 \odot (x_1 + \bar{x}_0) \odot x_1 \odot \bar{x}_0 \odot 0 \quad (A.14)$$

$$f(\bar{x}_2, \bar{x}_1, x_0) = (\bar{x}_2 + x_0) \odot (\bar{x}_1 + x_0) \quad (A.15)$$

$$f(\bar{x}_2, \bar{x}_1, \bar{x}_0) = (\bar{x}_2 + \bar{x}_0) \odot \bar{x}_2 \odot (\bar{x}_1 + \bar{x}_0) \odot \bar{x}_1 \quad (A.16)$$

Appendix B

Input and Output Data Formats

1. On-set CPOS maxterm (.max), CSOP minterm (.min) and COC maxterm (.coc) file formats are used for programs, named COCmultiseg as given in Chapter 4 and COCtabular as given in Chapter 5, respectively.

   Those formats are shown as follows. ".input", ".output", ".CPOS", ".CSOP", ".COC", ".polarity" and ".end" are the keywords of input, output, on-set CPOS maxterms, on-set CSOP minterms, on-set COC maxterms, the polarity of the expansion and the end of file. n and p indicate the number of variables and the number of polarity, respectively. "number" is the number of on-set coefficients and lies in the range of $[0, 2^n - 1]$.

   CPOS minterm coefficient format: (.min)

   .input n

   .output 1

   .CPOS number

   one set of on-set CPOS maxterm coefficients with integer number

   .end

CSOP maxterm coefficient format: (.max)

.input n

.output 1

.CSOP number

one set of on-set CSOP minterm coefficients with integer number

.end

COC maxterm coefficient format: (.coc)

.input n

.output 1

.polarity p

.COC number

one set of on-set COC maxterm coefficients with integer number

.end

2. The program used in Chapter 6 is named MMmapping. It reads on-set FPRM coefficients as an input and output on-set MMPRM coefficients as an output. On-set FPRM coefficients and on-set MMPRM coefficients file formats are shown as follows. ".input", ".output", ".FPRM", ".MMPRM", ".polarity" and ".end" are the keywords of input, output, on-set FPRM expansion coefficients, on-set MMPRM expansion coefficients, the polarity of the expansion and the end of file.

   n and p indicate the number of variables and the number of polarity, respectively. "number" is the number of on-set coefficients and can be any integer number but lies in the range of $[0, 2^n - 1]$.

FPRM on-set expansion coefficient format: (.fprm)

.input n

.output 1

.polarity p

.FPRM number

one set of on-set FPRM expansion coefficients with integer number

.end

MMPRM on-set expansion coefficient format: (.mmprm)

.input n

.output 1

.MMPRM number

one set of on-set MMPRM expansion coefficients with integer number

.end

3. The program used in Chapter 7 is named GPlacer. It requires netlist and architecture input files.

   An example netlist input file (.net) in which the logic block is a single LUT and one D-flipflop is given as follows. More details can be found in [19].

   .input a pinlist: a

   .input b pinlist: b

   .input c pinlist: c

   .input d pinlist: d

.input e pinlist: e

.output out:xor5

pinlist: xor5

.clb [3]

pinlist: a b c d [3] open

subblock: [3] 0 1 2 3 4 open

.clb xor5

pinlist: e [3] open open xor5 open

subblock: xor5 0 1 open open 4 open

An example of simple architecture input file (.arch) is given as follows. More details can be found in [19].

io_rat 2

chan_width_io 1

chan_width_x uniform 1

chan_width_y uniform 1

inpin class: 0 bottom

inpin class: 0 left

inpin class: 0 top

inpin class: 0 right

outpin class: 1 bottom

inpin class: 2 global top

subblocks_per_clb 1

subblock_lut_size 4

The output file of GPlacer (.gplacement) is given as follows:

The first line of the placement file list the netlist and architecture files used to generated placement. All the following lines have the format as follows.

block_name x y subblock block_number

The block name is the name of the block given in the .net file. x and y are the row and column in which the block is placed, respectively. The subblock number is "0" for CLBs and can be any number but lies in the range of $[0, iorat]$ for IOBs, where $iorat$ is the IO ratio.

An example placement file (.gplacement) is given as follows.

Netlist file: example.net Architecture file: example.arch

Array size: 5 x 5 logic blocks

#block_name	x	y	subblock	block_number
a	0	1	0	#0
b	1	0	0	#1
c	0	2	0	#2
d	1	6	1	#3
e	1	6	0	#4
out:xor5	0	2	1	#5
xor5	1	4	0	#6
[3]	1	2	0	#7

Appendix C

PLA File Format and Its On-set Minterms and Maxterms

This format is used by programs which manipulate PLAs to describe the physical implementation. Lines beginning with a "#" are comments and are ignored. Lines beginning with a "." contain control information about the PLA. The control information is given in the following order:

.i <number of inputs>

.o <number of outputs>

.p <number of product terms (π-terms)>

.e <the end of the pla description>

What follows then is a description of the AND and OR planes of the PLA with one line per product term. Connections in the AND plane are represented with a "1" for connection to the non-inverted input line and a "0" for connection to the inverted input line. No connection to an input line is indicated with "-". Connections in the OR plane are indicated by a "1" with no connection being indicated with "0". "-" indicates "don't care". Spaces or tabs may be used freely

```
.n 3
.o 1
.p 3
- 1 0 1
0 1 1 -
1 - - 1
.e
```

Figure C.1: An example of PLA file.

x_2	x_1	x_0	$f(x_2, x_1, x_0)$
-	1	0	1
0	1	1	-
1	-	-	1

Figure C.2: The product terms of example of PLA file.

and are ignored.

An example of PLA file is given in Figure C.1. Its equivalent functionality is shown in Figure C.2. The function can be expanded to canonical form in the corresponding truth table depending on the value of "don't care". For example, Table C.1 shows a tabular representation when "don't care" product term "011" is set to "1". The K-map is shown in Figure C.3. As a result, switching function is

$$f(x_2, x_1, x_0) = x_2 + x_1 \tag{C.1}$$

The respective CSOP minterm and CPOS maxterm expansions are

$$\begin{aligned} f(x_2, x_1, x_0) = {} & \bar{x}_2 x_1 \bar{x}_0 + \bar{x}_2 x_1 x_0 + x_2 \bar{x}_1 \bar{x}_0 \\ & + x_2 \bar{x}_1 x_0 + x_2 x_1 \bar{x}_0 + x_2 x_1 x_0 \end{aligned} \tag{C.2}$$

$$f(x_2, x_1, x_0) = (x_2 + x_1 + x_0)(x_2 + x_1 + \bar{x}_0) \tag{C.3}$$

Table C.1: Truth table for PLA file when "don't care" product term "011" is set to "1".

x_2	x_1	x_0	$f(x_2, x_1, x_0)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

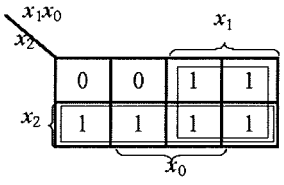


Figure C.3: K-map of example PLA file when "don't care" product term "011" is set to "1".

But if the "don't care" product term "011" is set to "0", the switching function is

$$f(x_2, x_1, x_0) = x_2 + x_1\bar{x}_0 \tag{C.4}$$

The CSOP minterm CPOS maxterm expansions are

$$f(x_2, x_1, x_0) = \bar{x}_2x_1\bar{x}_0 + x_2\bar{x}_1\bar{x}_0 + x_2\bar{x}_1x_0 + x_2x_1\bar{x}_0 + x_2x_1x_0 \tag{C.5}$$

$$f(x_2, x_1, x_0) = (x_2 + x_1 + x_0)(x_2 + x_1 + \bar{x}_0)(x_2 + \bar{x}_1 + \bar{x}_0) \tag{C.6}$$

Appendix D

Attached Disk

The attached disk contains the programs developed in the thesis and electronics version of the thesis.

1. Bidirectional multi-segment conversion program (COCmultiseg) between Standard Boolean and fixed polarity COC expansions of any polarity. This program can read CSOP minterms generated from the PLA file as an input and output fixed polarity COC expansions of any polarity. It can also read fixed polarity COC expansions of any polarity as an input and output CPOS maxterms. The input and output file formats can be found in Appendix B. More details can be found in Chapter 4.

2. Bidirectional serial and parallel tabular conversion program (COCtabular) between Standard Boolean and fixed polarity COC expansions of any polarity. This program can read CSOP minterms and CPOS maxterms generated from the PLA file as an input and output fixed polarity COC expansions of any polarity. It can also read fixed polarity COC expansions of any polarity as an input and output CPOS maxterms. The input and output file formats can be found in Appendix B. More details

can be found in Chapter 5.

3. On-set table method program (MMmapping) mapping from FPRM expansion to MMPRM expansion. This program reads FPRM expansion generated from the PLA file as an input and output MMPRM expansion. The input and output file formats can be found in Appendix B. More details can be found in Chapter 6.

4. Symmetrical FPGA Genetic algorithm program (GPlacer). This program reads ".net" file generated from VPack from VPR [18, 19] and ".arch". The ".net" file includes each net information and ".arch" file includes symmetrical FPGA architecture information, as shown in Appendix B. The placement output file format can also be found in Appendix B. More details can be found in Chapter 7.

5. Script language verifies that on-set CSOP minterms and CPOS maxterms generated from PLA are correct.

6. Script language is used to run programs against the benchmarks.