Ain Shams University
Faculty of Engineering
Computer And Ststems Engineering Department

Optimization of Digital circuits Using Binary Decision Diagram As Applied to Nuclear Reactor System circuits

A Thesis
Submitted in Partial Fulfillment of the
Requirements of the degree of
Ph.D. in Electrical Engineering

Submited By
Eng. Lamia Khalil Abdel Aziz
Nuclear Regulator & Safety Center
Atomic Energy Authority

Supervised By

Prof. Dr. Abdel Monem Wahdan
Prof. Dr Fayza M. Abdel-Mageed
Dr. Ayman Wahba

Cairo 2005

Table of Contents

1 Introduction To Boolean Function Manipulation…….…….....1
1.1 Introduction…...……..……..…..1
1.2 Manipulation of Boolean functions methods….……...2
1.2.1 Truth Tables representation …….……..2
1.2.2 Minterm & Maxterm ……...…3
1.2.3 Parse Tree..…….……...……5
1.2.4 Graph Representation……..…….……...5
1.3 Simplification of Boolean functions……..6
1.3.1 K-map……7
1.3.2 Binary decision iagram……....9
1.4 Overview of the thesis……..11
1.5 Heuristics Implementations…….……....11

2 BDD Manipulations For Boolean Function …….……12
2.1 Introduction……...12
2.2 Notation for Boolean functions……12
2.3 BDD & OBDD Representations……..13
2.4 The Correspondence between Function Graph and Boolean Function……14
2.5 Effect of Variable Ordering…….15
2.6 Complexity characteristic of variable ordering…….16
2.7 Data structure..……16
2.8 Operations…….17
2.8.1 Reducing operation……17
2.8.2 Apply operation……..19
2.8.3 Restrictions …..……..20
2.8.4 Compose……..20
2.8.5 Satisfying set……...23
2.9 Memory management……..24
2.10 Attributed edges…….24
2.10.1 Negative edges……..24
2.11 BDD Package…….25
2.11.1 BDD Package Design Issues……27
2.12 Applications Field……..30
2.12.1 Verification……...30
2.12.2 Logic synthesis……..31
2.12.3 Testing ……...…...31

3 Overview of variable order heuristics……..…….32

3.1 Introduction……..32
3.2 Static approaches……..32
3.2.1 Level heuristic…….....32
3.2.2 Variable ordering algorithm …….....33
3.3 Dynamic approaches…….35
3.3.1 Variable swap……...35
3.4 Window permutation……36
3.5 Sifting algorithm…….....38
3.6 Group sifting……..39
3.6.1 Preliminaries……39
3.6.2 Variable aggregation……....39
3.6.3 Reordering schemes……..40
3.7 Block restricted sifting ……..40
3.7.1 Compute the sub=function profile……..41
3.7.2 Find local minima and define variable blocks……41
3.8 experimental results……42
3.9 Genetic algorithms……...44
3.9.1 Outlines of the basic genetic algorithms in BDD……46
3.9.2 Selection……..46
3.9.3 Crossover & Mutation……...49
3.10 GA for variable order BDD……...50
3.10.1 Genetic Operators……50
3.10.2 Algorithm…….....51
3.11 Simulated Based Algorithm…….....52
3.11.1 Operators…….....52
3.11.2 Algorithm……..53

4 Novel Proposed Approaches……..……54
4.1 Introduction…….....54
4.2 K-average sifting algorithm ……..…...…….……//..54
4.2.1 Basic definition…….. ...55
4.2.2 Implementation of K-average sifting…….....56
4.2.3 Experiments results……..58
4.2.4 Conclusion……..59
4.3 New Genetic Algorithm……...60
4.3.1 Representation and objective function……....61
4.3.2 Initialization……....61
4.3.4 Operators……61
4.3.5 the algorithm……...63
4.3.6 Experimental results……..64
4.3.7 Conclusion……...66
4.4 K-average Linear Sifting ……66
4.4.1 Transformed BDD…….....66

4.4.2 XOR Transformation & implementation…….....67
4.4.3 Linear transformation……...68
4.4.4 Implementation of K-average Linear sifting……...69
4.4.5 Experimental results…….....73
4.4.6 Conclusion……...73
4.5 linear genetic algorithm …….....73
4.5.1 Experimental results……74
4.6 K-exact algorithm……75
4.6.1 Basic Definitions…….……75
4.6.2 New Lower Bound (NLB)…….....76
4.6.3 Average Lower Bound (ALB)…….....77
4.6.4 K-exact algorithm implementation…….....78
4.6.5.1 Implementation using NLB & ALB……..80
4.6.6 Experimental results…….....80
4.6.7 Conclusion……..…….....81

5 Reactor Safety System…….....83
5.1 The Multipurpose Reactor MPR……..83
5.1.1 Type of Reactor……...83
5.1.2.1 Reactor Protection System – RPS…….....84
5.1.2.2 Supervision and Control System SCS…….84
5.2 Reactor protection system [RPS……...86
5.2.1 Design Criteria ....……86
5.2.2 RPS Configuration…..…….87
5.2.3 Architecture……;……88
5.2.3 Specifications……;……107
5.3 Trip unit & Ordering heuristics…..……...108
5.4 Conclusion……...109

6 Conclusions &Future Works ……..…110
6.1 Conclusions ……110
6.2 Future Work…….111

Appendix I Readme file …….....112

References…….114

Acknowledgements

I sincerely thank Prof. Dr Abdel-Moneim Wahdan for his strong supports during the PHD. I also like to thank him for his encouragements and kindness.

I am deeply grateful for Prof. Dr Fayza Abdel-Mageed for being my supervisor and co-author of the paper we published. I would like to thank her for the valuable advices, stimulating discussions and assistance in the reactor protection system

I am indebted to my advisor Doctor Ayman Wahba for his acceptance to be my supervisor. Doctor Wahba has proposed some research points, provided guidance and support. Also he has improved my technical writing skills by helping me to publish my thesis work as paper conferences

I also like to deeply thank Prof. Dr. Ashraf Salem for proposing my research point.

I would like to thank my parents, my brother and my sisters for their support and encouragement specially my sister sally.

Finally, I would like to thank my husband and my children Essam, Farah, Mostafa and Marwan for their support. They have helped me beyond what this acknowledgment can fairly tell. This is for them.

Abstract

Boolean algebra forms the backbone of computer science and digital system design. Most of problems such as digital logic design, testing, artificial intelligence are expressed by a sequence of operations on Boolean function.

BDD's as a data structure for representing of Boolean functions were first popularized by [Ake 78] and [Mor 82]. Then they gained a widespread use after the restricted form ROBDD was presented by [Bry 86] because of its canonical representation form and manipulation efficiency. Due to the BDD size sensitivity to the variable ordering, a big effort was made toward the generalization of the OBDD structure to obtain more compact representations, by optimizing the variable ordering. There are examples that show exponential gabs in the sizes of functionally equivalent ROBDDs which differ only in their variable orderings. Since the problem of finding an optimal variable ordering is known to be NP-hard and there is a lack of knowledge about the approximability of the problem, there is a strong need for good working heuristics.

This thesis is focused on finding good variable ordering to optimize and accelerate the construction of BDD in three different directions. Those directions depend on the most popular heuristic, ***sifting,*** introduced by [Rud 93]. The basic idea behind the sifting algorithm is to look for the best position for each variable by moving it through the whole ordering one position at a time. The basic operation used to implement these movements is the exchange of two adjacent variables (swapping operation) which takes a time proportional to the number of nodes labeled by the swapped variables. The total time for sifting grows very fast with the number of variables, Lets consider only one variable move over a thousand of variables and we can see clearly that it takes too much time.

This thesis proposes three approaches with five algorithms to optimize& reduce BDDs construction time. The first approach is considered as an extension of the ***sifting*** approach, the second approach introduces two algorithms based on ***genetic*** algorithms as a new direction. The third uses the ***exact*** algorithms which gives the minimum BDD size for small circuits.

The first approach is based on the idea of reducing the number of swap operations by restricting its application on certain variables. Determination of those variables follow from a communication complexity argument: a weak information flow between two levels of an OBDD indicates candidate variable for swapping. The information flow can be estimated by an easily computable OBDD characteristic that we call ***share value***.
This first approach improves the time performance of the original ***sifting*** strategy considerably without causing a loss in the final size. Similarly, as in the case of the original sifting, no additional knowledge about the represented function is used. The second algorithm in this approach combines the previous idea with linear transformation.
The second approach that we present in the thesis uses those algorithms as a minimization tool in the field of ***genetic*** algorithms.
Furthermore, the third approach introduces two algorithms for speeding up the ***exact*** algorithm by using new lower bounds known from VLSI design.

For this thesis two software packages had been implemented, in C under RedHat Linux Operating system based on CUDD [Som 02]. In each package three new and different algorithms, in addition to the known algorithms, are implemented and tested. The packages features 6 new commands and they have been efficiently used to minimize and accelerate ISCAS'85 circuits and LGSynth93 circuits with up to 21 variables input**.** .

Chapter 1
Introduction to Boolean Function Manipulation

1.1 Introduction

Digital design is concerned with the design of digital electronic circuits. The subject is also known by other names such as logic design, digital logic, switching circuits and digital systems. Digital circuits are employed in the design of all systems in our live such as digital computers, control systems, data communications and many other applications that require electronic digital hardware.
In order to create any integrated circuit we must proceed in four stages: design, fabrication, testing and packaging Figure 1.1 illustrates the four stages and their contents. The design phase can be divided into three sequential procedures starting with modeling, synthesis & optimization, and verification.
Modeling is the corner stone of microelectronic design because it translates the designer idea and represents the vehicle used to convey information. It must be machine readable. Today it is implemented by using a hardware description language such as verilog or VHDL.
Circuit synthesis is the second procedure in the design stage. it starting in the designer's mind when sketching the first model. Its aim is to generate the details of the circuit and is performed by a stepwise refinement process during the detailed process of the original model in the first procedure. As the synthesis proceeds in representing the model more information is needed according to the technology and the desired design implementation such as the wires width in the geometric layout models.
Circuit optimization is always combined with synthesis. It enhances the overall quality of the circuit. Here quality means the circuit performance which consists of time, overall area, and testability. The time performance is the time needed to process an amount of information, the overall area performance is an objective of circuit design because smaller circuits relate to more circuit per wafer which minimizes the manufacturing cost. The testability performance means how easy it is to the circuit after manufacturing.
Verification consists of verifying the consistency of the models used during the design process as well as checking some properties of the original model.

The performance of digital systems greatly depends on the efficiency with which the Boolean functions are manipulated.

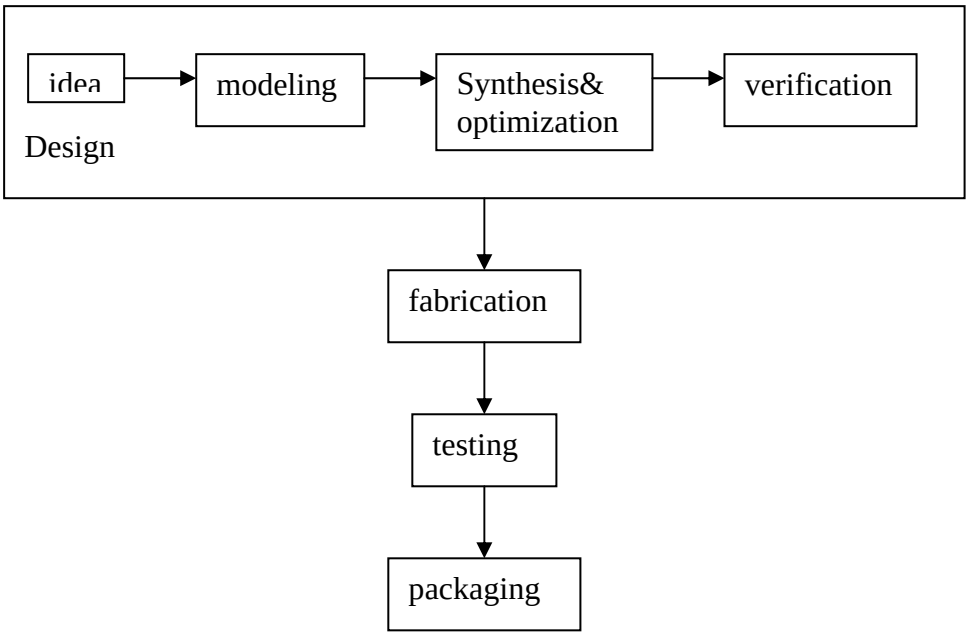


Figure (1.1) four stages to crate a digital circuit

A good data structure is the magic key to efficient Boolean function (switching function) manipulations. Some of the classical methods for representing and manipulating Boolean function are truth tables, parse tree, minterm & maxterm canonical form, cube sets and graph representation.

1.2 Manipulation of Boolean functions methods

1.2.1 Truth tables representation

A Boolean function is an expression formed with binary variables, joined with binary operators, AND, OR and unary operator NOT, parentheses and an equal sign. For example figure (1.2) illustrates the truth table for the following functions:

$$f_1 = xyz'$$
$$f_2 = x + y'z$$
$$f_3 = x'y'z + x'yz$$
$$f_4 = xy' + x'z$$

Which have three binary variables (x,y,z) we need a list of 2^3

combination of 1's and 0's of the binary variables. In general we need 2^n combinations to represent a function containing *n* binary variables
Truth tables are advantageous in that any Boolean function expressed as a truth table is in a canonical form (always has the same representation) so checking equivalence is simple,

X	*y*	*z*	f_1	f_2	f_3	f_4
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	1	0	1
1	0	1	0	1	0	1
1	1	0	1	1	0	0
1	1	1	0	1	0	0

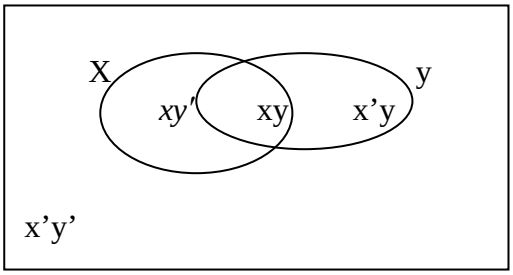
Figure (1.2) truth table (f_1,f_2,f_3,f_4)

computing the complement of a function is simple, computing the AND & OR of two functions is simple, checking for tautology is simple. They are suitable for manipulation on computers especially on high speed vector processors or parallel machines but the great drawback of truth table representations is the tremendous storage requirement.

1.2.2 Minterm & Maxterm

Any two binary variables (x,y) combined with an AND operation take four possible combinations : $xy, x'y, xy', x'y'$. Each of these AND terms represents one of the distinct areas in Venn diagram of figure (1.3) and is called minterm or a standard product. In a similar way *n* variables can be combined to form 2^n minterm. The 2^n different minterms can be combined by a method similar to the one shown in figure (1.4), for the case of three variables. The binary numbers from *0* to *n-1* are listed under the *n* variables. Each minterm is obtained from an AND term of the *n* variable with each variable being primed if it has a binary value *0* or unprimed if *1*. A symbol for each minterm is shown in the form m_j where *j* is the decimal number corresponding to the binary number of the minterm designated.
The maxterm are computed in a similar way except that the n variable form an OR terms.

From the previous discussion it is easy to demonstrate an important property of Boolean algebra: any Boolean function can be expressed as a sum of minterms or product of maxterms (They are also called POS forms, SOP forms, cube sets, PLA forms, covers or two level logics)
Cube sets sometime give more compact representation than truth tables but redundant cube may appear in logic operation and they need to be reduced for checking the equivalence or tautology so



Figure(1.3) Venn diagram of two variables.

	Minterm		Maxterm	
X Y Z	term	designation	term	desighnatio
000	*x'y'z'*	m_0	*x+y+z*	M_0
001	*x'y'z*	m_1	*x+y+z'*	M_1
010	*x'yz'*	m_2	*x+y'+z*	M_2
011	*x'yz*	m_3	*x+y'+z'*	M_3
100	*xy'z'*	m_4	*x'+y+z*	M_4
101	*xy'z*	m_5	*x'+y+z'*	M_5
110	*xyz'*	m_6	*x'+y'+z*	M_6
111	*xyz*	m_7	*x'+y'+z'*	M_7

Figure (1.4)Minterm&Maxterm of three variables.

the reduction process is time consuming. The other drawbacks are the exponential size of cube sets for parity functions and the difficulty in performing the complement operation.

1.2.3 Parse tree

Parse trees sometimes give compact representation for Boolean expressions that have many input variables and cannot be representing compactly by using truth tables. The disadvantage of this method is that there are different ways to express a given function so it is very hard to check for equivalence.

Figure (1.5) illustrates the parse tree for $f = (a + b)(c + d)$

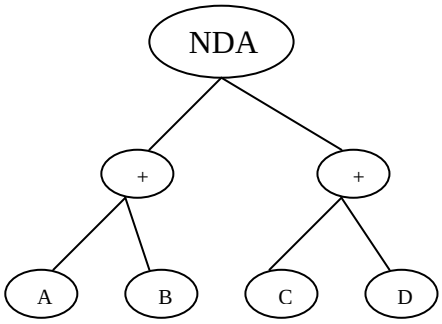


Figure (1.5) parse tree for $f = (a + b)(c + d)$

1.2.4 Graph representation

All the previous classical method are impractical because the increasing need to represent large scale functions , so we need a practical method to represent those functions. Binary Decision Diagrams (BDD's) were initially introduced by Ackers [Ake 78]. A BDD is a rooted DAG, *G(V,E).* The vertex set *V* is made up of two different types of vertices; terminal and non-terminal. Each BDD has one or two terminal vertice(s) with out-degree of zero and multiple non-terminal vertices, each with out-degree of exactly two. A terminal vertex v, has attribute *value(v)* $\in$ *{0,1}* . A non-terminal vertex v has attribute *index(v)* $\in$ *{1,2,..,n}* and two children *low(v), high(v)* $\in$ *V*. The attribute *index(v)* specifies a linear ordering of the variables in the *support* of the BDD; *i.e.* it satisfies the property such that for any non-terminal vertex v, *index(v)* < *{ index(low(v)), index(high(v)) }*.

Figure (1.6) shows the BDD of $f = (ab) + c$

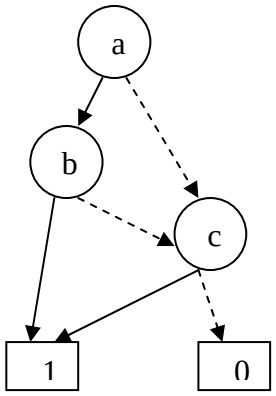


Figure (1.6) BDD of $f = (ab) + c$

Today the binary decision diagrams gain a widespread uses because of its ability for representing Boolean functions. We can easily check the equivalence of two functions because BDD's give a canonical form for the Boolean function and also their size is in the order of *kn* where *k* is a constant and *n* is the number of input variables. This size leads to suitable computation time and memory consumption.

1.3 Simplification of Boolean functions

The complexity of the digital logic gates that implement a digital function is directly related to the complexity of the of the algebraic expression from which the function is implemented, although the truth table representation of a function is unique but it can appear in many different forms. Boolean functions can be simplified by means of algebraic rules (described in figure (1.7))but this method have a drawback that it lacks specific rules to predict each succeeding step in the manipulation process. The K-map method provides a straight forward procedure for minimizing Boolean function. It is considered as a pictorial form of truth table or an extension of Venn diagram,
but it is not suitable to minimize a Boolean function with large number of input variable number due to the size of the needed tables.

Postulate 2	$(a)x + 0 = x$	$(b)x.1 = x$
Postulate 5	$(a)x + x' = 1$	$(b)x.x' = 0$
Theorem 1	$(a)x + x = x$	$(b)x.x = x$
Theorem 2	$(a)x + 1 = 1$	$(b)x.0 = 0$
Theorem 3,involution	$(x')' = 1$	
Postulate 3,commutative	$(a)x + y = y + x$	$(b)xy = yx$
Theorem 4,assiative	$(a)x + (y + z) = (x + y) + z$	$(b)x(yz) = (xy)z$
Postulate 4,distributive	$(a)x(y + z) = xy + xz$	$(b)(x + yz) = (x + y)(x + z)$
Theorem 5,demorgen	$(a)(x + y)' = x' + y'$	$(b)(xy)' = x' + y'$
Theorem 6,absorption	$(a)x + xy = x$	$(b)x(x + y) = x$

Figure (1.7) Postulates &theorem of Boolean algebra

1.3.1 K-map

The map method was first proposed by Veitch and modified by Karnaugh. It is known as Veitch diagram or Karnaugh map. The map is a diagram made up of squares, were each square represents one minterm. Since any Boolean function can be expressed as a sum of minterm, it follows that Boolean function is recognized graphically in the map from the area enclosed by those squares whose minterm are included in the function. In fact the map represents all possible ways a function may be expressed in a standard form. By recognizing different patterns we can derive multiple algebraic expressions for the same function so we can choose the simplest one, assuming that the simple exepression is the sum of product or product of sum which has the minimum number of literals. Figure (1.8) shows the map which simplify the function *f* where

$$f(x, y.z, w) = \sum(0,1,2,4,5,6,8,9,12,13,14)$$

Since the function has four variables we must use a four variable map. The minterm listed in the sum are marked with 1's in the map. Eight adjacent squares marked with 1's can be combined to form the one literal term *y'*. The remaining three 1's on the right can not be combined to give simplified term. They must be combined as two or four adjacent squares. The larger the number of squares combined, the smaller the number of literals in the term. In this example the top two 1's on the right are combined with the top two 1's on the left to give the term *w'z'*. Note that

it is permissible to use the same square more than once, we are now left with a square marked by 1 in the third row and fourth column (square 1110). Instead of taking this square alone (which will give a term of four literals), we combine it with squares already used to form an area for adjacent squares this squares comprise the two middle rows and the two end columns giving the term *xz'*, the simplified function is $f = y' + w'z' + xz'$.

In the map method we can also enter the functions expressed with product of sums ($f = (y'+z')(w'+x+z')$) by taking the complement of the function and from it find the squares to be marked with 0's which represent the minterm of *f'* (the remaining squares are marked with 1's if there is no don't care conditions). In spite of the simplicity of map simplification for truth tables, it has a great disadvantage As the number of input variable increase(n variable) the storage size increases (2^n) and also the computing time increases (exponential).so we use Binary decision diagrams which give a feasible size for many practical functions. Sometimes in the worst case BDD give an exponential size for the number of input, unlike truth tables which needs 2^n bit of memory.

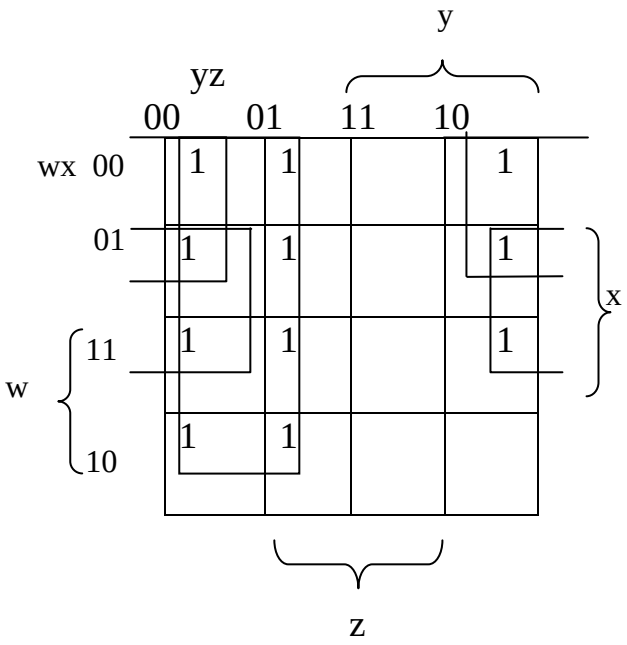


Figure (1.8) K-map for simplifying
$f(x, y.z, w) = \sum(0,1,2,4,5,6,8,9,12,13,14)$