SYNTHESIS AND OPTIMIZATION OF A 4-BIT MAGNITUDE COMPARATOR CIRCUIT USING BDD AND PRE-COMPUTATION BASED STRATEGY FOR LOW POWER

A thesis submitted in partial fulfillment of the requirements

for the award of degree of

MASTER OF TECHNOLOGY

In

VLSI Design and CAD

Submitted By

PREETI SINGH

Roll no. 601061018

Under guidance of

Mrs. Manu Bansal

Assistant Professor, ECED

T.U, Patiala

Department of Electronics and Communication Engineering

THAPAR UNIVERSITY, PATIALA

July 2012

CERTIFICATE

I hereby declare that the work which is being presented in the thesis entitled, "SYNTHESIS AND OPTIMIZATION of A 4 BIT MAGNITUDE COMPARATOR USING BDD AND PRECOMPUTATION BASED STRATEGY FOR LOW POWER" in partial fulfillment of the requirement for the award of degree of M.Tech in VLSI Design & CAD submitted in Electronics and Communication Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Ms. Manu Bansal, Assistant Professor, ECED.

The matter presented in this thesis has not been submitted in any other University/Institute for the award of degree.

Preeti Singh

Date: 6/6/12

(PREETI SINGH)

Roll No: 601061018

It is certified that the above statement made by the student is correct to the best of my knowledge and belief.

Manu Bansal 6/6/12

(Ms. Manu Bansal)

Assistant Professor

ECED, Thapar University

Countersigned by:

(Dr.Rajesh Khanna)

Professor & Head

ECED, Thapar University

Patiala-147004

(Dr. S. K. Mohapatra)

Dean of Academic Affairs

Thapar University

Patiala-147004

ACKNOWLEDGEMENT

To discover, analyze and to present something new is to venture on an untrodden path towards and unexplored destination is an arduous adventure unless one gets a true torchbearer to show the way. I would have never succeeded in completing my task without the cooperation, encouragement and help provided to me by various people. Words are often too less to reveals one's deep regards. I take this opportunity to express my profound sense of gratitude and respect to all those who helped me through the duration of this thesis. I acknowledge with gratitude and humility my indebtedness to **Ms. Manu Bansal, Assistant Professor,** Electronics and Communication Engineering Department, Thapar University, Patiala, under whose guidance I had the privilege to complete this thesis. I wish to express my deep gratitude towards her for providing individual guidance and support throughout the thesis work.

I convey my sincere thanks to **Head of the Department, Dr. Rajesh Khanna** as well as **PG Coordinator, Dr. Kulbir Singh, Assistant Professor,** Electronics and Communication Engineering Department, entire faculty and staff of Electronics and Communication Engineering Department for their encouragement and cooperation.

My greatest thanks are to all who wished me success especially my parents. Above all I render my gratitude to the Almighty who bestowed self-confidence, ability and strength in me to complete this work for not letting me down at the time of crisis and showing me the silver lining in the dark clouds. I do not find enough words with which I can express my feelings of thanks to my dear friends for their help, inspiration and moral support which went a long way in successful competition of the present study.

(PREETI SINGH)

Abstract

Symbolic model checking has been successfully applied in verification of various hardware circuits. A core technology underlying this success is the Binary Decision diagram (BDD) representation. Given the importance of BDDs in model checking, as a result the computational aspects of BDDs are well understood and BDD based model checking is to be stable in terms of performance. In BDD based realization of logic circuits, the area and power consumption is determined by the total number of nodes. A proper polarity selection of the sub-functions can not only reduce the number of BDD nodes, but also the switching activity. This study addresses the performance issue of 4-bit magnitude comparator specially for low power by developing a general evaluation methodology, and by BDD computation as well as pre-computation strategy.

INDEX

LIST OF ABBREVIATIONS

Abbreviation	Meaning
RTL	Register transfer Level
HDL	Hardware Description Language
BDD	Binary decision diagram
OBDD	Ordered Binary Decision Diagram
ROBDD	Reduced Ordered Binary Decision Diagram
VLSI	Very Large Scale integration
IC	Integrated Circuit

LIST OF FIGURES

LIST OF TABLES

Chapter-1

Introduction

1

1.1 BASIC INTRODUCTION

Today, digital design, almost without exception, uses synthesis tools at the register transfer level, which require a designer to explicitly incorporate power reducing features within the system design. Approaches exist to lowering the power. One is to reduce the operating voltage of the circuit, or to reduce the voltage supply. The weight and cost of power supply generally depends on the maximum possible power used at some instant. These are guided by a global cost function, which evaluates the current configuration with respect to user-specified objectives on area, delay, and now power consumption. The current trend towards low-power design is mainly driven by two forces the growing demand for long-life autonomous portable equipment, and the technological limitations of high-performance VLSI systems.For the first category of products, low-power is the major goal for which speed and dynamic range might have to be sacrificed. High speed and high integration density are the objectives for the second application category. The most efficient way to reduce the power consumption of digital circuits is to reduce the supply voltage. Power optimization approaches at the High-level are significant since research results Indicates that higher levels of abstraction have greater potential for power reduction, as shown in table-1.1.1.

In the past, the major concerns of the VLSI designer were area, performance, cost, and reliability; power considerations were mostly of only secondary importance. In recent years, this has changed and, power is being given weight comparable to area and performance. Several factors have contributed to this trend. This is mainly due to the remarkable success of personal computing devices which demand high speed computation and complex functionality with low power consumption

1.1.1 Power Reduction at Different Levels

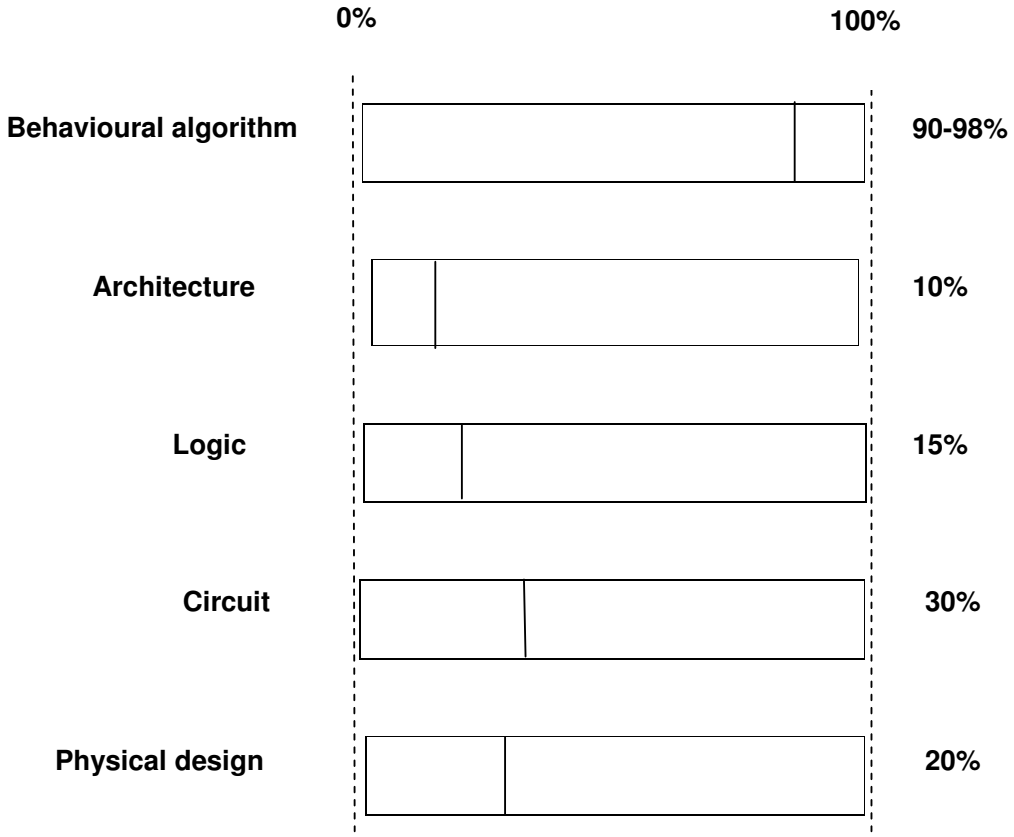


Table-1.1.1

Limits discussed so far are fundamental since they do not depend on the technology or the choice of power supply voltage. However a number of obstacles or technological limitations are in the way to approaching these limits in practical circuits, and ways to reduce the effect of these various limitations can be found at all levels of digital design ranging from device to system. Battery powered systems such as laptop, computers, electronics devices etc. the need of these systems arise from the need to extend battery life. Low power design is not only needed for portable applications but also to reduce the power of high performance systems, with large integration density and improved speed of operation.

Emerging of portable computing and communication equipment, such as laptop, palmtops, cell-phones, etc growth rate of these portable equipment are very high.

1.1.2 Contribution of various power dissipations

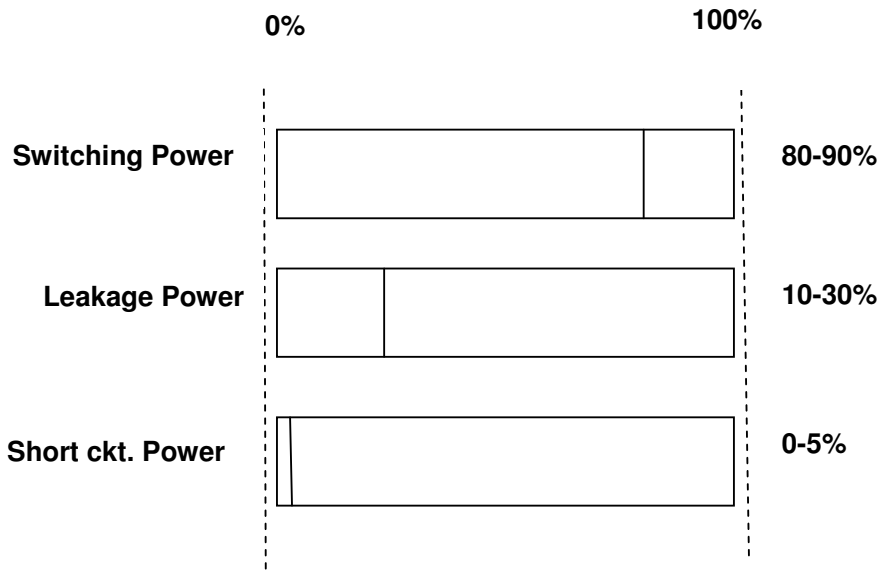


Table-1.1.2

As these are battery operated, battery life is of primary concern. Unfortunately, the battery technology has not kept up with the energy requirement of the portable equipment. Commercial success of these products depends on weight cost and battery life.

1.2 BDD Introduction

Binary Decision Diagrams (BDDs) are useful data structures for symbolic Boolean manipulations. For the last two decades, BDD have gained great popularity in representing discrete functions. BDD in general is a direct acyclic graph representation of Boolean functions proposed by Bryant and Akers [3]. The success of this technique has attracted many researchers in the area of synthesis and verification of digital VLSI circuits. Because of its efficiency in representing a variety of practical functions, BDDs became very popular data structures. The efficiency of BDDs depends mainly on its size, which is the size of their graph representations. This size depends dramatically on

the variable ordering adopted to build the BDD. Finding an optimal variable order is often worth spending considerable computational efforts because this implies savings for further operations on the constructed BDD [5].

Another critical parameter during the construction of BDDs is the maximum memory requirement, which is directly proportional to the number of nodes. A good variable ordering can lead to a smaller BDD and faster runtime, whereas a bad ordering can lead to an exponential growth in the size of the BDD and hence can exceed the available memory. Consequently, much attention has been devoted to techniques dedicated to finding good variable ordering. The evaluation time is also another important parameter, which uses BDDs to evaluate logic functions. The evaluation time is proportional to the path length in the BDD. Therefore, minimization of the path length can improve the performance of the circuit implementing a Boolean function, which will eventually enhance the performance of the final implementation. In general the minimization of the path length in Decision Diagrams is important in database structures, pattern recognition, logic simulation and software synthesis [12]. Binary Decision Diagrams form an efficient data structure for propositional logic. As such, they have been applied to a variety of integrated circuits and system design activities. Probably the most successful applications for BDDs have been made in the scope of automatic formal verification: equivalence checking for combinational circuits, equivalence checking for sequential circuits, or temporal logic model checking of sequential systems. Other applications include logic synthesis and test pattern generation.

When implementing BDD-based analyses, it is important to consider the cost of any preprocessing required by the analysis, as well as the cost of encoding the input to the analysis in BDDs. Although BDDs can store large relations compactly, precomputing any prerequisite relations and encoding them in BDDs can be costly if the input relations are large. In designing PADDLE, careful attention was paid to minimizing the amount of data that must be precomputed and encoded in BDDs [7]. In this respect, a key advantage of the PADDLE architecture is that it interleaves execution of BDD operations with traditional code; in contrast, systems such as the analysis of Berndl et al. [2002, 2003] and bddbddb,

require that all input relations be fully precomputed before the analysis begins to execute. As a result, in PADDLE, input code is encoded in BDDs only when a given method is found to be reachable by the analysis; when using bddbddb, all input code

must be encoded at the beginning, even though the analysis may not need it, because it may not be reachable.

1.2.1 OBDD:

An ***ordered binary decision diagram*** (OBDD) is a BDD with the constraint that the input variables are ordered and every source to sink path in the OBDD visits the input variables in ascending order.

1.2.2 ROBDD:

A ***reduced ordered binary decision diagram*** (ROBDD) is an OBDD where each node represents a distinct logic function. Bryant was the first to prove that the ROBDD **is** well-defined.

Bryant **also** showed the ROBDD **is** a canonical form for a logic function; that is. **two** functions are equivalent if, and only if, the ROBDD'S for each function are isomorphic.

1.3 LOGIC SYNTHESIS AND OPTIMIZATION

Logic synthesis and optimization is the process of converting a high level description of the design into an optimized gate level representation, given a standard cell library and certain design constraints. A standard cell library can have simple cells, such as basic logic gates like and, or and nor. A standard cell library is also known as technology library. The designer would partition the design into high level blocks, draw them, and describe the functionality of the circuit. Finally each block would be implemented on schematic, using the cells available in the standard cell library. The advent of computer aided logic synthesis tools has automated the process of converting high level description to logic gates. These are fed into the computer aided logic synthesis tool, which performs several iterations internally and generates the optimized gate level description. Advances in logic synthesis have pushed HDLs into the forefront of digital design technology. Logic synthesis tools have cut design cycle time significantly. Designers can design at higher level of abstraction and thus reduce design time.

Logic optimization is done internally in the logic synthesis tool and is not visible to the designer. The designer can control only the RTL description. Writing efficient RTL description, design constraints should be accurate and having a good technology library are very important to produce optimal digital circuits.

1.3.1 Synthesis Process

Automated logic synthesis has significantly reduced time for conversion from high level design representation to gates. This has allowed designers to spend more time on designing at a higher level of representation, because less time is required for converting the design to gates. Designers describe the high level design in terms of HDLs. Verilog HDL has become one of the popular HDLs for the writing of high level description. Figure -1.3.2 illustrates the synthesis process.

1.3.2 Verilog HDL synthesis

Designs are currently written in an HDL at a register transfer level. The term RTL is used for an HDL description style that utilizes a combination of data flow and behavioural constructs. Logic synthesis tool take the register transfer level HDL description and convert it to an optimized gate level netlist. Verilog one of the most popular HDLs used to describe the functionality at the RTL level. Behavioural synthesis tools that convert a behavioural description into an RTL description, RTL based synthesis is currently the most popular design method. Almost all operators in verilog are allowed for logic synthesis.

1.3.3 Process flow chart

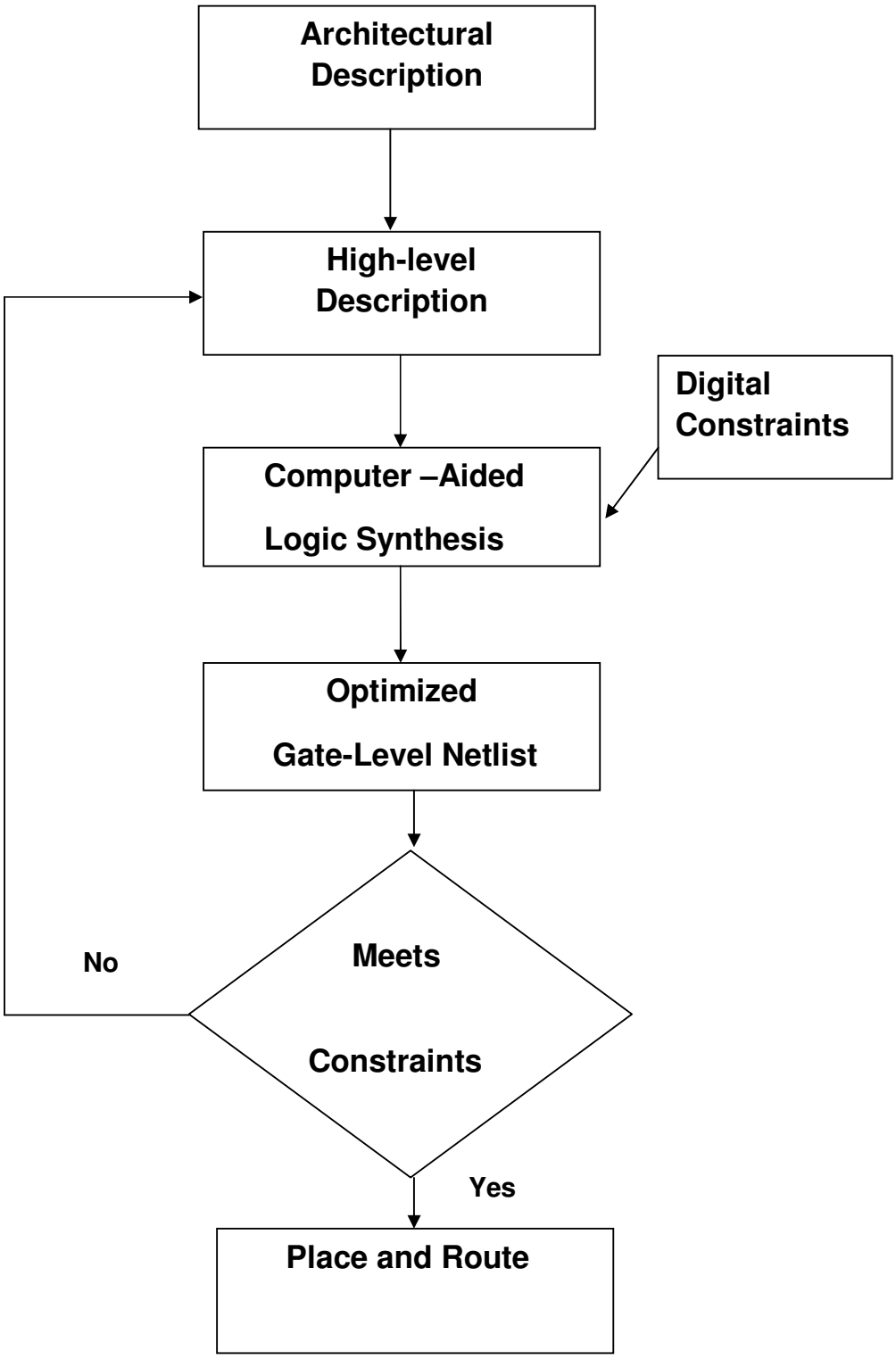


Figure -1.3.2

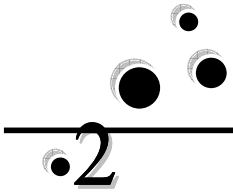
1.3.4 Impact of Logic Synthesis

Logic synthesis has revolutionized the digital design industry by significantly improving productivity and by reducing design cycle time. Before the days of automated logic synthesis, when designs were converted to gates manually, the design process had many limitations. Logic synthesis tool allow technology independent design. A high level description may be written without the IC fabrication technology in mind. Logic synthesis tools convert the design to gates, using cells in the standard cell library provided by an IC fabrication vendor. if technology changes or IC fabrication vendor changes, designers simply use logic synthesis to retarget the design to gates, using the standard cell library for the new technology. Logic synthesis tools optimize the design as a whole. If a bug is found in the gate level design, the designer goes back and changes the high level description to eliminate the bug. Then, the high level description is again read into the logic synthesis tool to automatically generate a new gate level description. Conversion from high level design to gates is fast. With this improvement, design cycle times are shortened considerably. What look months before can now be done in hours or days.

A synthesis method also can cope with significantly larger functions [8]. The basic idea is as follows: First, for the function to be synthesized a BDD is built. This can be efficiently done for large functions using existing well-developed techniques. Then, each node of the BDD is substituted by a cascade of reversible gates. Since BDDs may include shared nodes causing fan-outs (which are not allowed in reversible logic), this may require additional circuit lines. As a result, circuits composed of Toffoli or elementary quantum gates, respectively, are obtained in linear time and with memory linear to the size of the BDD. Moreover, the size of the resulting circuit is bounded by the BDD, so that theoretical results known from BDDs can be transferred to reversible circuits. Although the BDD-based synthesis often leads to larger circuits with respect to gate count and number of lines, the resulting quantum cost is significantly lower in most of the cases. Note that quantum cost is more important than gate count since they consider gates with more control lines to be more costly. Furthermore, the total number of circuit lines that have been added by the BDD-BASED SYNTHESIS is moderate considering the obtained quantum cost reductions.

Chapter-2

BDD Based Optimization



2.1 Optimization of BDD

A BDD is a directed acyclic graph; the graph has two sink nodes labelled 0 and 1 representing the Boolean functions 0 and 1. Each sink node is labelled with a Boolean variable and has two out edges labelled 1 and 0. BDDs are used in many tasks in VLSI, such as equivalence checking, property checking, logic synthesis, and false paths. In this paper we describe a new approach for the realization of a BDD package, to perform manipulations of Boolean functions. Synthesis, verification, and testing algorithms of VLSI circuits manipulate large number of switching functions. Therefore it is important to have efficient methods to represent and manipulate such functions. A large class of problems in the area of VLSI CAD can be solved by adopting efficient underlying data structures. During the last decade, several methods based on Decision Diagram have been proposed and successfully used in many industrial applications. Recently Binary Decision Diagrams have emerged as one of the best representation methods for wide range of applications**. Despite the fact that BDD is relatively old technique, its advantages as canonical representations was only recognized and made clear by Bryant.** Binary Decision Diagram (BDD) is one of the most popular data structures to represent Boolean functions. In contrast to other methods, Boolean functions based on BDD request the least space and hence have great computation efficiency. Therefore, it is widely employed in EDA industry and academic research such as logic synthesis and optimization. For BDD, its size is a key factor to decide whether it is possible to utilize the symbolic technique based on BDD. There are many techniques for the reduction of the number of nodes [5]. It is shown that the BDD size is closely related to variable

orders. The node number of reduced ordered BDD (ROBDD) for a function is likely of linear and exponential difference [16].

BDD representation of logic functions, effort is made for improving the detection techniques for logic function suitable to XOR implementation. A new XOR logic detection algorithm based on CTs is presented and a corresponding algorithm to identify whether the logic function fits for XOR logic implementation is developed [9].

The success of this technique has attracted many researchers in the area of synthesis and verification of digital VLSI circuits. BDD packages are based on recursive synthesis operation [2]. Since all binary synthesis operation can easily be described by the use of alternative concepts have been proposed. BDDs as introduced by Bryant have been traditionally used to solve the equivalence checking problem due to their canonical property. However, it is this requirement for canonicity that makes BDDs inefficient in representing certain classes of functions. For example, integer multipliers have displayed exponential memory requirements for any variable ordering .There has been increased interest in BDDs techniques that reduce the time and space needed to solve the equivalence checking problem. The combinational equivalence checking needs to perform very fast due to the use of larger designs which require more comparisons to be carried out. In such situations no dynamic variable ordering will be considered mainly due to time complexity [15].

2.2 BDD Example

Binary Decision diagram based minimization of a logic circuit plays a significant role. A Boolean function can be in factored form in multi level realization. BDD is a form of realization of multi level logic and is resulted from direct implementation of Shannon's Expansion of a logic function. Each node in a BDD signifies some logic function and a decision is made at each node of it and determines either the node is getting a zero value or one at each leg. This continues until the tree diagram reaches its leaf, Each BDD node can be easily realized using multiplexer. Thus, minimization of total number of nodes of a BDD means minimizing the number of variable present in the function, hence reduction in area [3]. A proper polarity selection of the sub-functions can reduce not only the number of BDD nodes, but also the switching.

Consider the switching function,

f= AvBC

and assume we are interested in defining a procedure for determining the binary value of f given the binary values of A, B, and C. One way to do this would be to begin by looking at the value of A. If A = 1, then f= 1 and we are finished. If A = 0, we look at B. If B = 1, then f= 0 and again we are finished. Otherwise, we look at C and its value will be the value of f Fig.2.2.1 shows a simple diagram of this procedure.

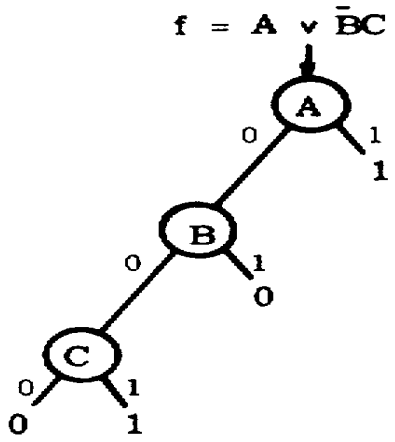


Figure 2.2.1: BDD representation of function A+B'C [3]

We enter at the node indicated by the arrow and then simply proceed downward through the diagram, noting at each node the value of its variable and then taking the indicated branch. When a 0 or 1 value is reached, this gives the value of f and the process ends.

Fig. 2.2.2 shows similar diagrams for some simple AND, OR, and XCLUSIVE-OR functions. In each case, the reader should have little difficulty in confirming that the diagram does indeed describe a procedure for finding the value of the indicated function. We shall refer to these diagrams as binary decision diagrams.'

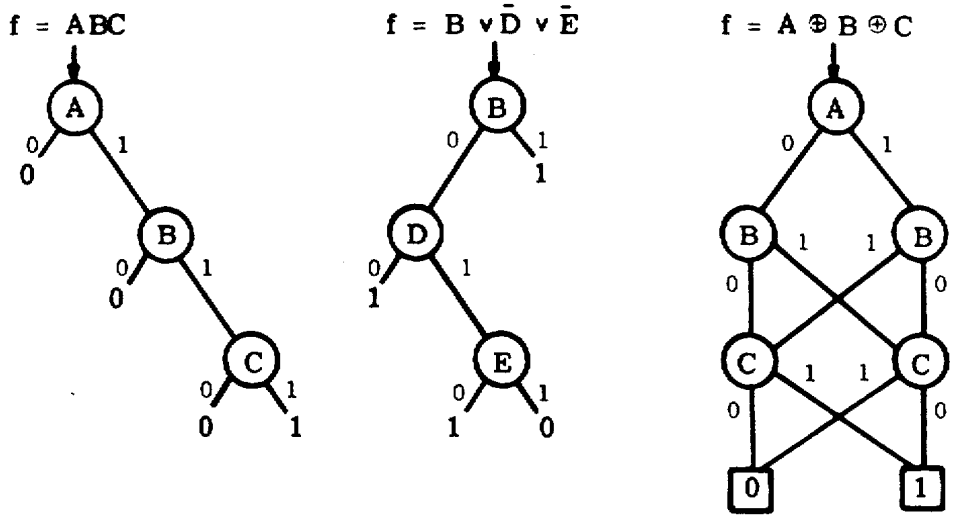


Figure 2.2.2: BDD representation of AND, OR and XOR [3]

Before examining some of the properties and uses of these diagrams, let us look at how they may be derived. A number of procedures are possible depending on the form in which the function f is defined. If, for example, we are given nothing more than the truth table for f, then a simple but possibly cumbersome procedure is to construct a diagram such as that shown in Fig. 2.2.3 which has a one-to-one correspondence between the 2" rows of the table and the 2^n paths to the outputs of the diagram. These outputs may then be labeled with the corresponding binary values of f and the required diagram automatically results.

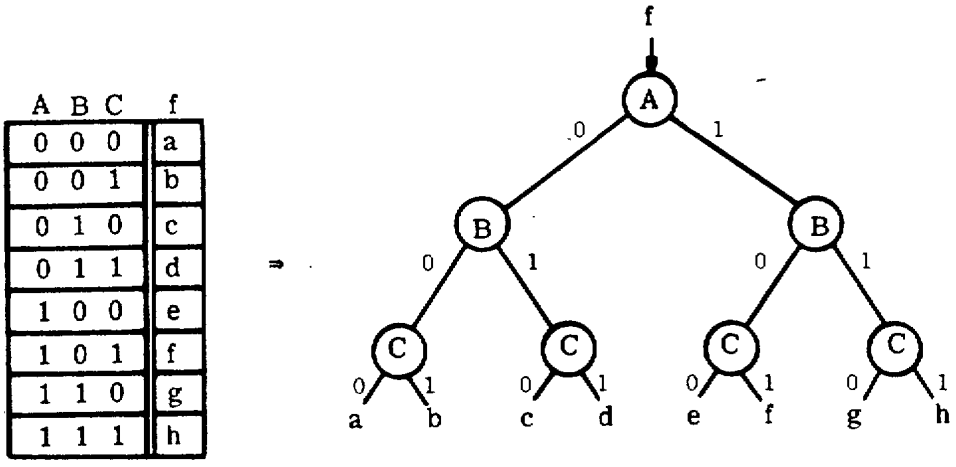


Figure 2.2.3: BDD representation using truth table [3]

With n variables, there will initially be 2^n -1 nodes in such a diagram. There are, however, several obvious ways in which this number can be considerably reduced. Consider the diagram in Fig. 2.2.4(a) which results from the truth table for f= ABC v AC. We note that the value of f obtained at the leftmost C-node is 0 regardless of the value of C. Accordingly; we can remove this node and replace it by 0. Likewise, the two rightmost C-nodes are identical in the sense that they lead to identical output values, so we can combine them into a single node. The result is Fig. 2.2.4(b). But now we note that the rightmost B-node is superfluous, since both of its branches go to the same node. Thus, we can remove it to obtain the simplified diagram of Fig. 2.2.4(c).

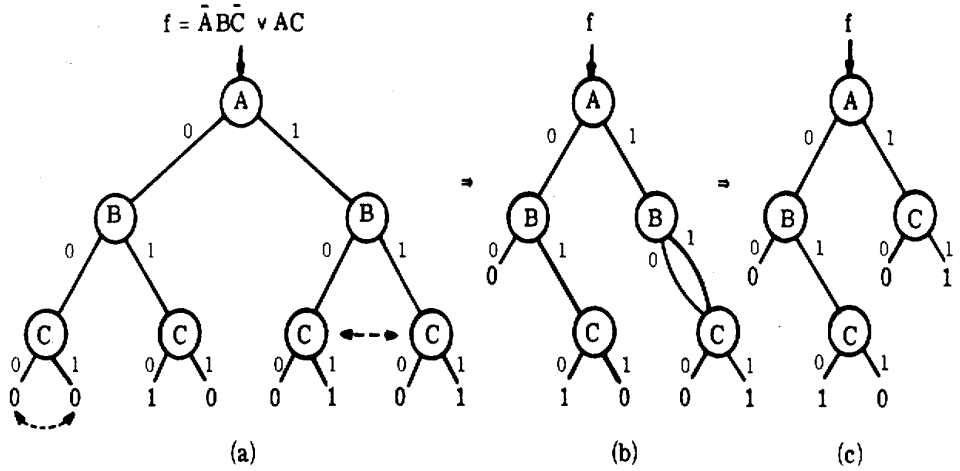


Figure 2.2.4: Optimization of BDD of function f [3]

.

Fig. 2.2.5 shows this procedure for the 5-variable function,

f= B(ACvCE)vE(ABvBD).

We begin by setting A = 0 in f to obtain the function f_o which must be realized below the A = 0 branch. We then do the same for A = 1 to obtain f_1 as shown in Fig. 2.2.5(a). Now the process is repeated for variable B to obtain the four functions in Fig. 2.2.5(b). Now we may note that two of the branches lead to the same function (DE) so these may be directed to the same node. We continue in this fashion merging identical sub functions, and then expanding each about one of its remaining variables-until all paths have terminated with a 0 or 1. Clearly, all paths will terminate in at most n steps. Note that in the process of deriving this diagram we have actually obtained the diagrams for a number of functions. If, for example, we enter the diagram of Fig. 2.2.5(d) at the node indicated by f_o, we will exit with the value of f_0. If we enter at the f_1-node, we will exit with the value of f_1 etc. Thus, we can use a single diagram to specify a number of

functions depending on the node at which we enter the diagram. If, in fact, we had initially wanted to derive a diagram for the two functions, f_o and f_1, we would have followed the same steps described above, beginning at Fig. 2.2.5(b) and ending with the diagram of Fig. 2.2.5(d) (with the A-node missing). Thus, this same procedure can be used to derive a diagram for an arbitrary number of functions.

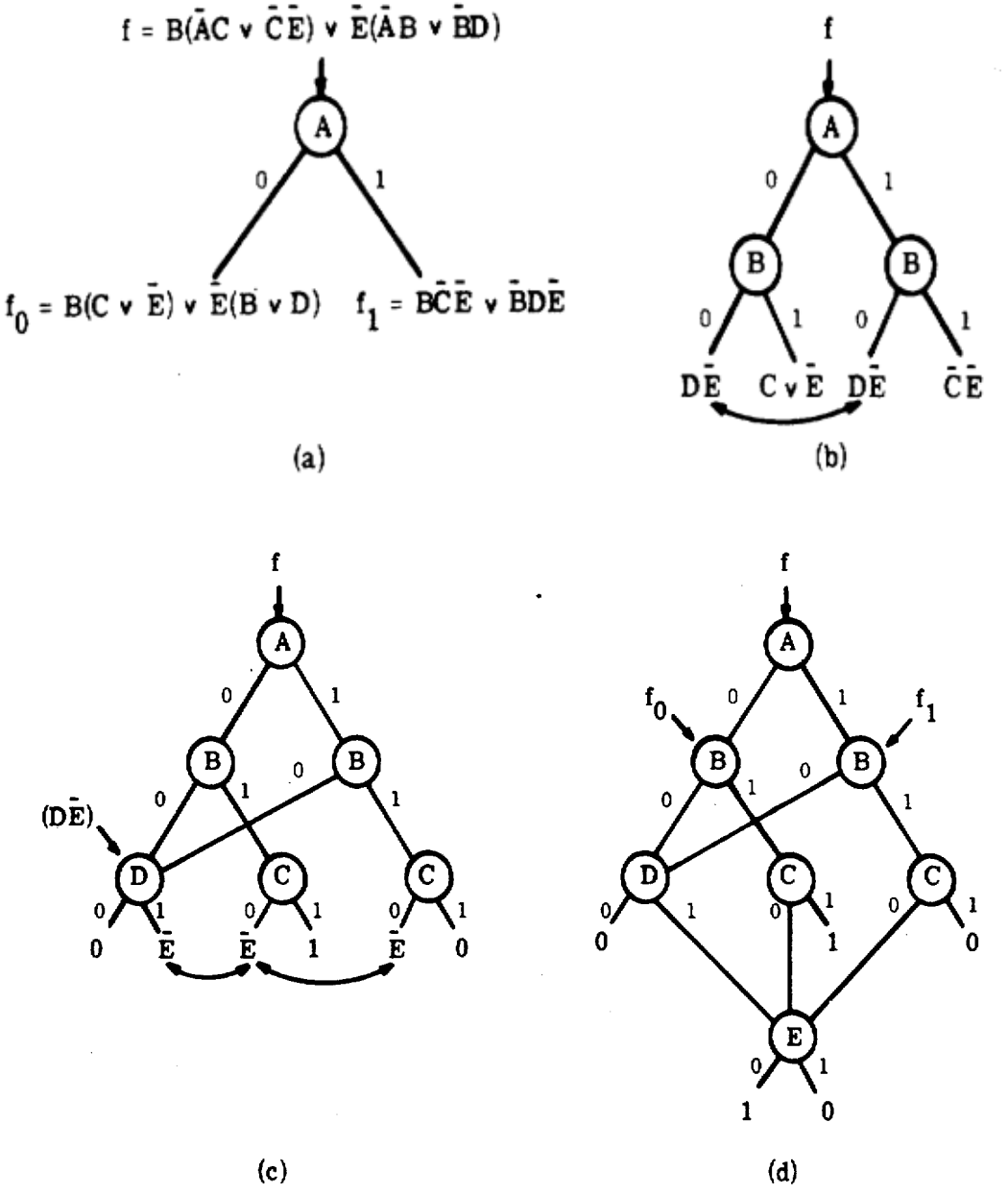


Figure 2.2.5: Optimized BDD for five variable function [3]

2.3 BDD Package [2]:

BDD packages typically share common implementation features. There are three main components in a BDD package: the BDD algorithm component, the dynamic variable reordering component, and the garbage collection component. In this section, we describe the common features in each of these components.

2.3.1 BDD Algorithm

This component computes the result BDDs for various Boolean operations. The implementation of these algorithms is typically based on depth-first traversal. The unique tables are hash tables with the hash collisions resolved by chaining. A separate unique table is associated with each variable to facilitate the dynamic-variable-reordering process. The computed cache is a hash-based direct mapped (1-way associative) cache. BDD nodes support complement edges where for each edge, an extra bit is used to indicate whether or not the target function should be complemented (Boolean negation). The advantage of this encoding is that a function and its complement can be represented by the same BDD and use this extra bit in the reference edge to interpret the BDD either in the positive or the negated form. Implementation-wise, this extra bit is typically encoded in the least significant bit of the address pointer (the reference edge) to avoid incurring extra memory cost. This encoding exploits the property that address pointers in modern machines are always at least 4-byte aligned, which means the least significant bit is always 0. Thus it can be used to encode the complement information.

2.3.2 Dynamic Variable Reordering

As the variable order can have significant impact on the size of a BDD graph, dynamic variable reordering is an essential part of all modern BDD packages. The goal for this component is to dynamically establish a good variable order as the computation progresses. Typically, when a variable reordering algorithm is invoked, all top-level operations that are currently being processed are aborted. When the variable reordering algorithm terminates, these aborted operations are restarted from the beginning. The dynamic variable reordering algorithms are generally based on the *sifting* algorithm; i.e., the variable orders are changed by exchanging nodes in one level with nodes in the adjacent level. Figure 2.3.2.1 illustrates this process. This figure shows the sifting

process for exchanging the orders of the variable a and the variable b. We tag each node with a number so that we can refer to them easily. To understand this operation, let us first discuss the equation that defines the function represented by a BDD in terms of its cofactors.

$$f = \overline{v} \cdot f|_{v \leftarrow 0} + v \cdot f|_{v \leftarrow 1}$$

where v is the variable in b's root node and the 0-cofacto f $|_{v<-0}$ is recursively defined by the reachable subgraph of b's 0-branch child. Similarly, the 1-cofactor f $|_{v<-1}$ is recursively defined by the reachable subgraph of b's 1-branch child.

Using this definition, the BDD in Figure 2.3.2.1(a) can be represented as

a'. (b'. f_0 + b . f_1) + a . (b'. g_0 + b . g_1)

By rearranging this formula, we

b'. (a'. f_0 + a . g_0) + b . (a'. f_1 + a . g_1)

which is the BDD after the sifting process in fig 2.3.2.1. Implementation-wise, node 4 and node 5 are created to represent the new children of node 1. Node 1 is updated to reference these new children and is relabeled with variable b. As for node 2 and node 3, they remain unchanged and if there are no references to them, they will be garbage collected later. Note that because node 1 might be referenced by others, it is important that node 1 is reused with its reachable graph representing the same function. Without this reuse, we will need to create a new node in place of node 1 and will have to locate all the references to node 1 and update them to reference this new node, which is very inefficient. The node-reuse technique allows the sifting algorithm to be a local operation involving only the node and its children.

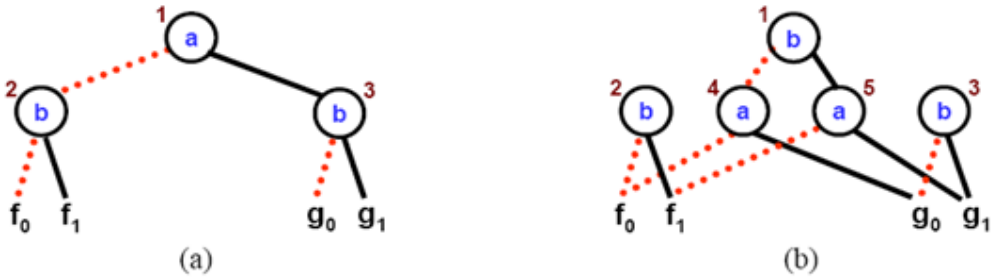


Figure 2.3.2.1: Shifting process for dynamic variable reordering [2]

Figure 2.3.2.2 illustrates the sifting process for the 2-bit comparator example Figure 2.3.2.2(a) is the BDD representation for the variable order with all the a_i's before all the b_i's. By exchanging the orders of variable a_0 and b_1, we obtain the BDD for the

interleaved variable order (Figure 2.3.2.2(b)). In this example, node 1 and node 2 are reused so that we do not need to update references from their parent (node 0). As for nodes 3, 4, 5, and 6, they are no longer part of the comparator function and if there are no other references to them, they can be garbage collected. Note that the reachable subgraph of node 0 in Figure 2.3.2.2(b) has an interleaved variable order.

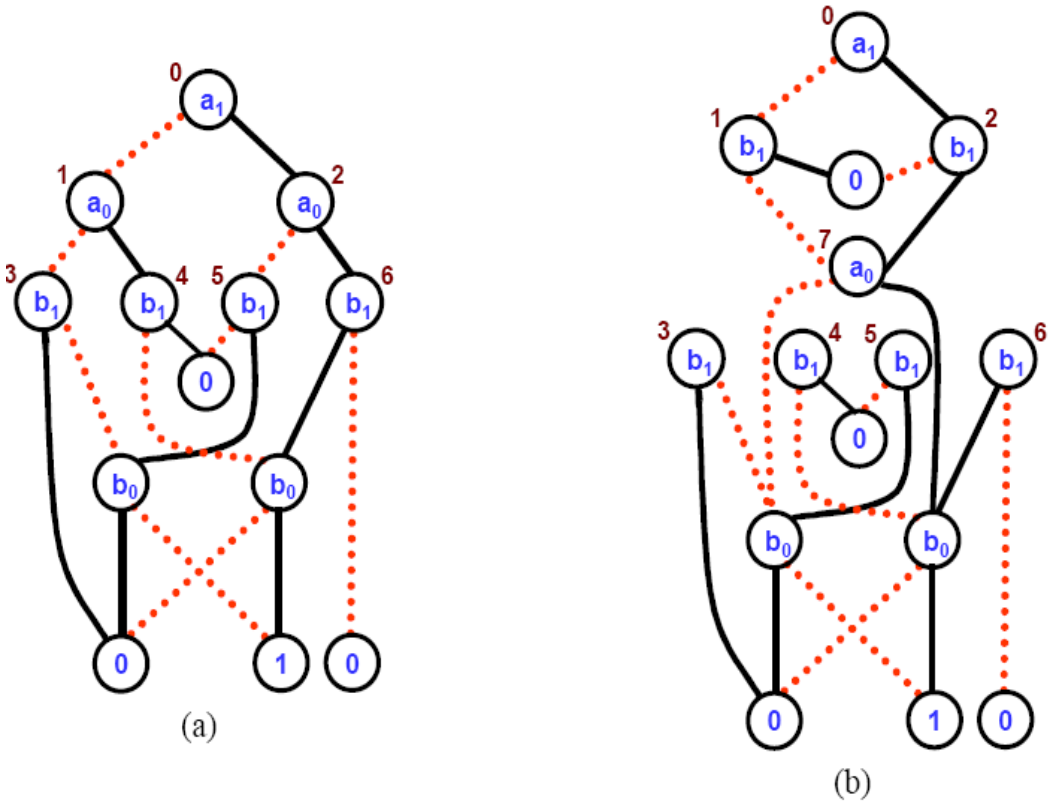


Figure 2.3.2.2: Shifting process for 2-bit comparator [2]

2.3.3 Garbage Collection

BDD computations are inherently memory intensive because after all, it is all about traversing and constructing graphs. Furthermore, in verification, many intermediate BDD results are created to arrive at a simple final answer—true or false. Thus, it is important to have a good garbage collector to automatically remove BDD nodes that are no longer useful. We will refer to a BDD node as *reachable* if it is in some BDD that external user has a reference to. As external users free references to BDDs, some BDD nodes may no longer be reachable (*deaths*). We will refer to these nodes as *unreachable* BDD nodes.

2.4 Comparator

Comparing two binary numbers for equality is a commonly used operation in computer system and device interfaces. A circuit that compares two binary numbers and indicates whether they are equal is called a comparator. Comparators interpret their input numbers as signed or unsigned numbers and also indicate an arithmetic relationship between the numbers often called magnitude comparator.

2.4.1 Comparator structure

Exclusive-R and Exclusive-NOR gates may be viewed as 1-bit comparators. Figure-2.4.2.1 shows an interpretation of a 2-input XOR gate as a 1-bit comparator. The high output is asserted if the inputs are different. The outputs of four XOR gates are ORed to create a 4-bit comparator. The output is asserted if any of the input bit pairs are different. Comparators can also be built using Exclusive-NOR gates sometimes called Equivalence gates. Figure -2.4.2.2 shows 2-input XNOR gate produces a 1 output if its two inputs are equal. A 4-bit comparator can be constructed using four XNOR gates, and ANDing all of their outputs together. The output of that AND function is 1 if all of the individual bits are pairwise equal [10].

2.4.1.1 2-input XOR gate as a 1-bit comparator

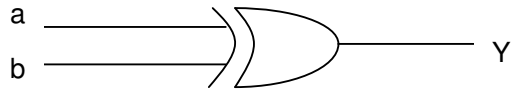


Figure 2.4.1.1 : 2-input XOR gate

Truth Table :

		F1
a	b	Y
0	0	0
0	1	1
1	0	1
1	1	0

2.4.1.2 2-input XNOR gate as a 1-bit comparator

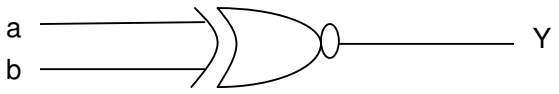


Figure 2.4.1.2 : 2-input XNOR gate

Truth Table:

a	b	Y
0	0	1
0	1	0
1	0	0
1	1	1

2.4.2 Block diagram of magnitude comparator

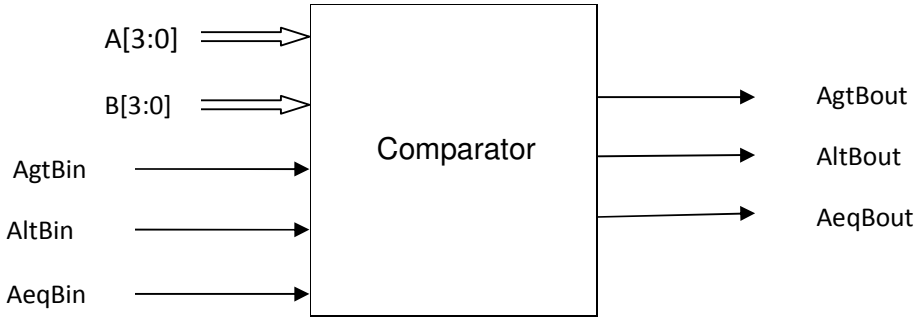


Figure 2.4.2: Block diagram of magnitude comparator

2.4.3 4-bit Magnitude Comparator:

Comparator applications are common enough that magnitude comparators have been developed commercially. The logic diagram of 4-bit magnitude comparator is shown in figure-2.4.3 [11]. It provides greater-than output A > B and less-than output A < B as

well as an equal output A = B. In normal operation exactly one input and one output should be asserted. The logic for a 4-bit magnitude comparator is as follows:

Let the two 4-bit numbers be A= $A_3A_2A_1A_0$ and B = B_3 B_2 B_1 B_0.

1. If $A_3 = 1$ and $B_3 = 0$, then A > B. Or
2. If A_3 and B_3 coincide, and if $A_2 = 1$ and $B_2 = 0$, then A > B. Or
3. If A_3 and B_3 coincide, and if A_2 and B_2 coincide, and if $A_1 = 1$ and $B_1 = 0$, then A > B.
4. If A_3 and B_3 coincide, and if A_2 and B_2 coincide, if A_1 and B_1 coincide, and if $A_0 = 1$ and $B_0 = 0$, then A > B.

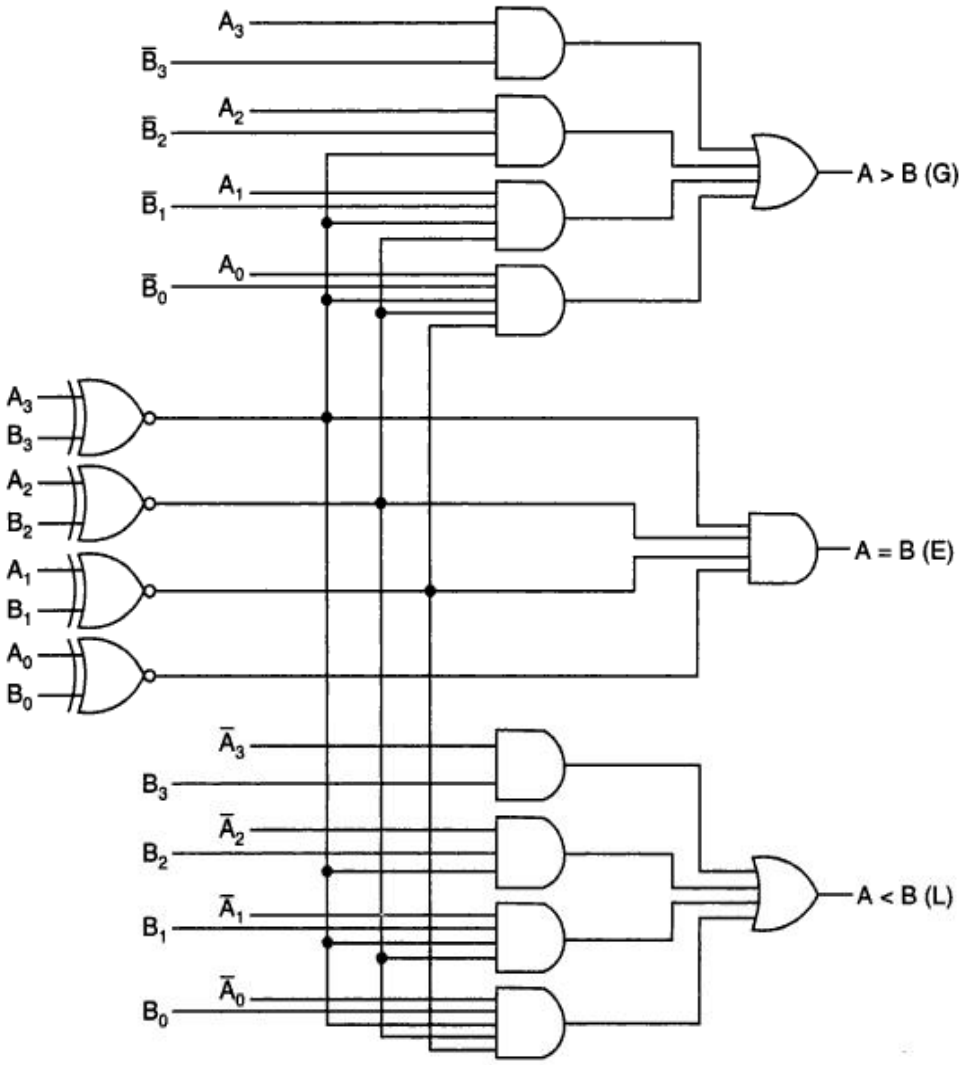


Figure 2.4.3: Logic diagram of 4-bit comparator [11]

2.4.4 Truth table of Comparator

Inputs							**outputs**		
Comparing inputs				Expansion inputs					
A3,B3	**A2,B2**	**A1,B1**	**A0,B0**	**A>B**	**A<B**	**A=B**	**A>B**	**A<B**	**A=B**
A3>B3	X	X	X	X	X	X	H	L	L
A3<B3	X	X	X	X	X	X	L	H	L
A3=B3	A2>B2	X	X	X	X	X	H	L	L
A3=B3	A2>B2	X	X	X	X	X	H	L	L
A3=B3	A2=B2	A1>B1	X	X	X	X	H	L	L
A3=B3	A2=B2	A1<B1	X	X	X	X	L	H	L
A3=B3	A2=B2	A1=B1	A0>B0	X	X	X	H	L	L
A3=B3	A2=B2	A1=B1	A0<B0	X	X	X	L	H	L
A3=B3	A2=B2	A1=B1	A0=B0	L	L	H	L	L	H
A3=B3	A2=B2	A1=B1	A0=B0	H	L	L	H	L	L
A3=B3	A2=B2	A1=B1	A0=B0	L	H	L	L	H	L

The truth table explains how the comparator functions. The top line shows that if A3 is greater than b3, the output A>B goes high. Likewise if A3 is less than B3, A is less than B. The output A<B goes high. Line 3 and 4 of the truth table show what happens if A3 and B3 are equal, A2 and B2 are compared, and the outputs are set accordingly. The bottom three lines show what happens when A3A2A1A0 is equal to B3B2B1B0 only then are the cascading inputs A>B, A<B, A=B considered in the decision.

Cascading outputs defined according to the following logic equations.

AGTBOUT = (A>B) + (A=B). AGTBIN

AEQBOUT = (A=B). AEQBIN

ALTBOUT = (A<B) + (A=B). ALTBIN

AGTBOUT is asserted if A>B or if A=B, if higher order bits are equal, we have to look at the lower order bits for answer. The arithmetic comparisons can be expressed using normal logic expressions.

(A>B) = A3.B3'+ (A3⊙B3).A2.B2'+ (A3⊙B3). (A2⊙B2). A1.B1' +

(A3⊙B3). (A2⊙B2). (A1⊙B1). A0.B0'

(A<B) = A3'.B3+ (A3⊙B3).A2'.B2+ (A3⊙B3). (A2⊙B2). A1'.B1 +

(A3⊙B3). (A2⊙B2). (A1⊙B1). A0'.B0

(A=B) = (A3⊙B3). (A2⊙B2). (A1⊙B1). (A0⊙B0)

Such expressions must be substituted into the logic equations above to obtain genuine logic equations for the comparator output. Based on the above logic equations we can further expand in terms of sum of product form as follow.

AEQBOUT = (A=B). AEQBIN [say AEQBIN=Ae]

= (A3⊙B3). (A2⊙B2). (A1⊙B1). (A0⊙B0). AEQBIN

= A3B3A2B2A1B1A0B0Ae + A3B3A2B2A1B1A0'B0'Ae

+ A3B3A2B2A1'B1'A0B0Ae + A3B3A2B2A1'B1'A0'B0'Ae

+ A3B3A2'B2'A1'B1'A0B0Ae + A3B3A2'B2'A1'B1'A0'B0'Ae

+ A3'B3'A2B2A1B1A0B0Ae + A3'B3'A2B2A1B1A0'B0'Ae

+ A3'B3'A2B2A1'B1'A0B0Ae + A3'B3'A2'B2'A1B1A0'B0'Ae

+ A3'B3'A2'B2'A1B1A0B0Ae + A3'B3'A2'B2'A1B1A0'B0'Ae

+ A3'B3'A2'B2'A1'B1'A0B0Ae + A3'B3'A2'B2'A1'B1'A0'B0'Ae

+ A3B3A2B2A1B1A0B0Ae + A3B3A2B2A1B1A0B0Ae

Total SOP terms in AEQBOUT = 16.

AGTBOUT = (A>B) + (A=B). AGTBIN [say AGTBIN=Ag]

= A3B3' + A3B3A2B2' +A3'B3'A2B2' + A3B3A2B2A1B1'

+ A3B3A2'B2'A1B1' + A3'B3'A2B2A1B1' + A3'B3'A2'B2'A1B1'

+ A3B3A2B2A1B1A0B0' + A3B3A2B2A1'B1'A0B0'

+ A3B3A2’B2’A1B1A0B0’ + A3B3A2’B2’A1’B1’A0B0’

+ A3’B3’A2B2A1B1A0B0’ + A3’B3’A2B2A1’B1’A0B0’

+ A3’B3’A2’B2’A1B1A0B0’ + A3’B3’A2’B2’A1’B1’A0B0’

+A3B3A2B2A1B1A0B0Ag + A3B3A2B2A1B1A0’B0’Ag

+ A3B3A2B2A1’B1’A0B0Ag + A3B3A2B2A1’B1’A0’B0’Ag

+ A3B3A2’B2’A1’B1’A0B0Ag + A3B3A2’B2’A1’B1’A0’B0’Ag

+ A3’B3’A2B2A1B1A0B0Ag + A3’B3’A2B2A1B1A0’B0’Ag

+ A3’B3’A2B2A1’B1’A0B0Ag + A3’B3’A2’B2’A1B1A0’B0’Ag

+ A3’B3’A2’B2’A1B1A0B0Ag + A3’B3’A2’B2’A1B1A0’B0’Ag

+ A3’B3’A2’B2’A1’B1’A0B0Ag + A3’B3’A2’B2’A1’B1’A0’B0’Ag

+ A3B3A2B2A1B1A0B0Ag + A3B3A2B2A1B1A0B0Ag

Total SOP terms in AGTBOUT= 31

ALTBOUT = (A<B) + (A=B). ALTBIN [say ALTBIN=Al]

= A3’B3 + A3B3A2’B2 +A3’B3’A2’B2 + A3B3A2B2A1’B1

+ A3B3A2’B2’A1’B1 + A3’B3’A2B2A1’B1 + A3’B3’A2’B2’A1’B1

+ A3B3A2B2A1B1A0’B0 + A3B3A2B2A1’B1’A0’B0

+ A3B3A2’B2’A1B1A0’B0 + A3B3A2’B2’A1’B1’A0’B0

+ A3’B3’A2B2A1B1A0’B0 + A3’B3’A2B2A1’B1’A0’B0

+ A3’B3’A2’B2’A1B1A0’B0 + A3’B3’A2’B2’A1’B1’A0’B0

+A3B3A2B2A1B1A0B0Al + A3B3A2B2A1B1A0’B0’Al

+ A3B3A2B2A1’B1’A0B0Al + A3B3A2B2A1’B1’A0’B0’Al

+ A3B3A2’B2’A1’B1’A0B0Al + A3B3A2’B2’A1’B1’A0’B0’Al

+ A3’B3’A2B2A1B1A0B0Al + A3’B3’A2B2A1B1A0’B0’Al

+ A3’B3’A2B2A1’B1’A0B0Al + A3’B3’A2’B2’A1B1A0’B0’Al

+ A3’B3’A2’B2’A1B1A0B0Al + A3’B3’A2’B2’A1B1A0’B0’Al

+ A3’B3’A2’B2’A1’B1’A0B0Al + A3’B3’A2’B2’A1’B1’A0’B0’Al

+ A3B3A2B2A1B1A0B0Al + A3B3A2B2A1B1A0B0Al

Total SOP terms in ALTBOUT= 31

Finally total sum of products in comparator is equal to (16+31+31=78)..

Chapter-3

Pre-computation Strategy

3

3.1 Pre-computation:

We present a powerful sequential logic optimization method that is based on selectively precomputing the output logic values of the circuit one clock cycle before they are required, and using the precomputed values to reduce internal switching activity in the succeeding clock cycle. The primary optimization step is the synthesis of precomputation logic, which computes the output values for a subset of a input conditions. If the output values can be precomputed, the original logic circuit can be "turned off' in the next clock cycle and will not have any switching activity. Since the savings in the power dissipation of the original circuit is offset by the power dissipated in the precomputation phase, the selection of the subset of input conditions for which the output is precomputed is critical. The precomputation logic adds to the circuit area and can also result in an increased clock period.

3.2 Pre-computation architecture:

We describe two different precomputation architectures and discuss their characteristics in terms of their impact on power dissipation, circuit area and circuit delay .

3.2.1 First Pre-computation architecture [6]:

Consider the circuit of Fig. 3.2.1.1. We have a combinational logic block **A** that is separated by registers *RI* and *R2.* While *RI* and *R2* are shown as distinct registers in Fig. 3.2.1.1 they could, in fact, be the same register. We will first assume that block **A** has a single output and that it implements the Boolean function *f.*

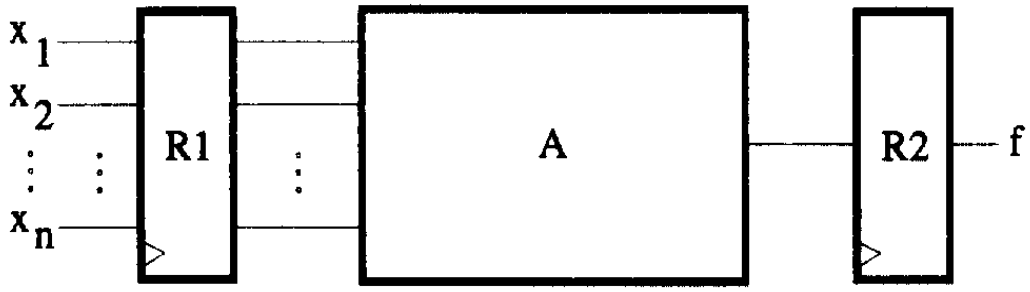


Figure 3.2.1.1 [6]

In Fig. 3.2.1.2 the first precomputation architecture is shown. Two Boolean functions g1 and *g2* are the predictor functions. We require:

g1 = 1 => f = 1 (1)

g2 = 1 => f = 0 (2)

Therefore, during clock cycle t if either g1 or g2 evaluates to a 1, we set the load enable signal of the register R1 to be 0. This means that in clock cycle t + 1 the inputs to the combinational logic block A do not change. If g1 evaluates to a 1 in clock cycle t, the input to register R2 is a 1 in clock cycle t+ 1, and if g2 evaluates to a 1, then the input to register R2 is a 0.Note that g1 and g2 cannot both be 1 during the same clock cycle due to the conditions imposed by (1) and (2). A power reduction in block A is obtained because for a subset of input conditions corresponding to g1 + g2 the inputs to A do not change implying zero switching activity. However, the area of the circuit has increased due to additional logic corresponding to g1, g2, the two additional gates shown in the figure, and the two flip-flops marked FF. The delay between R1 and R 2 has increased due to the addition of the AND-OR gate. Note also that g1 and g2 add to the delay of paths that originally ended at R 1 but now pass through g1 or g2 and the NOR gate before ending at the load enable signal of the register RI. Therefore, we would like to apply this transformation on noncritical logic blocks.

The choice of g1 and g2 is critical. We wish to include as many input conditions as we can in g1 and g2. In other words, we wish to maximize the probability of gl or g2 evaluating to a 1. In the extreme case this probability can be made unity if g1 = f and g2 = f' . However, this would imply a duplication of the logic block A and no reduction in power with a two-fold increase in area! To obtain reduction in power with marginal increases in circuit area and delay, g1 and g2 have to be significantly less complex than f. One way of ensuring this is to make g1 and g2 depend on significantly fewer inputs than f. This leads us to the second precomputation architecture of Fig. 3.2.1.2.

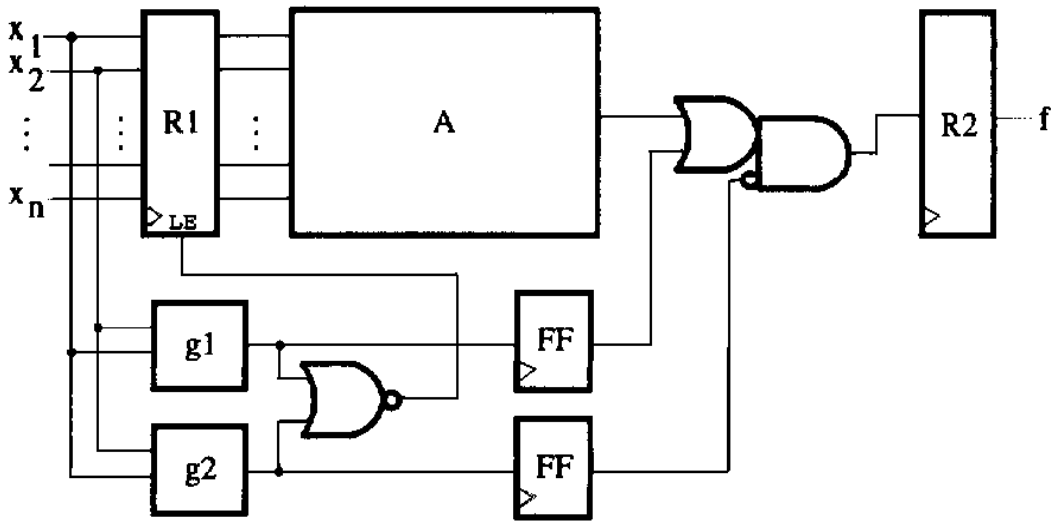


Figure 3.2.1.2: First pre-computation architecture [6]

3.2.2 Second computation architecture [2]:

In the architecture of Fig. 3.2.2.1, the inputs to the block A have been partitioned into two sets, corresponding to the registers R1 and R2. The output of the logic block A feeds the register R3. The functions g1 and g2 satisfy the conditions of (1) and (2) as before, but g1 and g2 only depend on a subset of the inputs to f. If g1 or g2 evaluates to a 1 during clock cycle t, the load enable signal to the register R2 is turned off. This implies that the outputs of R2 during clock cycle t+ 1 do not change. However, since the outputs of register R1 are updated, the function f will evaluate to the correct logical value. A power reduction is achieved because only a subset of the inputs to block A change implying reduced switching activity. As before, g1 and g2 have to be significantly less complex than f and the probability of g1 + g2 begin a 1 should be high in order to achieve substantial power gains. The delay of the circuit between R1/R2 and R3 unchanged, allowing precomputation of logic that is on the critical path. However, the delay of paths that originally ended at R1 has increased. The choice of inputs to g1 and g2 has to be made first, and then the particular functions that satisfy (1) and (2) have to be selected.

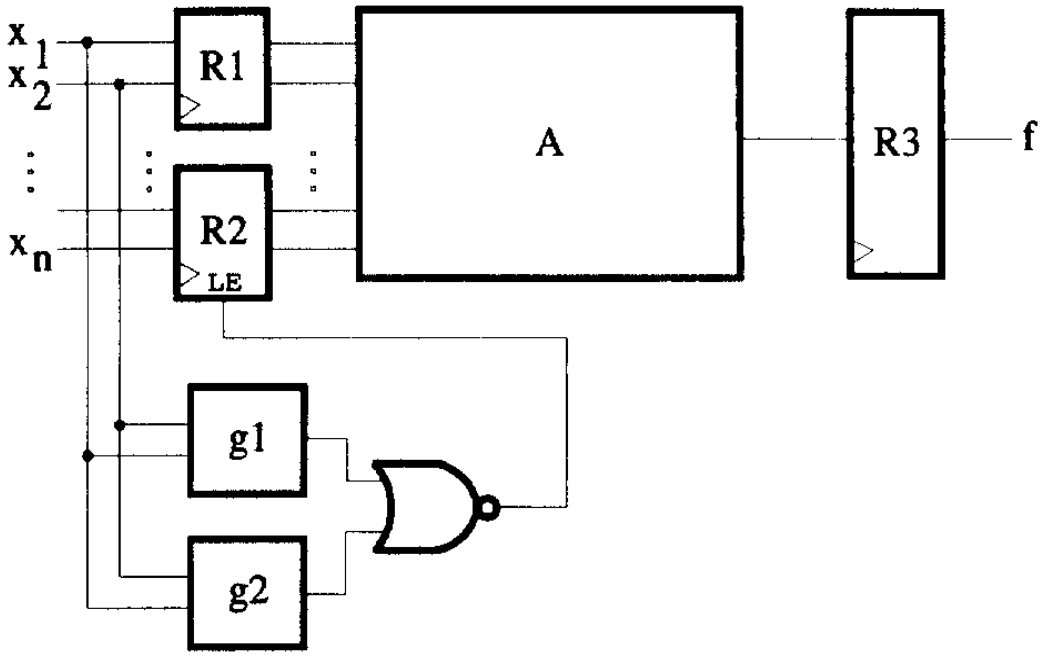


Figure 3.2.2.1: Second pre-computation architecture [6]

Second precomputation architecture is applied to a comparator circuit in fig 3.2.2.2

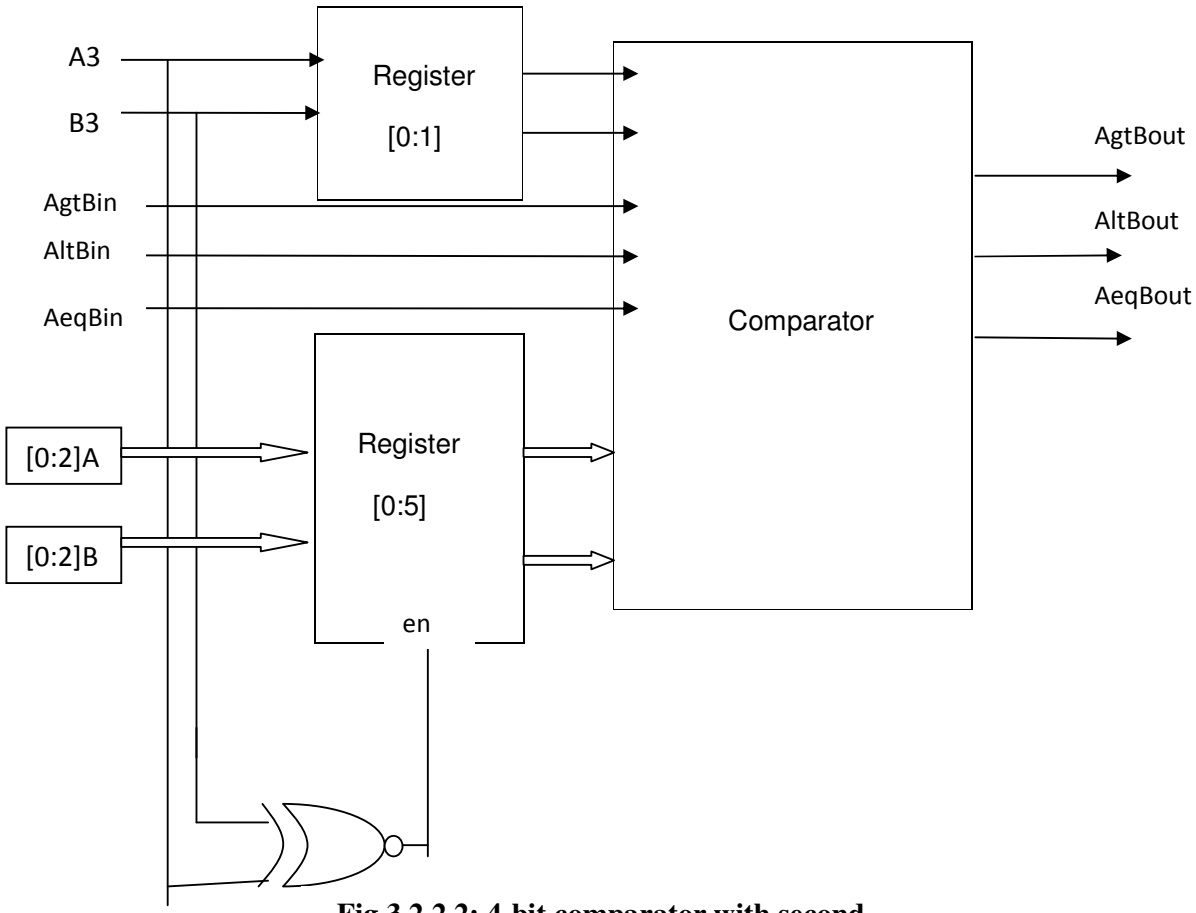


Fig 3.2.2.2: 4-bit comparator with second pre-computation architecture

In pre-computation architecture of comparator one two bit register, one six bit register and one xnor gate is implemented along with the comparator circuit as shown in figure-3.2.2.2.

3.3 Register

A collection of two or more flip-flops with a common clock input is called a register. A individual flip-flop can be used to store one bit, in machines in which data are handled in words consisting of many bits, it is convenient to arrange a number of flop-flops into a common structure called register. Registers are often used to store a collection of bits, such as a byte of a data in a computer. A single register can also be used to store unrelated bits of data.

Chapter-4

Analysis and Result

4

4.1 Synthesis with Synopsis Tool:

4.1.1 Comparator in Verilog :

Verilog has built-in comparison operators : >,>,=. These operators can be applied to bit vectors, and the bit vectors are interpreted as unsigned numbers with the most significant bit on the left, regardless of how they are numbered. When a comparison operation is used in a Verilog module, the compiler synthesizes corresponding comparator logic. The size and speed of synthesized comparator logic depends on the target technology and the optimization capabilities of the Verilog compiler. Comparator can be built from XOR or XNOR gates and an AND or OR gate. The XOR or XNOR gates all operate in parallel, and a reasonably fast AND or OR gate of any size can be built using a tree–like structure.

Like the 74x85, this module is set up for cascading information to flow from less significant to more significant stages. Thus in first if statement, if the a and b inputs are equal, then the comparison will depend on less significant stages, and the cascading outputs are set equal to the cascading inputs. If they are unequal, the next if and the else statement set the cascading outputs solely dependent on the comparison result in the current stage. After simulating this module in Synopsys tool the RTL schematic is coming as figure-4.1.2 and waveform representation of comparator module is coming as shown in figure-4.1.3.

4.1.2 RTL View :

When we define all eleven inputs individually and then generated RTL is coming as show in figure-4.1.2. Their corresponding simulation result is coming as shown in figure-4.1.3.

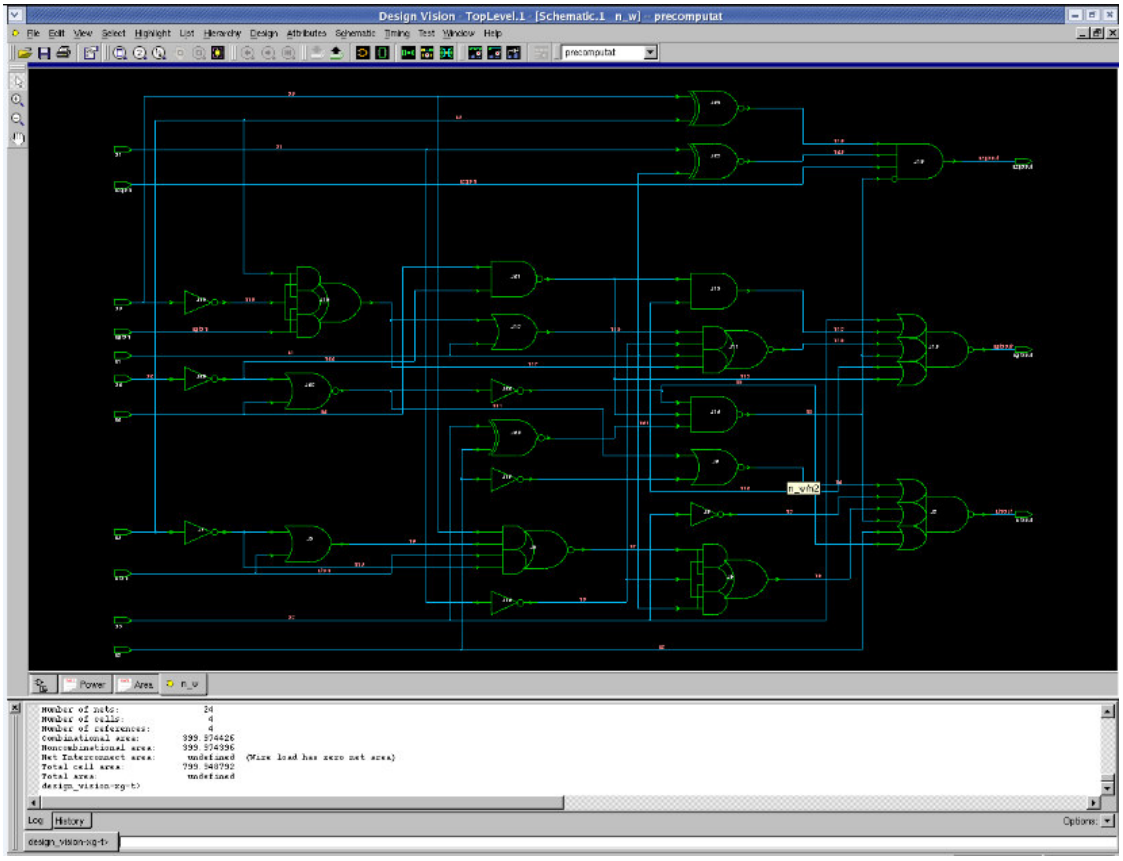


Figure – 4.1.2

4.1.3 Simulated Waveform:

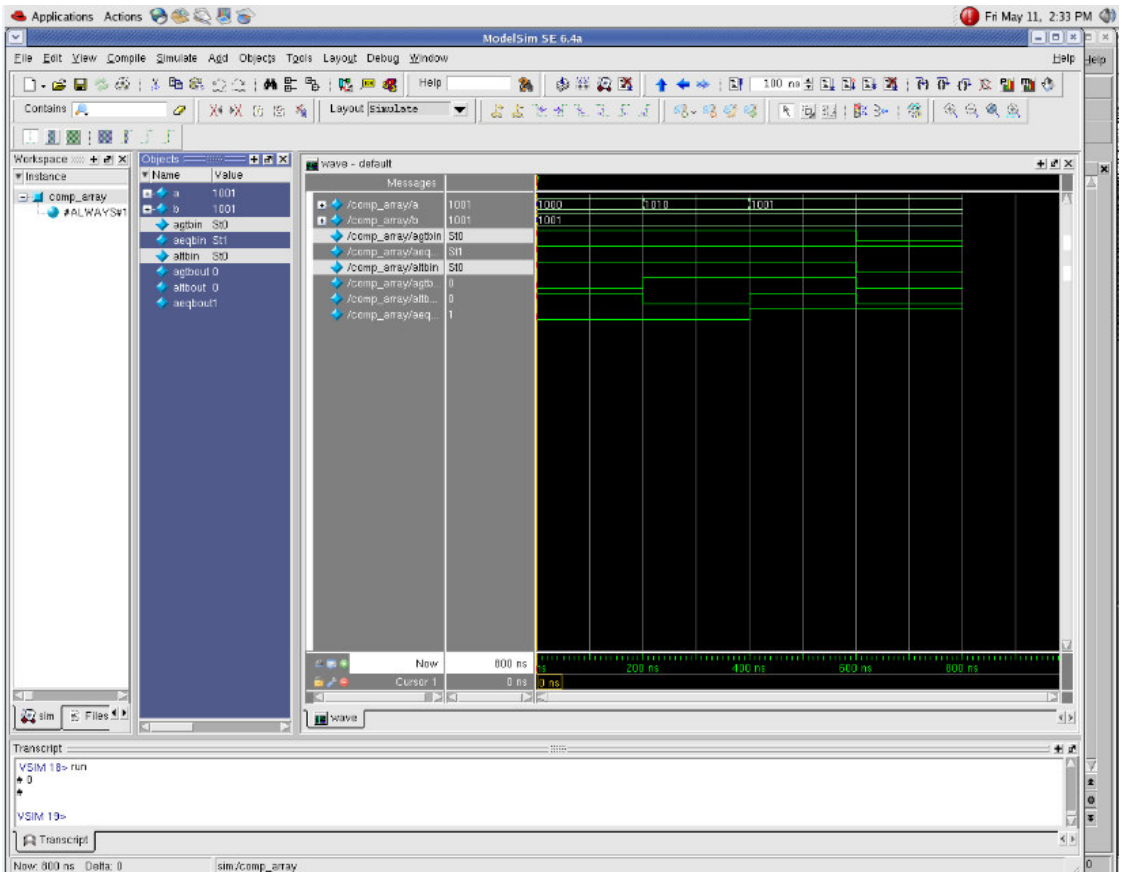


Figure – 4.1.3

4.1.4 Design-Vision Result:

After synthesization of comparator module in Synopsys tool the area and power is coming as 78 units and 164.2940 µw which is shown in table-4.1.4.1 and table-4.1.4.2 respectively.

Table-4.1.4.1

****

Report : area
Design : comparator
Version: Y-2006.06-SP4
Date : Thu May 24 12:35:36 2012

****

Library(s) Used:

fsa0a_c_generic_core_tt1p8v25c(File:/cad/DigitalFDKs/faraday180nm/core180nm/fsa0a_c/2009Q2v2.0/GENERIC_CORE/FrontEnd/synopsys/fsa0a_c_generic_core_tt1p8v25c.db)

Number of ports: 14
Number of nets: 31
Number of cells: 20
Number of references: 13

Combinational area: 78.231995
Noncombinational area: 0.000000
Net Interconnect area: undefined (Wire load has zero net area)

Total cell area: 78.231995
Total area: undefined

***** End Of Report **

Table-4.1.4.2

****
Report : power
-analysis_effort low
Design : comparator
Version: Y-2006.06-SP4
Date : Thu May 24 12:35:15 2012
****

Library(s) Used:

fsa0a_c_generic_core_tt1p8v25c(File:/cad/DigitalFDKs/faraday180nm/core180nm/fsa0a_c/2009Q2v2.0/GENERIC_CORE/FrontEnd/synopsys/fsa0a_c_generic_core_tt1p8v25c.db)

Operating Conditions: TCCOM Library: fsa0a_c_generic_core_tt1p8v25c
Wire Load Model Mode: enclosed

Design	Wire Load Model	Library
comparator	G5K	fsa0a_c_generic_core_tt1p8v25c

Global Operating Voltage = 1.8

Power-specific unit information :
Voltage Units = 1V
Capacitance Units = 1.000000pf
Time Units = 1ns
Dynamic Power Units = 1mW (derived from V,C,T units)
Leakage Power Units = 1pW

Cell Internal Power = 77.7304 uW (49%)
Net Switching Power = 86.5635 uW (51%)

Total Dynamic Power = 164.2940 uW (100%)
Cell Leakage Power = 308.7623 pW

***** End Of Report *****

4.2 Synthesis Using Pre-Computaion Architecture:

4.2.1 RTL of Pre-computation design:

The resultant RTL of Comparator comes as like figure- 4.2.1, where we can see eight input bits ,three cascading inputs ,one clock signal and three outputs. And simulated output is shown in figure-4.2.2.

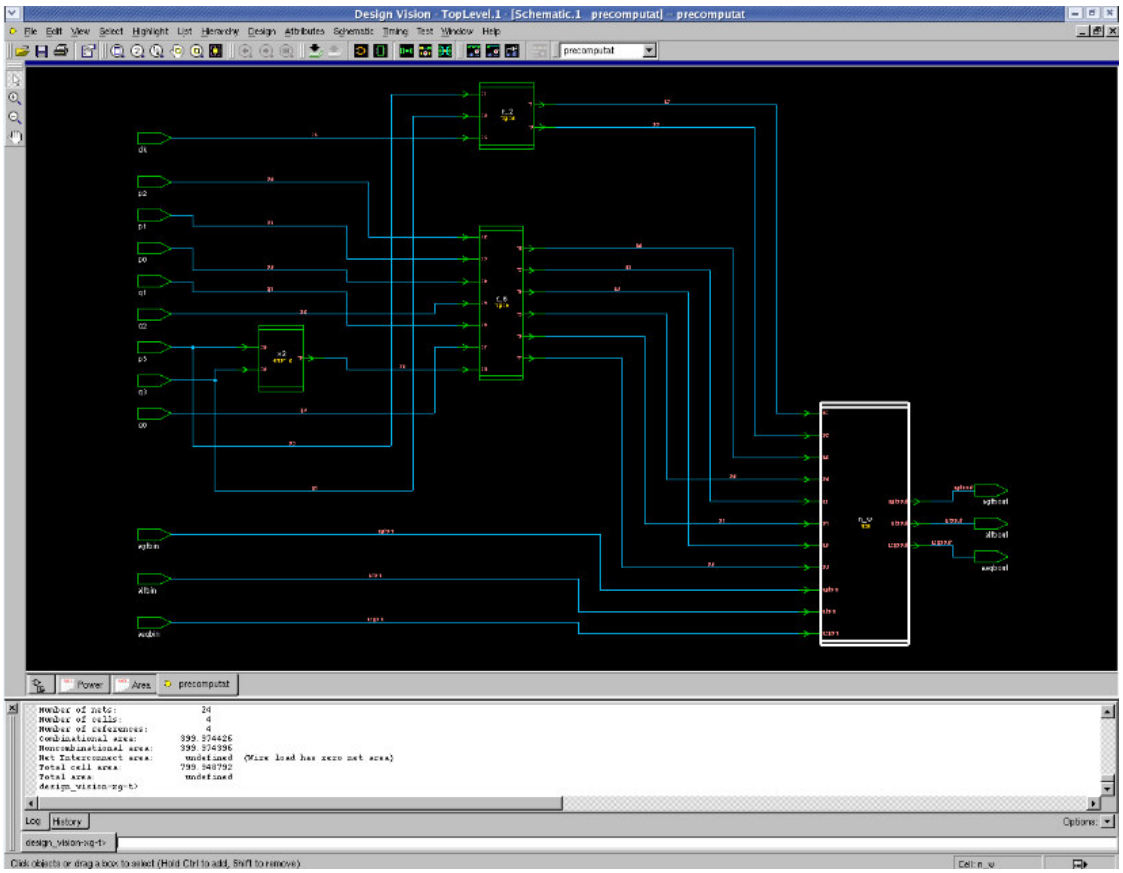


Figure-4.2.1

4.2.2 Simulated waveform:

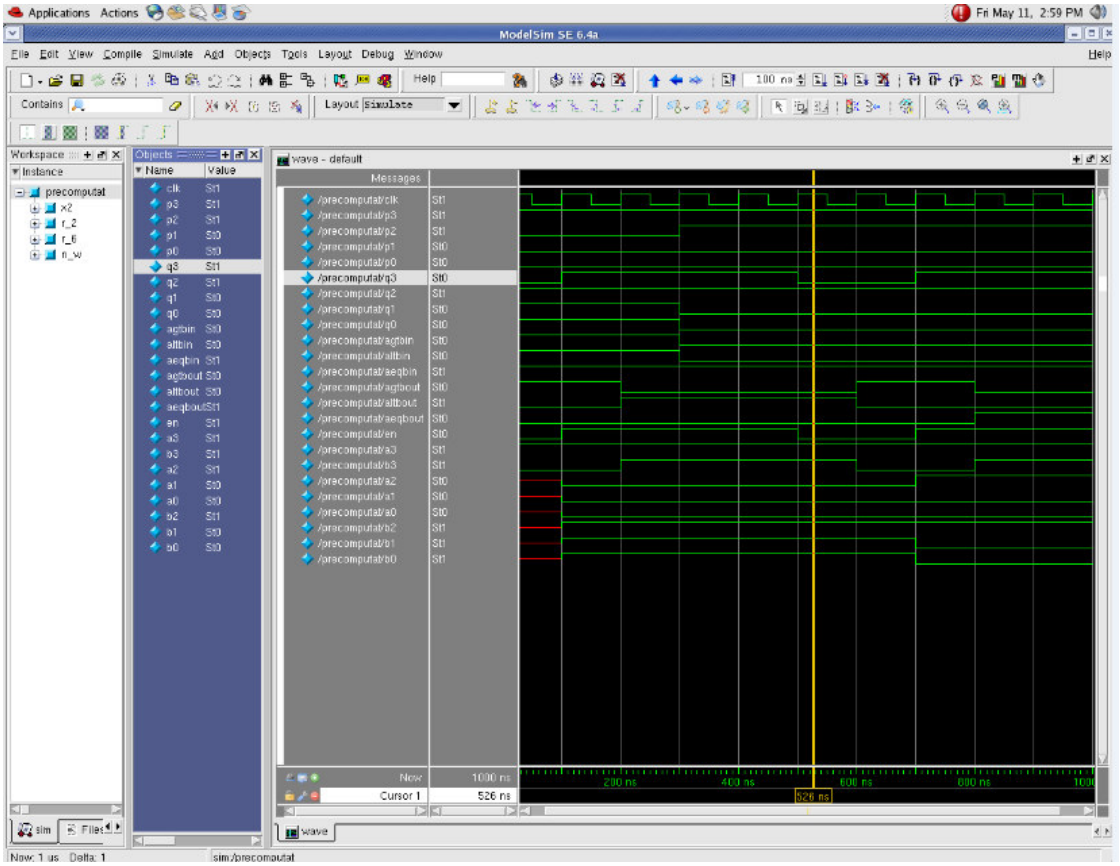


Figure-4.2.2

4.2.3 Design-Vision Result of comparator for pre-computation strategy

Table-4.2.3.1

****

Report : power

-analysis_effort low

Design : precomputat

Version: Y-2006.06-SP4

Date : Thu May 24 12:43:28 2012

```
****************************************
Library(s) Used:

fsa0a_c_generic_core_tt1p8v25c(File:/cad/DigitalFDKs/faraday180nm/core180nm/fsa0a_c/
2009Q2v2.0/GENERIC_CORE/FrontEnd/synopsys/fsa0a_c_generic_core_tt1p8v25c.db)

Operating Conditions: TCCOM   Library: fsa0a_c_generic_core_tt1p8v25c

Wire Load Model Mode: enclosed


Design       Wire Load Model          Library

------------------------------------------------

precomputat          G5K            fsa0a_c_generic_core_tt1p8v25c

xnor_2               enG5K          fsa0a_c_generic_core_tt1p8v25c

regis2               enG5K          fsa0a_c_generic_core_tt1p8v25c

regis6               enG5K          fsa0a_c_generic_core_tt1p8v25c

new                  enG5K          fsa0a_c_generic_core_tt1p8v25c


Global Operating Voltage = 1.8

Power-specific unit information :

   Voltage Units = 1V

   Capacitance Units = 1.000000pf

   Time Units = 1ns

   Dynamic Power Units = 1mW    (derived from V,C,T units)

   Leakage Power Units = Unitless


 Cell Internal Power  =  49.1214 uW   (74%)

 Net Switching Power  =  17.5521 uW   (26%)

                         ---------

Total Dynamic Power    =  66.6735 uW  (100%)
```

***** End Of Report *****

Table-4.2.3.2

```
****************************************
Report : area
Design : precomputat
Version: Y-2006.06-SP4
Date   : Thu May 24 12:44:08 2012
****************************************

Library(s) Used:

fsa0a_c_generic_core_tt1p8v25c(File:/cad/DigitalFDKs/faraday180nm/core180nm/fsa0a_c/
2009Q2v2.0/GENERIC_CORE/FrontEnd/synopsys/fsa0a_c_generic_core_tt1p8v25c.db)

Number of ports:              15
Number of nets:               24
Number of cells:               4
Number of references:          4

Combinational area:       399.974426
Noncombinational area:    399.974396
Net Interconnect area:    undefined  (Wire load has zero net area)

Total cell area:          234.948792
Total area:               undefined

***** End Of Report *****
```

4.3 BDD Evaluvation:

In 4-bit magnitude comparator 78 (16+31+31=78) product terms are present; these product terms are in Boolean expression. Since BDD package accept only PLA format, we make a PLA file with the help of Boolean terms, which has eleven input terms and three output terms. The output of BDD comes as in terms of reduced node count that is 46 as shown in table-4.3.3, previously it was 78 node counts because one product term is consider as one node.

Based on these sum of product terms, PLA of the comparator circuit is as follow, where eleven input (i = 11) terms and three output (o = 3) terms is present.

4.3.1 PLA Format of Comparator:

```
.i 11
.o 3
.ilb aeqbin agtbin altbin a3 a2 a1 a0 b3 b2 b1 b0
.ob aeqbout aqtbout altbout
.p 78
10011111111 100
10011101110 100
10011011101 100
10011001100 100
10010111011 100
10010101010 100
10010011001 100
10010001000 100
10001110111 100
10001100110 100
10001010101 100
10001000100 100
10000110011 100
10000100010 100
10000010001 100
10000000000 100
0001---0--- 010
00011--10-- 010
00001--00-- 010
000111-110- 010
000101-100- 010
000011-010- 010
000001-000- 010
00011111110 010
00011011100 010
00010111010 010
00010011000 010
00001110110 010
00001010100 010
00000110010 010
00000010000 010
01011111111 010
01011101110 010
01011011101 010
01011001100 010
01010111011 010
```

```
01010101010   010
01010011001   010
01010001000   010
01001110111   010
01001100110   010
01001010101   010
01001000100   010
01000110011   010
01000100010   010
01000010001   010
01000000000   010
0000---1---       001
00010--11--       001
00000--01--       001
000110-111-     001
000100-101-     001
000010-011-     001
000000-001-     001
00011101111   001
00011001101   001
00010101011   001
00010001001   001
00001101111   001
00001001101   001
00000101011   001
00000001001   001
00111111111   001
00111101110   001
00111011101   001
00111001100   001
00110111011   001
00110101010   001
00110011001   001
00110001000   001
00101110111   001
00101100110   001
00101010101   001
00101000100   001
00100110011   001
00100100010   001
00100010001   001
00100000000   001
.e
```

After compiling this PLA in BDD package the generated tree is comes out as follow.

4.3.2 Generated Tree :

ROOT: 0

ROOT: 0

ROOT: 1480
[22] 10: 0 1
[23] 10: 1 0
[831] 9: 0 22
[832] 8: 0 831
[833] 7: 0 832
[868] 9: 0 23
[869] 8: 0 868
[870] 7: 0 869
[878] 6: 870 833
[917] 9: 22 0
[918] 8: 0 917
[919] 7: 0 918
[958] 9: 23 0
[959] 8: 0 958
[960] 7: 0 959
[968] 6: 960 919
[969] 5: 968 878
[1011] 8: 831 0
[1012] 7: 0 1011
[1049] 8: 868 0
[1050] 7: 0 1049
[1058] 6: 1050 1012
[1095] 8: 917 0
[1096] 7: 0 1095
[1133] 8: 958 0
[1134] 7: 0 1133
[1142] 6: 1134 1096
[1143] 5: 1142 1058
[1144] 4: 1143 969
[1187] 7: 832 0
[1224] 7: 869 0
[1232] 6: 1224 1187
[1267] 7: 918 0
[1304] 7: 959 0
[1312] 6: 1304 1267
[1313] 5: 1312 1232
[1352] 7: 1011 0
[1388] 7: 1049 0

[1396] 6: 1388 1352
[1430] 7: 1095 0
[1466] 7: 1133 0
[1474] 6: 1466 1430
[1475] 5: 1474 1396
[1476] 4: 1475 1313
[1477] 3: 1476 1144
[1478] 2: 1477 0
[1479] 1: 1478 0
[1480] 0: 0 1479

4.3.3 RESULT

Table-4.3.3

Initial SA=7.943359 Initial NC=78.000000 Initial leakage=1572.260864

Gen:0 TopFitness = 1.000000

1 1 1 : Fitness 1.000000

0 0 1 : Fitness 1.000000

0 1 1 : Fitness 1.000000

0 0 1 : Fitness 1.000000

0 1 0 : Fitness 1.097188

0 0 0 : Fitness 1.097188

0 0 0 : Fitness 1.097188

1 1 0 : Fitness 1.097188

1153730649 0 0 : Fitness 1.097188

0 0 0 : Fitness 1.097188

Final TopFitness = 1.000000

1 1 1 Final SA=7.943359 Final NC=46.000000 Final leakage=1572.260864

4.3.4 2x1 multiplexer :

After compiling BDD package the final node count is coming 46, which is shown in table-4.3.3. In BDD each node is represented by a single 2x1 multiplexer. In 2x1 multiplexer

when select line s is set to zero then output y is equal to a and when s is set to one then output is b which is shown as below and is expressed logically as y= s'a+sb.

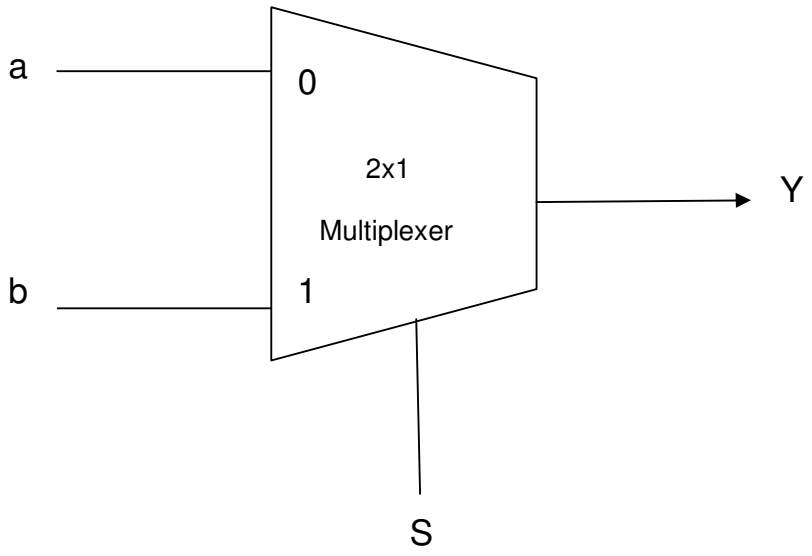


Figure-4.3.4

After synthesization of a 2x1 multiplexer in Synopsys tool, the area and power is coming 14.87 units and 762.3125 nW respectively which is shown in table-4.3.7.1 and table-4.3.7.2 respectively. The RTL of 2x1 multiplexer is shown in figure-4.3.5.A 2x1 multiplexer used to select one of sources of data to transmit on a bus.

4.3.5 Simulated waveform:

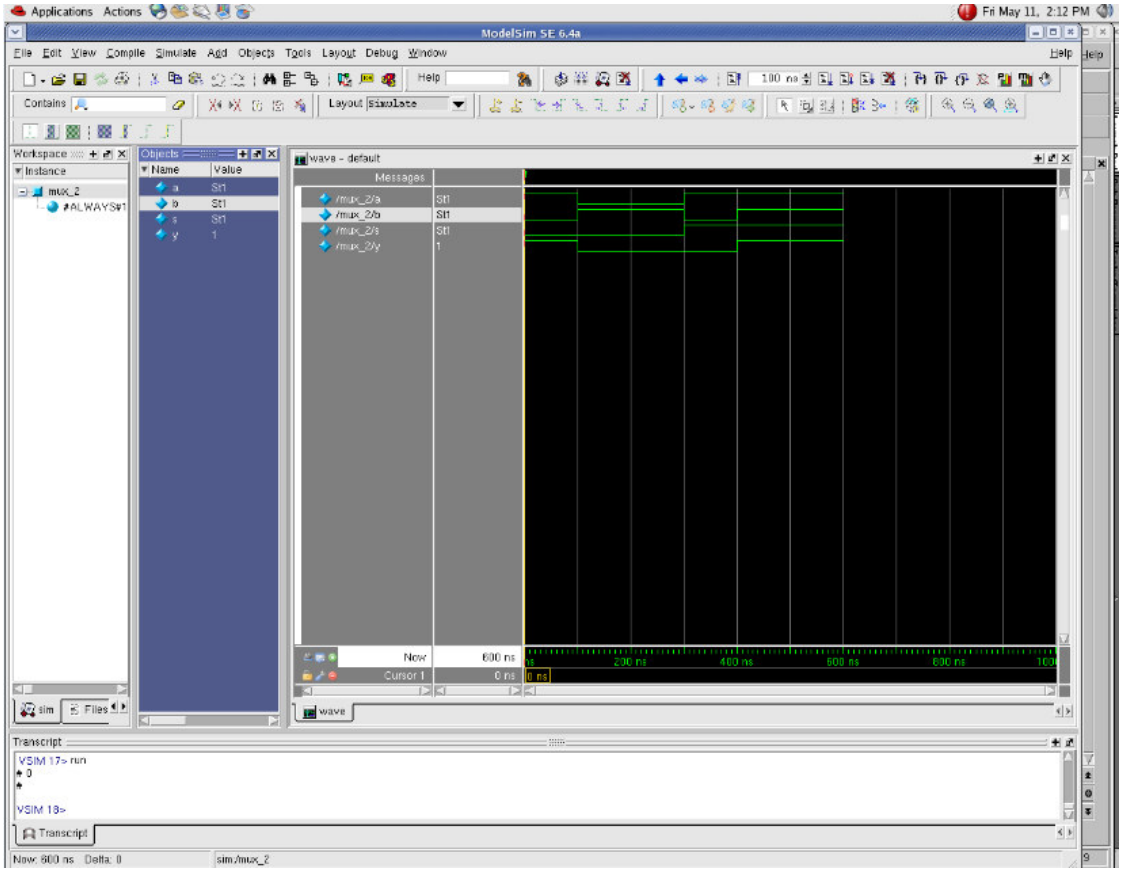


Figure-4.3.5

4.3.6 Design-Vision Result of Mux_2:

Table-4.3.6.1

****

Report : power

-analysis_effort low

Design : mux_2

Version: Y-2006.06-SP4

Date : Thu May 24 12:18:01 2012

****

Library(s) Used:

fsa0a_c_generic_core_tt1p8v25c(File:/cad/DigitalFDKs/faraday180nm/core180nm/fsa0a_c/2009Q2v2.0/GENERIC_CORE/FrontEnd/synopsys/fsa0a_c_generic_core_tt1p8v25c.db)

Operating Conditions: TCCOM Library: fsa0a_c_generic_core_tt1p8v25c

Wire Load Model Mode: enclosed

Design	Wire Load Model	Library
mux_2	G5K	fsa0a_c_generic_core_tt1p8v25c

Global Operating Voltage = 1.8

Power-specific unit information :

Voltage Units = 1V

Capacitance Units = 1.000000pf

Time Units = 1ns

Dynamic Power Units = 1mW (derived from V,C,T units)

Leakage Power Units = Unitless

Cell Internal Power = 765.1176 nW (87%)

Net Switching Power = 97.1948 nW (13%)

Total Dynamic Power = 762.3125 nW (100%)

***** End Of Report **

Table-4.3.6.2

****

Report : area
Design : mux_2
Version: Y-2006.06-SP4

Date : Thu May 24 12:24:56 2012
****

Library(s) Used:

fsa0a_c_generic_core_tt1p8v25c(File:/cad/DigitalFDKs/faraday180nm/core180nm/fsa0a_c/2009Q2v2.0/GENERIC_CORE/FrontEnd/synopsys/fsa0a_c_generic_core_tt1p8v25c.db)

Number of ports: 4
Number of nets: 4
Number of cells: 1
Number of references: 1

Combinational area: 14.873600
Noncombinational area: 0.000000
Net Interconnect area: undefined (Wire load has zero net area)

Total cell area: 14.873600
Total area: undefined

***** End Of Report **

Since we have 46 node count, we multiply area and power by 46 times so that we can find out the total area as well as power in implementation of comparator in Synopsys tool. Thus total power is coming as follow

Total power = 46* 762.3125 nw
= 35066.375 nw
= 35.0664 µw

4.4 Comparison :

Table-4.4

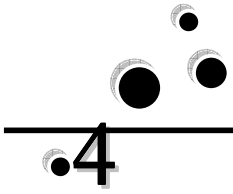
	BDD	Synopsys	Pre-computation
POWER	46*762.3125 nw =35.0664 µw	164.29 µw	66.6735 µw

When we started, implementation of 4-bit magnitude comparator in BDD, then the total product terms was 78 that means 78 node count , but with the help of BDD package tool it reduced to 46 node count as shown in table-4.3.3. Now one node is represented by a 2x1 multiplexer. After synthesizing 2x1 multiplexer in Synopsys tool, the power required for it is 762.3125 nw as shown in table-2.3.7.1. Since we have total 46 nodes so total power taken by 4-bit comparator is 46*762.3125 nw which is equal to 35.0664 µw.

When we synthesized 4-bit magnitude comparator in Synopsys tool then the power comes as 164.29 µw which is shown in table-4.1.4.2. After applying pre-computation technique in comparator then the total power comes as 66.6735 µw shown in table-4.3.2.1, which is less then compare to without applying pre-computation technique in comparator. But when we compare all the three ways then we can conclude that, implementation of 4-bit magnitude comparator through BDD is the best way for low power aspect.

Chapter-4

Conclusion And Future Work



Symbolic model checking has proven to be a powerful paradigm to automatically verify real world applications. In this thesis a BDD based optimization considering the output phase has been presented and found to achieve our objective of power minimization. The result found to be comparable to other schemes of optimization, such as pre-computation technique.In this paper I have calculated a BDD based simulation of a 4-bit magnitude comparator and compare with pre-computation strategy and find that the significant improvement in term of power.

By optimizing the multiplexers we can reduce further area requirement, which can be taken as future work. Also we can reduce the of count of multiplexer by using new versions of BDD Packages in which BDD and Dynamic variable reordering algorithms are design to optimize the count of multiplexers.

Reference

5

[1] S.Chaudhury and S.Chattopadhyay "Output phase assignment for area and power optimization in multi-level multi-output combinational logic circuits".

[2] B. Yang. "Optimizing Model Checking based on BDD Characterization." School of Computer Science –Carnegie Mellon University, May 1999. Available as researchreport CMU-CS-99-129.

[3] S. B. Akers, "Binary Decision Diagram," IEEE Trans. Computers, Vol. 27, 1978.

[4] K. S. Brace and R. L. Rudell and R. E. Bryant, "Efficient Implementation of a BDD Package," Design Automation Conference, 1990.

[5] P.W.C. Prasad, and A. K. Singh, "An Efficient Method for Minimization of Binary Decision Diagrams," 3rd International Conference on Advances in Strategic Technologies (ICAST), pp. 683-688, 2003.

[6] Mazhar Alidina, Jose Monteiro, Srinivas Devadas, "Pre-computation based sequential logic optimization for low power", IEEE transaction on VLSI system, No.4 , DECEMBER 1994.

[7] ONDREJ LHOTAK and LAURIE HENDREN, " Evaluating the Benefits of Context-Sensitive Points-to Analysis Using a BDD-Based Implementation" Sep 2008, ACM, Proceedings of the 15th International Conference on Compiler Construction. (page 19)

[8] Robert Wille and Rolf Drechsler, "BDD based synthesis of reversible logic for large function", Design automation conference,2009, DAC'09.46th ACM/IEEE.

[9] Fei Sun and Yinshui Xia, "BDD based detection algorithm for xor-type logic", 2008 11th IEEE international conference on communication technology proceedings .

[10] M.Morris Mano "digital logic and computer design" PRENTICE-HALL, INC., ENGLEWOOD CLIFFS, 2009

[11] A. Anand Kumar "Fundamentals of Digital Design circuits", 2nd edition, PHI Learning private limited-2009

[12] Nagayama, S., A. Mishchenko, T. Sasao and J.T. Butler, 2003. "Minimization of average path length in BDDs by variable reordering". Intl. Workshop on Logic and Synthesis.

[13] A. Raghunathan and N. Jha " Behavioral synthesis for low power." In Proceedings of the International Conference on Computer Design, pages 318–322, Boston, MA, Oct. 1994.

[14] Chandrakasan, A.P., and Brodersen, R.W., "Low Power Digital CMOS Design," Kluwer Academic Publishers, 1995.

[15] R. Rudell, "*Dynamic variable ordering for ordered binary decision diagrams,*" *International Conference on Computer-Aided Desing*, pp. 42–47, 1993.

[16] Shih-Chieh Chang, David Ihsin Cheng and Malgorzata Marek-Sadowska, "Minimizing ROBDD size of incompletely specified multiple output function", European Design and Test Conference, 1994.

[17] M. Horowitz, T. Indermaur, and R. Gonzalez, "Low-power digital design," in Proc. Symp. Low Power Electronics, pp. 8–11, Oct. 1994

[18] A. Raghunathan and N. Jha " Behavioral synthesis for low power." In Proceedings of the International Conference on Computer Design, pages 318–322, Boston, MA, Oct. 1994.

[19] Chandrakasan, A.P., and Brodersen, R.W., "Low Power Digital CMOS Design," Kluwer Academic Publishers, 1995

[20] K. Roy and S. Prasad. "SYCLOP: Synthesis of CMOS Logic for Low Power Applications." In Proceedings of the Int'l Conference onComputer Design: VLSI in Computers and Processors, pages 464–467, October 1992.

[21] P Landman and J. Rabaey." Power estimation for high level synthesis". In Proceedings of the European Design utomation Conference, pages 361–366, Paris, Feb. 1993.

[22] R. Mehra and J. Rabaey, "Behavioral level power estimation and exploration," in Proc. Int. Wkshp. Low Power Design, pp. 197–202,Apr. 1994.

[23] H. Andreas, R. Drechsler and B. Becker, "MORE: An Alternative Implementation of BDD-Packages by Multi- Operand Synthesis," IEEE European Design Automation Conference, pp. 164-169, 1996.

[24] R. Rudell and A. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization," IEEE Transactions on CAD/ICAS, vol. CAD-6, n. 5, pp. 727-750, 1987.

[25] R. Drechsler, J. Shi, and G. Fey, Synthesis of fully testable circuits from BDDs, IEEE Transaction on Computer-Aided Design, Vol. 23, No. 3, pp. 440-443, March 2004.

[26] W. Gunther and R. Drechsler, ACTion: Combining logic synthesis and technology mapping for MUX-based FPGAs, J.Systems Architecture, Vol. 46(14) 2000, pp. 1321-1334.

[27] S. V. A. Campos. Symbolic model checking in practice. In XII Symposium on Integrated Circuits and System Design (SBCCI'2000), pages 98–101. Brazilian Computer Society, Computer Society Press, 1999.

[28] A. Gupta and P. Ashar, "Integrating a Boolean satisfability checker and BDDs for combinational equivalence checking," Proc. Of the International Conference on VLSI Design, 1998.

[29] A.Raghunathan andN. Jha. "ILP formulation for lowpower based on minimizing switched capacitance during data path allocation." In Proceedings of the International Symposium on Circuits & Systems, 1995.

[30] D. Agrawal, "Low Power Design by Hazard Filtering," 10th Int'l Conf. VLSI Design, Jan. 1997, pp. 193-197