ANALYSIS OF IP BASED IMPLEMENTATIONS OF ADDERS AND MULTIPLIERS IN SUBMICRON AND DEEP SUBMICRON TECHNOLOGIES

By

VIJAYA CHANDRA KURAPATI

Bachelor of Technology in Electrical and Electronics

Engineering

Jawaharlal Nehru Technological University

Hyderabad, Andhra Pradesh

2006

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2008

ANALYSIS OF IP BASED IMPLEMENTATIONS OF

ADDERS AND MULTIPLIERS IN SUBMICRON AND

DEEP SUBMICRON TECHNOLOGIES

Thesis Approved:

Dr. James E. Stine, Jr.

Thesis Adviser

Dr. Louis G. Johnson

Dr. Sohum Sohoni

Dr. A. Gordon Emslie

Dean of the Graduate College

ACKNOWLEDGMENTS

I am very grateful to my adviser Dr. James E. Stine, Jr. who has allowed me to work under him. His thorough support and exemplary guidance all through my work have helped me a lot. Also, I would like to thank Dr. Louis G. Johnson and Dr. Sohum Sohoni for being a part of my thesis committee.

Finally, I would like to thank my family who has encouraged me to pursue my Master’s degree. I would also like to thank my friends and relations for their support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER I

INTRODUCTION

With the continuous technological advancements being achieved in the semi-conductor industry, the market has seen increased demand for factors such as portability, performance and high functional integration in digital devices. This increased demand has made scaling of MOS transistors inevitable. Continuous decrease in feature size of the MOS transistors has resulted in decreased sizes of CMOS gates and enabling highly dense packaging of integrated circuits and thus increasing wiring densities.

The trend of continuously scaling the device size and increasing chip densities has not only resulted in high design complexity but also has caused the design time to increase. Over the past decade for this very reason the concepts of design reuse and Intellectual Property (IP) have been adopted in the design of digital circuits. This adoption as a mainstream design practice has changed the approach of designers and has resulted in increased design productivity.

With IP being the popular choice of the designers so as to reduce the design time and with designs being implemented in deep submicron technologies, the designers are facing new set of challenges. In the deep submicron regime where static power dissipation is no more ignorable, the focus of the designers has now shifted towards minimizing the value of average power consumed by the circuit. The figure below shows a comparison of dynamic power and leakage power in different technologies

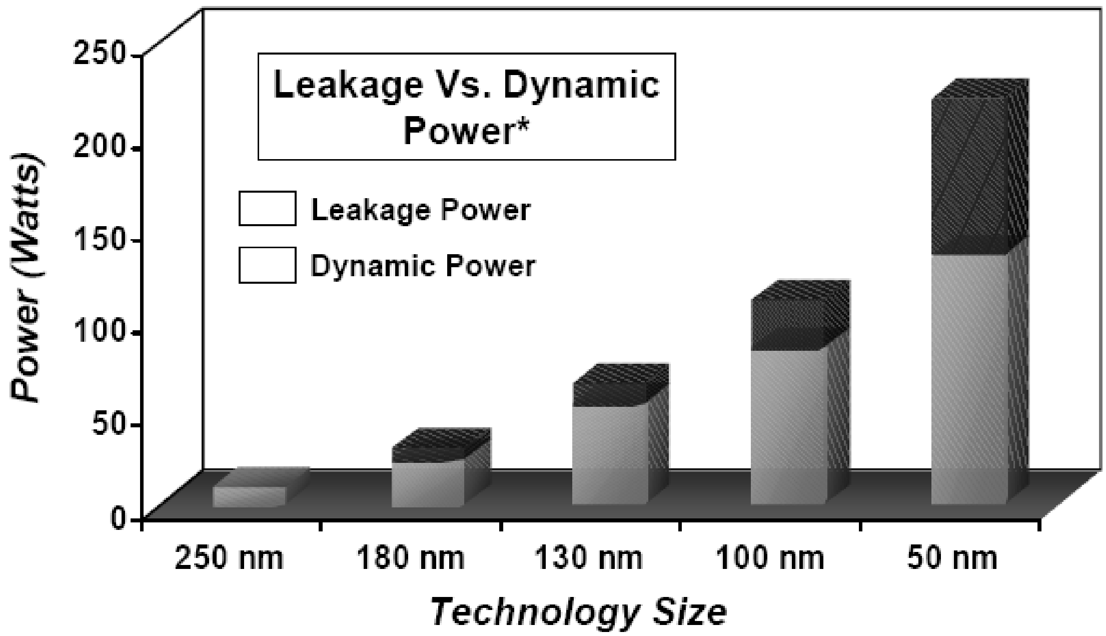


Figure 1.1: Power Consumed in various Technologies

1.1 Motivation

The core of any kind of processor is its data path. Data path is the one of the crucial component which decides the key parameters such as the clock frequency, area and power dissipation of the design. Adders and multipliers are the main components in the data path and they are of major concerns for the designers of the data path. The use of IP being popular for designing large systems, it is of more importance to investigate the performance of various adder and multiplier implementations that are available with the commercially available IP. This thesis focuses on analyzing adders and multipliers of various implementations that are available with Synopsys DesignWare IP.

1.2 Overview of various Adder Architectures

Adders are one of the key components of any data path. As any component in VLSI design, the choice of adder architecture is constrained by the important factors of area, speed and power. Among the various architectures of adders available few of them are briefly described in this section.

1.2.1 Ripple Carry Adder

Ripple carry adders are one of the most simple adder architectures available. A n-bit ripple carry adder is made of up a collection of n number of individual full adder cells. These adders are simple in design and also they occupy less area. But they are constrained in their performance capabilities. For the modern day designs where high speed of operation is required, these adders fall short by a large extent as the delay through the adder chain to produce the output is very large. Hence, these adders are not very popular to be implemented in the modern day designs. Because of their simplicity in design there are certain circuit implications which can be efficiently implemented using ripple carry adders. The figure below gives a generalized structure for a ripple carry adders [6].

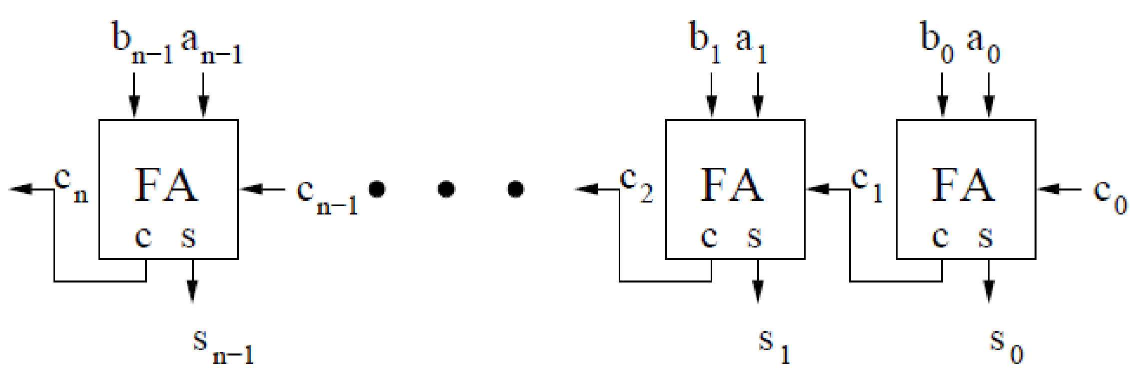


Figure 1.2: Generalized Structure of a n-bit Ripple Carry Adder

1.2.2 Carry Look-Ahead Adders

As seen earlier ripple carry adders are limited in their performance capabilities. So, adders with improved performance are required. Carry look-ahead adders are one such solution. As the name suggests, in carry look-ahead adders the carry chain is generated ahead of time utilizing all of the inputs to improve the addition operation. This is achieved at the expense of increased area and power in the form of increased number of gates. The carries are precomputed using the generate and propagate signals which are computed using the below equations [7]

$$G_i = A_i \,.\, B_i$$

$$P_i = A_i + B_i$$

Where A and B are inputs and k represents the i[th] bit

The sum and carry bits in terms of the generate and propagate signals are given by the below equations [7]

$$S_i = C_{i-1} \oplus P_i$$

$$P_i = A_i \oplus B_i$$

$$C_{i+1} = G_i + P_i.C_i$$

The figure below shows a 4-bit carry look-ahead adder as an example [6]

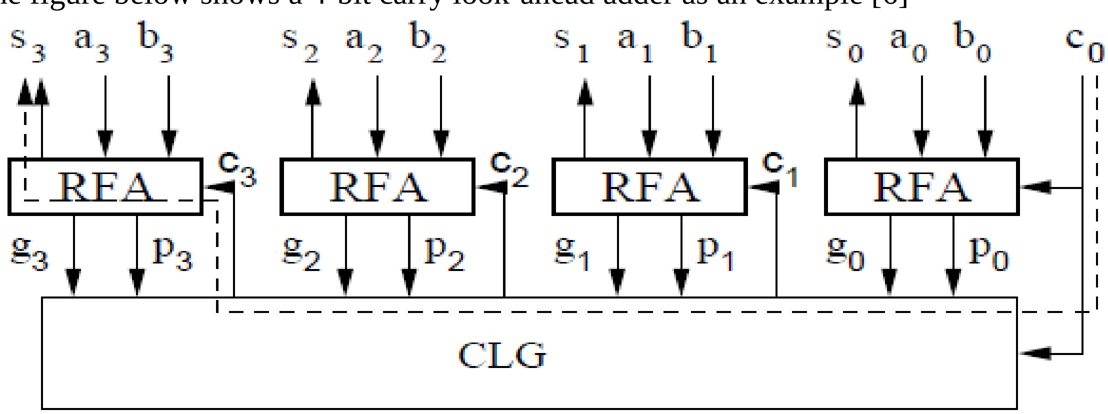


Figure 1.3: A 4-bit Carry Look-Ahead Adder Implementation using a Carry Look-Ahead Generator (CLG)

1.2.3 Carry Select Adders

Carry select adders are one of the other popular architectures which show improved performance over ripple carry adders. As in ripple carry adders they are popular for their regular layout structure. These adders basically consist of blocks where each block executes two additions. One assumes that the input carry is ‘1’ and the other assumes that the input carry is ‘0’. The input carry signal ‘0’ generates a block generate signal and the input carry signal ‘1’ generates a block propagate signal which are used to produce the carry out signal for the subsequent block which selects the appropriate set of sum bits.

The figure below shows a 16-bit carry select adder implementation [6]

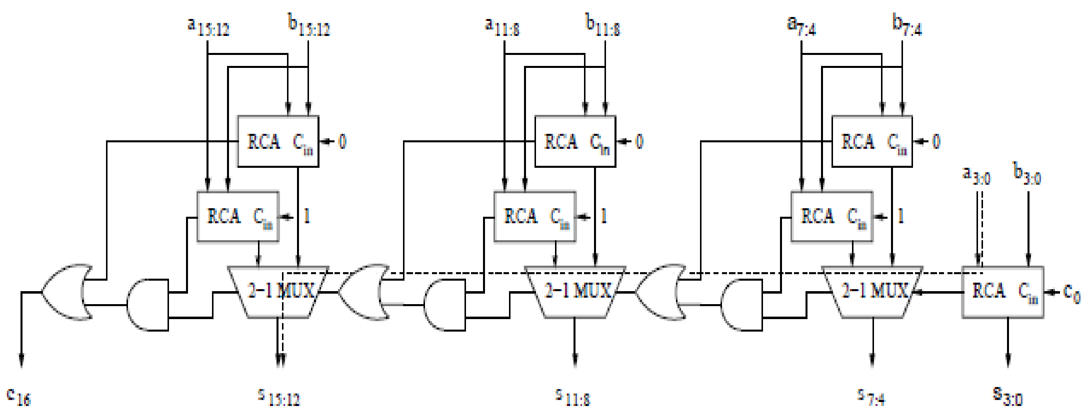


Figure 1.4: A 16-bit Carry Select Adder Implementation with 4 blocks

1.2.4 Conditional Sum Adders

Conditional sum adder architecture is supposed to be the fastest adder theoretically. These are very similar to the carry select adders in concept. The idea lies in precomputing the results for the addition assuming input carry to be ‘0’ and other assuming input carry to be ‘1’ and selecting the proper results based upon the actual value of input carry signal using a multiplexer control. The figure below shows a 4-bit conditional sum adder implementation [10]

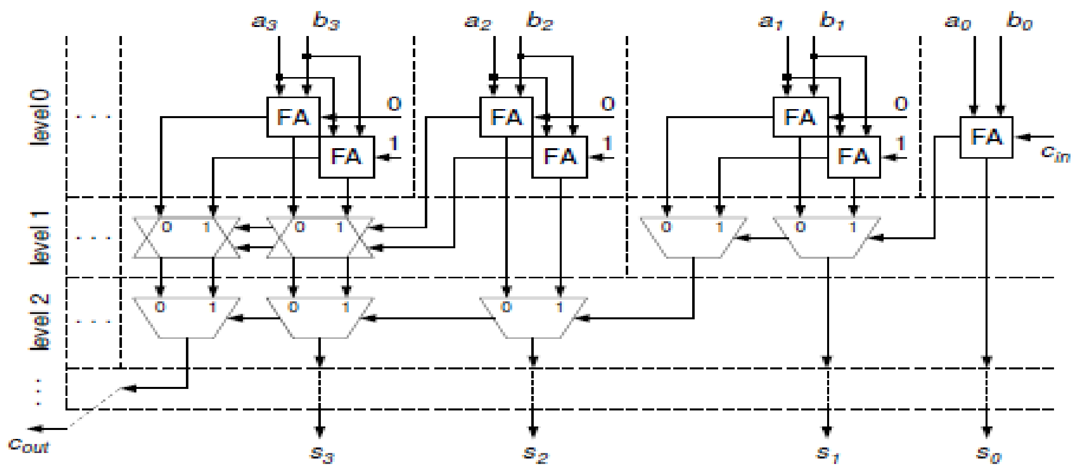


Figure 1.5: A 4-bit Conditional Sum Adder Implementation

1.3 Overview of Multiplier Architectures

Multipliers are the key components in the datapath which consume huge amount of power and occupy large areas. In multipliers, the power dissipation is huge owing to the power dissipated in the large number of gates which are a part of the multiplier structure. Adder blocks form the building blocks for various multiplier structures. In general, any multiplication operation can be divided into three steps [6]

1) Partial Product Generation – With the inputs available generating partial products utilizing a collection of gates.
2) Partial Product Reduction – Utilizing the adders to reduce the partial products to sum and carry vectors for further computation.
3) Final Carry-Propagate Addition – Adding sum and carry vectors to produce the final result.

A multiplication operation performed on an M-bit number and an N-bit number results in a result with (M + N) number of bits. The figure below shows a basic scheme for an unsigned M x N-bit multiplier [12]

				a_{M-1}	a_{M-2}	$\dots$	a_1	a_0	Multiplicand
				b_{N-1}	b_{N-2}	$\dots$	b_1	b_0	Multiplier
				$a_{M-1}b_0$	$a_{M-2}b_0$	$\dots$	a_1b_0	a_0b_0	
			$a_{M-1}b_1$	$a_{M-2}b_1$	$\dots$	a_1b_1	a_0b_1		
				$\vdots$					Partial Products
	$a_{M-1}b_{N-1}$	$a_{M-2}b_{N-1}$	$\dots$	a_1b_{N-1}	a_0b_{N-1}				
p_{N+M-1}	p_{N+M-2}			$\dots$			p_1	p_0	Product

Figure 1.6: Basic Scheme for an M x N-bit Multiplier

In general, multipliers can be classified in three broad categories [12]

1) Sequential Multipliers – in these types of multipliers, the partial products are generated sequentially and these are added to the previously accumulated sum. The shift and add multipliers are an example of sequential multipliers. The delay of sequential multipliers is very large and so hardly put into use in modern designs.

2) Parallel Multipliers – in these types of multipliers, the partial products are generated in parallel and multi operand fast adders are used for accumulation of the product.

3) Array Multipliers – these types of multipliers iteratively utilize identical cells that generate new partial products and accumulate them simultaneously.

Among the various available multiplier architectures few of them are briefly described below

1.3.1 Carry-Save Array Multiplier

Carry-save array multiplier is one of the simplest available architecture in multipliers. This architecture is very similar to the traditional human method of performing multiplication operations. Carry-save array multipliers show simplicity in layout and hence are preferred. This multiplier makes use of modified half adder (MHA) and modified full adder (MFA) as the building block. A MHA consists of an AND gate that produces the partial product bit and a half adder (HA). The MHA adds the partial product bit from the AND gate with the partial product bit from the previous row. A MFA consists of an AND gate that produces a partial product bit and a full adder (FA) that adds the partial product bit with the sum and carry bits from the previous row. In general, carry-save array multiplier has a complexity proportional to the order of n^2 for area and order of n for delay associated with the product generation [6]. The figure below shows a carry-save multiplier used for multiplication of two 4-bit numbers [6]

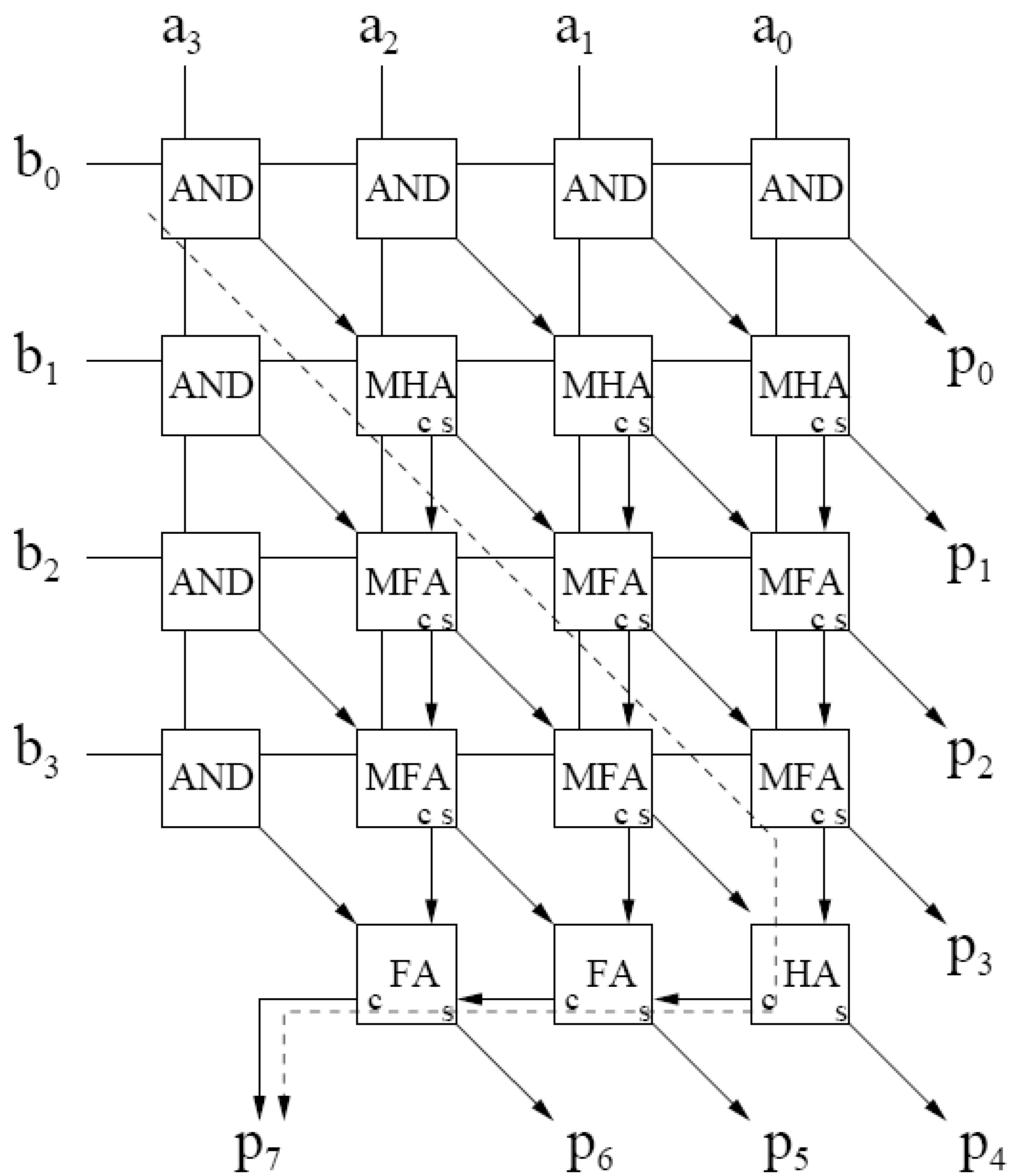


Figure 1.7: 4 x 4 bit Carry-Save Array Multiplier

1.3.2 Wallace Tree Multiplier

C S Wallace introduced this multiplier architecture where the partial products were summed using a tree of carry-save adders. Wallace tree adders follow a three step technique to multiply two numbers [8].

1) Initially the bit products are formed

2) Using the tree of carry-save adders the bit product is reduced to a two row matrix

3) To produce the product these two rows are summed using fast carry propagate adders

In Wallace tree multipliers, the rows are grouped into sets of three and the rows which do not form a group are transferred to the next reduction stage. The height of the matrix in the j^{th} reduction stage is where w_j is defined by the following recursive equations [6]

$$w_0 = n$$

$$w_{j+1} = 2 \cdot \lfloor \frac{w_j}{3} \rfloor + (w_j \bmod 3)$$

Utilizing the above equations the intermediate matrix heights are determined based on the bit size of the operands. A Wallace tree multiplier yields a delay proportional to the logarithm of operand size n which is of the order of $\log_{3/2} n$ [4]. The structure of the Wallace tree multiplier makes it difficult for custom layout when compared to the array multipliers. The figure below shows dot representation for a Wallace tree multiplier which computes the product for two 4-bit numbers [6].

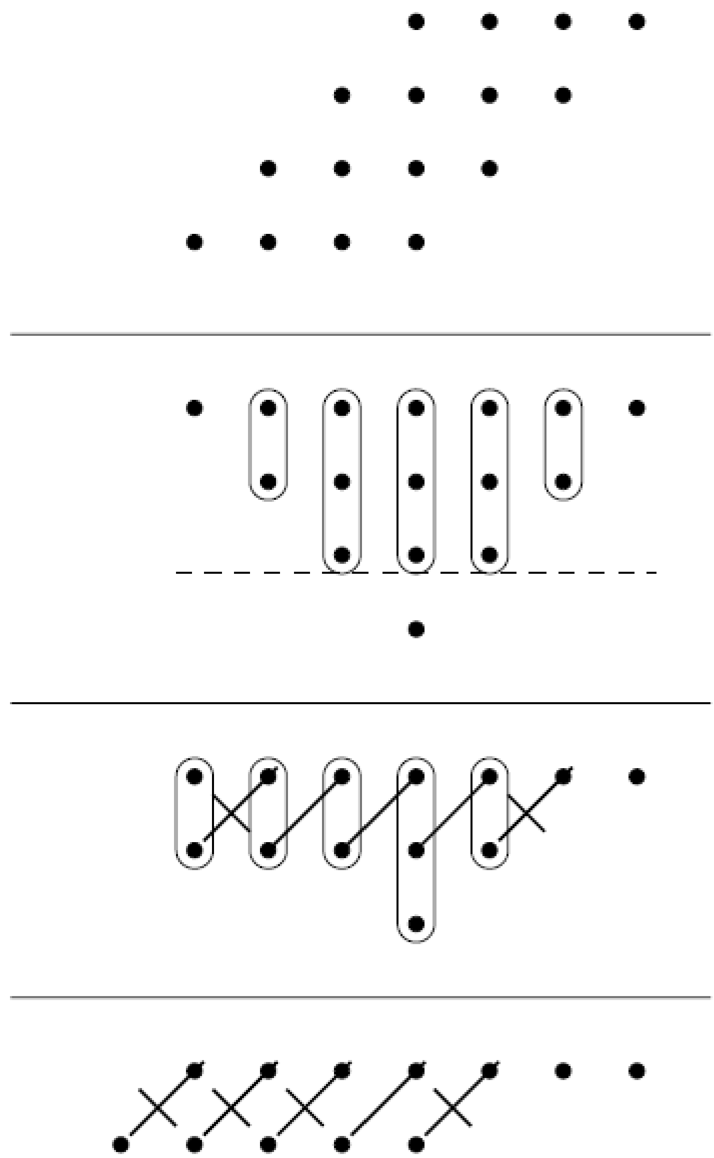


Figure 1.8: Dot representation of 4-bit x 4-bit Wallace Tree Multiplier

1.3.3 Booth's Algorithm

Booth's algorithm is one of the best known algorithms for implementing multipliers. Sometimes this algorithm is also referred to as Booth's Recoding Algorithm. This algorithm tries to minimize the number of partial products generated during multiplication. This is achieved utilizing the fact that multiplication with bit '0' requires only a shift operation to be performed on the product. This algorithm can be utilized conveniently to perform signed magnitude multiplication and 2's complement multiplication of numbers. But, care has to be taken in the case of 2's complement multiplication for the sign bit. Booth's recoding is usually done in two steps of encoding and selection. The process of encoding involves selection of certain number of bits of the multiplier and determines the type of operation to be performed on the multiplicand. Then the selection of the partial products required for the operation is made. Booth's algorithm has been implemented in two variations. One, Radix-2 Booth Recoding where in two bits are examined to define the operation. Two, Radix-4 Booth Recoding where in three bits are examined to define the operation. The table below shows the Radix-4 Booth Recoding Algorithm [8]

$x_{i+2}x_{i+1}x_i$	Add to partial product
000	+0Y
001	+1Y
010	+1Y
011	+2Y
100	-2Y
101	-1Y
110	-1Y
111	-0Y

Table 1: Radix-4 Booth Recoding Algorithm

The benefit of generation of less number of partial products in the Booth's algorithm comes at an expense of increased hardware.

1.4 Pipelining

Pipelining is a popular technique that has been employed in the design industry over several years. This is an architectural choice employed by designers to reduce power.

Over the years, systems have been pipelined to improve performance. Arithmetic circuits such as adders and multipliers which are a key part of the system's datapath can be pipelined to improve performance. The key terms associated with any pipelined systems are

1) Latency - The delay from when an input is established until the output associated with that input becomes valid.
2) Throughput: The rate at which inputs or outputs are processed is available.

Pipelining as such does not reduce power by itself but reduces the critical path delay by inserting registers between combinational logic. The clock signal to registers has high activity thus contributing to dynamic power. By pipelining glitches can be prevented from propagating over register boundaries but logic activity is unchanged. The timing slack from pipelining can be used for voltage scaling and gate downsizing to achieve significant power savings.

The figure below shows the advantage of pipelining two logic blocks connected in series.

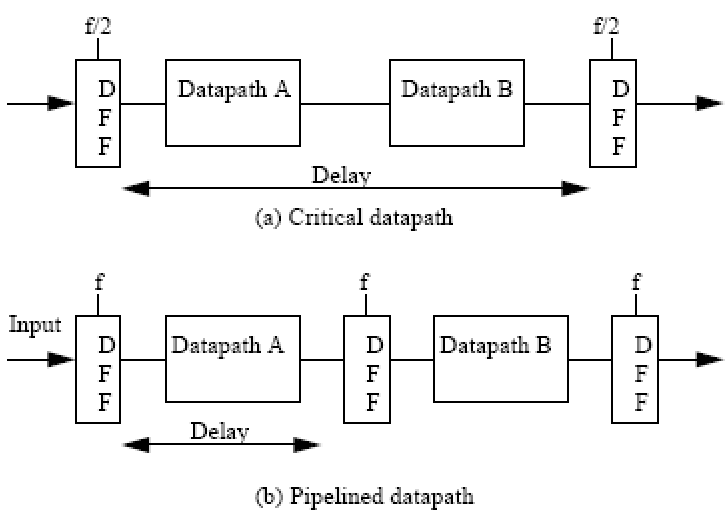


Figure 1.9: Pipelining

The figure below shows how the process of pipelining increases throughput

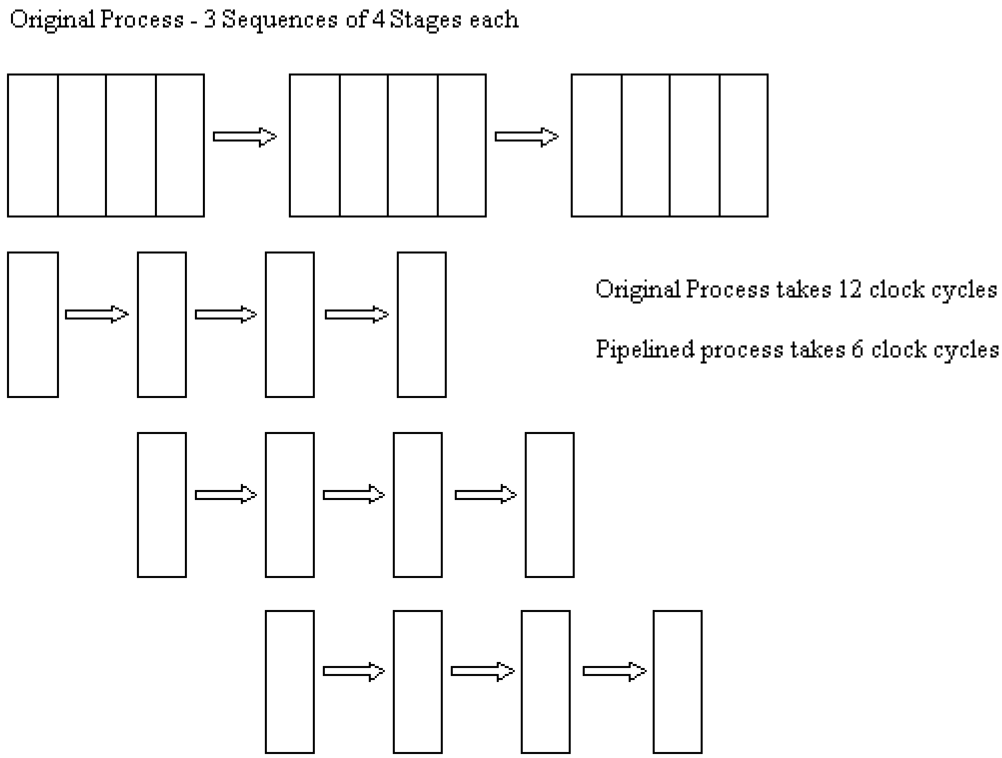


Figure 1.10: Pipelining Process

CHAPTER II

REVIEW OF LITERATURE

The design of high performance arithmetic circuits has always attracted ASIC processor designers. There have been many works that try to improve the performance of these circuits in terms of power consumption or delay associated. This chapter focuses on few of such literary works that had been done previously in this field.

In [11], Sean Kao *et al* presented the impact of design choices on power and performance of domino CMOS adders through the use of an optimization tool to confirm the results. Also, they came up with a 64-bit fast and energy efficient adder design utilizing sparse radix-4 Ling adder topology. The design was implemented in a general purpose 90nm CMOS technology and the adder performed 64-bit addition in 240ps while consuming a power of 260mW at a supply voltage of 1V and room temperature. In [12], Keivan Navi and Omid Kavehei came up with a new 1-bit full adder cell design style called “Bridge”. This full adder cell was supposed to consume low power and offer high performance. Simulations were performed using HSPICE simulator in 90nm standard CMOS technology and the results of these simulations were compared in terms of power, delay and power-delay product and were found to be superior to a conventional CMOS 1-bit full adder cell implemented in the same technology. In [13], Lan Wei as a part of his

Master's thesis studied the effect of pipelining on various adder structures. He studied the effect of pipelining on four different adder structures at the physical implementation level and came up with an optimal adder structure. He implemented the adder structures using a 0.35 um technology based standard library at a nominal supply voltage of 3.3V.

In [14], Sheng Sun and Carl Sechen made an extensive study of carry look-ahead (CLA) adders and carry-select adders with a wide range of trade-offs in logic levels, fan-out's and wiring complexity. They also proposed sparse CLA adder architectures based on buffering techniques to reduce logic redundancy and improve energy efficiency. All the designs were implemented using an energy-delay layout optimization flow with full RC extraction. In [15], Vojin G. Oklobdzija and Bart R. Zeydel presented energy-delay estimation (EDE) method which extends logical effort (LE) and its application to the analysis and selection of high-performance VLSI adders. To demonstrate the accuracy of the method in the energy-delay space for selecting adder architecture they implemented and compared the designs in 130nm and 100nm CMOS technologies.

In [15], Amir Ali Khatibzadeh *et al* presented the design of an 8 X 8-bit digital multiplier which provides superior performance when compared to conventional array multipliers in terms of power consumption and speed. The proposed multiplier was implemented in TSMC 0.18um technology and was estimated to operate at a maximum frequency of 1.1GHz while dissipating 22mW of power. In [17], Nazir Mehmood as part of his Master's thesis presented an energy efficient 32-bit multiplier architecture. The multiplier presented was based on the Modified Booth Encoding scheme. The multiplier was implemented in 90nm technology and was found to be superior to a conventional 32-bit CMOS multiplier in terms of power, speed and area. In [18], Pouya Asadi and Keivan Navi proposed a 54X54-bit multiplier design which used high speed, self timed carry look-ahead adder structures. The proposed multiplier was implemented using a radix-4 booth encoding scheme to reduce the number of the partial products that had to be generated. The multiplier was implemented using a 0.13um CMOS process at a nominal supply voltage of 1.3V.

In [19], Ryusuke Egawa *et al* laid their focus on the increasing power density values with circuits being implemented in deep submicron technologies. To address this issue in multiplier designs they proposed a sophisticated multiplier which aims at partial product reduction tree and incorporates bit level parallelism. The proposed multiplier design was applied to a 32-bit design and was compared to conventional 32-bit multipliers and was shown to achieve significant improvement in terms of power consumption and area occupied. In [20], Dimitris Bekiaris *et al* presented a radix-4 array multiplier based on 4-to-1 multiplexers. The proposed multiplier was implemented using TSMC 0.13um technology library and was compared to Modified-Booth array multiplier.

Christian Schuster *et al* in their paper [21] focused on comparison of multipliers at architecture level and aimed at selecting the multiplier architecture that offered the minimum total power dissipation by simultaneously optimizing both static and dynamic power dissipation. The designs were analyzed in UMC 0.18um technology. In [22], Thomas K. Callaway and Earl E. Swartzlander, Jr aimed at analyzing the power-delay characteristics of CMOS multipliers. They implemented four multiplier architectures in three different bit widths and modeled the multipliers for the product of the power consumed and the delay associated with the multiplier. Using the results obtained they were able to identify the best possible multiplier architecture that offered optimum power-delay product. In [23], Leonardo L. de Oliveira *et al* presented performance comparisons between two multiplier architectures. They drew comparisons between a radix-4 array multiplier which was modified to handle sign bits in 2's complement and a general Modified Booth multiplier at the physical implementation level. They compared these multiplier architectures for both pipelined and unpipelined versions.

CHAPTER III

POWER DISSIPATION IN CMOS GATES

The three main issues the researchers face during the design of VLSI circuits and systems involve area, performance and power [2]. Until recently the prime focus was laid on the parameters of area and performance and power had tertiary importance. With designs being implemented in deep submicron technologies (feature size less than 130nm) the focus has now been laid on the problem of power and is of primary importance. This is because of the possibility of implementing tens of millions of gates on a small die which has increased power density and total power dissipation and is at the limits what packaging, cooling and other infrastructure can support [1].

Historically, both power density and power consumption in integrated circuits have increased with the development of technology. The high power density in the deep submicron technology not only poses problems with packaging and cooling but also addresses reliability concerns [1]. This is because in temperature causes the mean failure time of devices to increase exponentially, possibility of formation of local hot spots on the chips, increased leakage and may also lead to timing degradation [1]. Addressing this issue International Technology Roadmap for Semiconductors (ITRS) has predicted some values for power in deep submicron technology which are listed in the table below [1]

Node	90nm	60nm	45nm
Dynamic Power per cm^2	1X	1.4X	2X
Static Power per cm^2	1X	2.5X	6.5X
Total Power per cm^2	1X	2X	4X

Table 3.1: Power Consumption in deep submicron technology

3.1 Sources of Power Dissipation in CMOS circuits

Average power dissipation in traditional CMOS circuits can be expressed as sum of three main components [3]:

1) Static Power Dissipation
2) Dynamic Power Dissipation
3) Short-Circuit Power Dissipation during switching of transistors

In the form of an equation it can be expressed as below

$$P_{avg} = P_{static} + P_{dynamic} + P_{short\text{-}circuit}$$

Where P_{avg} is the average power dissipation, P_{static} is the static power dissipation, $P_{dynamic}$ is the dynamic power dissipation due to the switching of transistors and $P_{short\text{-}circuit}$ is the short-circuit power dissipation.

The figure below shows power consumed by the microprocessor chips over the years [4].

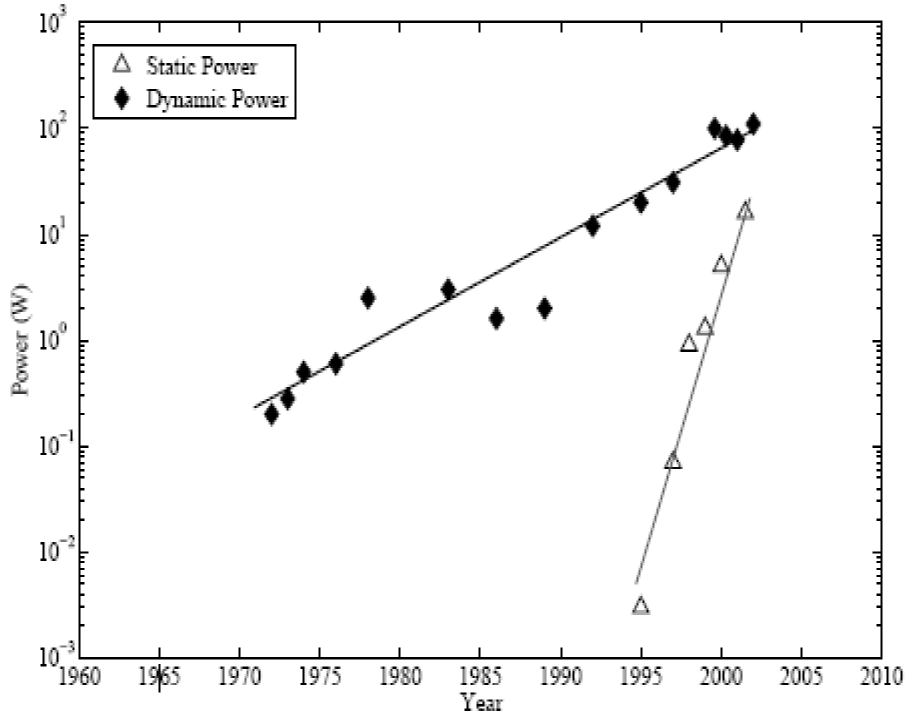


Figure 3.1: Power Consumed in Watts for Microprocessor Chips

The figure clearly shows that in recent years static power is of equal importance during the design process. This significant change in static power is due to the fact that leakage in CMOS has increased with reduction in transistor sizes.

3.1.1 Static Power Dissipation

CMOS circuits even in their idle states dissipate some power [3]. This is known as Static Power. This dissipation is a result of the various leakage currents through the nMOS and pMOS transistors in their nominally off condition. There are four main sources of leakage currents through a CMOS gate and have been shown in the figure below [1]

1) Sub-Threshold Leakage (I_{SUB}): This is the current which flows from the drain to the source current of a transistor operating in the weak inversion region.

2) Gate Leakage (I_{GATE}): This is the current which flows from the gate through the oxide to the substrate due to gate oxide tunneling and hot-carrier injection.

3) Gate Induced Drain Leakage (I_{GIDL}): This is the current which flows from the drain to the substrate induced by a high field effect in the MOSFET drain caused by a high V_{DG}.

4) Reverse Bias Junction Leakage (I_{REV}): This is the current caused by minority carrier drift and generation of electron/hole pairs in the depletion region.

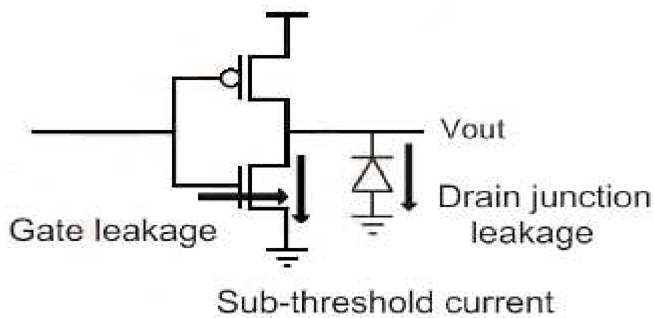


Figure 3.2: Leakage Currents

Sub-threshold current is the current which flows through a gate when it is not turned off completely [1]. The value of the sub-threshold current is dependent upon the thermal voltage and it increases exponentially with increasing temperature [1]. Sub-threshold current value also depends on the exponential difference between the V_{GS} and V_T of the gate. A pretty good approximation of the sub-threshold current value can be given by the following equation [1]

$$I_{SUB} = \mu C_{ox} V_{th}^2 \frac{W}{L} \cdot e^{\frac{V_{GS}-V_T}{nV_{th}}}$$

Where W and L are the dimensions of the transistor, V_{th} is the thermal voltage and n is a fabrication process dependent parameter which usually varies from 1.0 – 2.5 [1].

The tunneling of current through the gate oxide causes gate leakage. At the deep submicron level the gate oxide thickness is so thin that the value of gate leakage current is substantial. This value can be as large as the value of the sub-threshold current and hence important. Modern researchers have found out that the gate leakage could be reduced by using high-k dielectric materials as gate oxides [1].

3.1.2 Dynamic Power Dissipation

The power dissipated by a CMOS gate due to the charging and discharging of the capacitances in the circuit is dynamic power. The figure below illustrates power dissipation in a CMOS inverter [5]

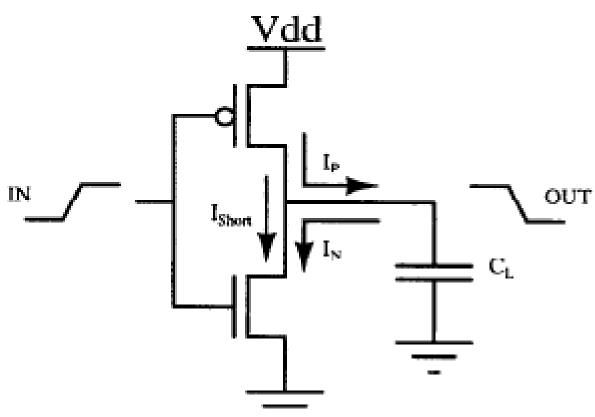


Figure 3.3: Dynamic Power in a CMOS Inverter

Here C_L is the sum of the parasitic capacitances of nMOS and pMOS gates, wire capacitance and the internal capacitance of the circuits driven by the inverter.

The energy per transition in the above CMOS gate is given by [1]:

$$Energy/Transition = C_L V_{dd}^2$$

Here V_{dd} is the supply voltage.

Using the above equation of Energy/Transition we can now describe the dynamic power of the CMOS gate by the following equation [1]:

$$P_{dynamic} = Energy/Transition \times f_{clock} = C_{eff}V_{dd}^2 f_{clock}$$

$$C_{eff} = C_L P_{trans}$$

Here P_{trans} is the probability of an output transition and f_{clock} is the system clock frequency.

The above equation clearly shows that dynamic power is directly related to the switching activity in the gate and also the capacitance of the gate. Hence, dynamic power is data dependant rather than transistor size.

As the technology has scaled down there has been a constant increase in the value of the dynamic power dissipation owing to the factors of increased clock frequencies and increased functional requirements of the circuits. One can effectively reduce the dynamic power dissipation value by lowering the value of the supply voltage as its value varies quadratically with the supply voltage. This lowering of the supply voltage in the modern designs has been limited because at the deep submicron level. This is because lowering the value of supply voltage decreases the value of the drive current resulting in slower circuits. To maintain consistency the threshold voltage value needs to be lowered which could increase the static power dissipation. This causes problems in deep submicron technologies where static power dissipation cannot be ignored. Hence now various other options are being explored to reduce dynamic power dissipation.

3.1.3 Short-Circuit Power Dissipation

In CMOS gates under some switching conditions there exists a direct path between the power supply and ground. This is when current flows directly from the power supply to the ground through the CMOS gate. The power dissipation occurring under this condition is known as short-circuit power dissipation. This power dissipation occurs because of the finite rise and fall times of the input waveforms at the gate.

Short-circuit power dissipation in CMOS gates can be reduced by matching the rise and fall times at the inputs of the CMOS gates and can be kept in check. This value can also be reduced by lowering the value of the supply voltage [5]. One can easily note that in dynamic circuits there is no short-circuit power dissipation as there never exists a path between the power supply and ground at any time because the precharge and the evaluation stages in the dynamic logic circuits are independent of each other.

CHAPTER IV

METHODOLOGY

4.1 Design Flow

Design flow describes a series of sequential steps that are performed during the design process. These steps at various levels of the design process are coordinated by the designer with the help of various electronic design automation (EDA) tools. In this section an overview of the flow of design process has been given and also the tools used have been briefly described. The figure below gives the implementation methodology

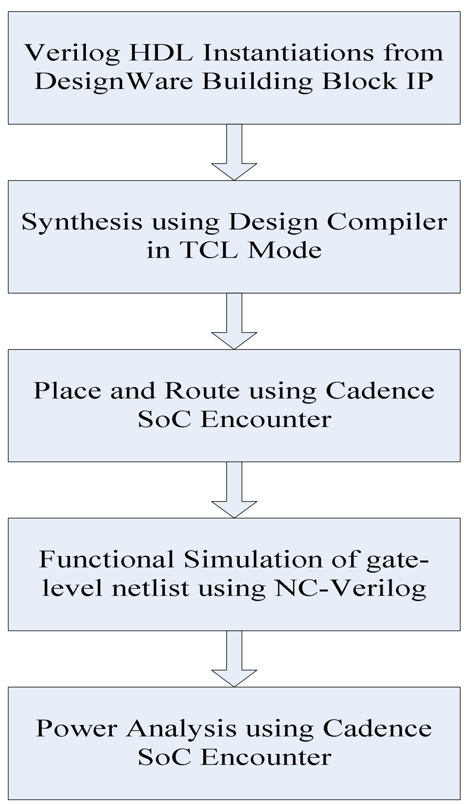


Figure 4.1: Implementation Methodology

4.1.1 Overview of EDA Tools Used

DesignWare and Building Block IP

The DesignWare Building Block Intellectual Property (IP) is a collection of reusable IP blocks that can be used by the designers to have transparent and high-level optimization of performance of the IP blocks during synthesis. The IP provides HDL instantiations that are technology independent and can be directly used by the designers. This enables design reuse and increased productivity. This IP is a product of the Synopsys, Inc.

Design Complier

Design Compiler is a synthesis tool that synthesizes the HDL designs available into optimized technology-dependent, gate-level netlists. This tool is a product of the Synopsys, Inc. which supports a wide range of design styles and can optimize both combinational and sequential designs for speed, area and power.

SoC Encounter

SoC Encounter is a product of the Cadence Design Systems, Inc. which provides a fast and feasibility analysis of the designs whether they meet the required targets and is physically realizable. The SoC Encounter system supports advanced timing closure and routing, as well as signoff analysis engines for final implementation.

NC-Verilog

The NC-Verilog simulator is a Verilog HDL simulator that will simulate the behavior of a digital circuit provided that a Verilog HDL model file exists for that circuit. This tool is a product of the Cadence Design Systems, Inc. and delivers high performance and high capacity verilog simulation.

4.2 Synthesis

Synthesis is one of the important parts of the design phase where the designs in HDL are converted to gate-level netlists. In this thesis, synthesizable and technology independent Verilog HDL instantiations of adder and multiplier architectures that are available with the DesignWare Building Block IP of Synopsys, Inc. is utilized for synthesis. The figure below shows Verilog HDL instantiation of Adder in DesignWare Building Block IP.

```
module DW01_add_inst( inst_A, inst_B, inst_CI, SUM_inst, CO_inst );

  parameter width = 8;

  input [width-1 : 0] inst_A;
  input [width-1 : 0] inst_B;
  input inst_CI;
  output [width-1 : 0] SUM_inst;
  output CO_inst;

  // Instance of DW01_add
  DW01_add #(width)
    U1 (.A(inst_A), .B(inst_B), .CI(inst_CI), .SUM(SUM_inst), .CO(CO_inst) );

endmodule
```

Figure 4.2: Verilog HDL instantiation of Adder in DesignWare Building Block IP

The adder and multiplier architectures were synthesized for various bit widths in sub-micron and deep submicron technologies. Adder architectures are implemented for bit widths of 16-bits, 32-bits, 64-bits, 128-bits and multiplier architectures are implemented for bit widths of 16-bits, 32-bits, 64-bits in all the technologies. The details of the implemented adder architectures, multiplier architectures and the implemented technologies are presented in the below tables

Implementation	Function
rpl	Ripple Carry Synthesis Model
rpcs	Ripple Carry Select Architecture Synthesis Model
pparch	Delay-Optimized Flexible Parallel-Prefix Synthesis Model
csm	Conditional-Sum Synthesis Model
clf	Fast Carry-Look Ahead Synthesis Model
cla	Carry-Look Ahead Synthesis Model
bk	Brent-Kung Architecture Synthesis Model

Table 4.1: Synthesis Implementations of Adder Architectures

Implementation	Function
wall	Booth-recoded Wallace-Tree Synthesis Model
pparch	Delay-Optimized Flexible Booth Wallace Synthesis Model
nbw	Either a non-Booth (A_width+B_width ≤ 41) or a Booth Wallace-tree (A_width+B_width > 41) Synthesis Model
csa	Carry-Save Array Synthesis Model

Table 4.2: Synthesis Implementations of Multiplier Architectures

Library	Process	Voltage
GSCLIB045	Cadence 45nm	1.1 V
GSCLIB090	Cadence 90nm	0.9V
GSCLIB180	Cadence 180nm	3.0V
OSU250	TSMC 250nm	2.5V

Table 4.3: Implementation Technologies

The process of synthesizing various adder and multiplier architectures for the specified bit widths in various technologies is done using Design Compiler and is automated using Tcl script file. The script file contains design compiler directives that are executed in a sequential manner.

In the initial part of the synthesis the user defined variables are set and also the required technology library and Synopsys database are set. Then the design is read-in and later the synthesis environment, design constraints and compiler directives are set which control the synthesis process. Now, the read-in design is initially roughly compiled for timing only in the first compilation stage and later in the second compilation stage the circuit is refined for circuit area and timing. At the end of the second compilation stage a gate-level netlist is generated and also the simulation information on timing, area and power are saved into reports. The generated netlist and reports are technology dependent and differ from one particular implementation to the other.

The figure below shows the synthesis flow

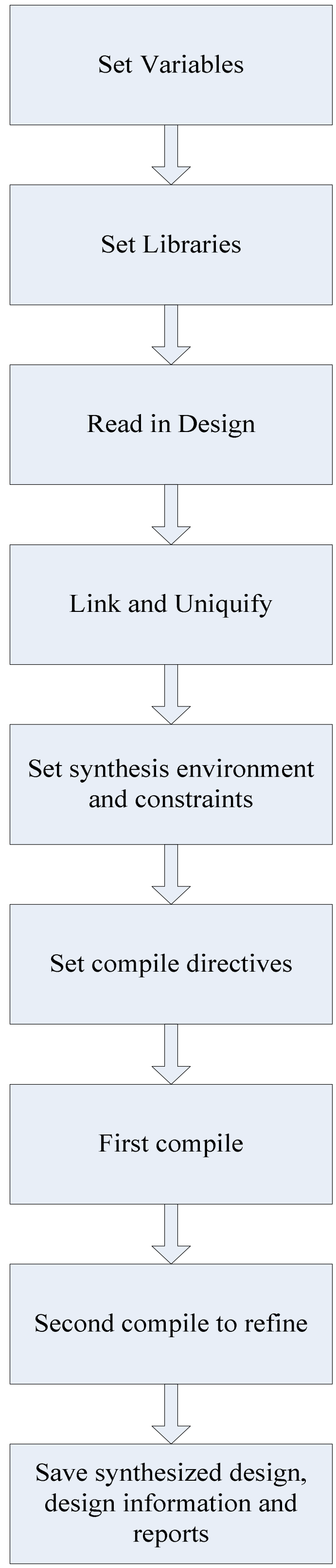


Figure 4.3: Synthesis Flow

4.3 Place & Route

Place and Route is the process of generating a physical design from the gate-level netlist that is generated from the synthesis stage. The generated netlist after the synthesis stage is technology dependent and comprises of the design implemented using standard cells from the implemented technology library.

The place and route of the synthesized design is done using Cadence SoC Encounter and the process is automated using a script file in Tcl. The script file consists of a series of commands internal to the Cadence SoC Encounter that are executed in a sequential order which is in accordance with the place and route process flow. Initially, the design which is in the form of gate-level netlist and also the lef file of the technology library used are setup. An initial floorplan is created for the design and the power structures are created. Later the design which comprises of the standard cells of the technology library is placed which is followed by the routing of the power nets. Then a trial route is performed and then the timing graph is built and the results are saved into a preliminary timing report. The design is optimized prior to clock tree synthesis (CTS) and then clock tree synthesis is performed on the design if the design contains a clock port. The results from the clock tree synthesis are saved and RC extraction is done and the timing results are saved. The design is again optimized after the clock tree synthesis and again the timing results are saved. The design is now optimized for leakage power and later global routing is done. After completion of global routing of the design, timing graph is built and the final results which give the delay associated with design are saved. Now the design, netlist, sdf and def files associated with the design are saved.

The figure below shows the Place & Route flow

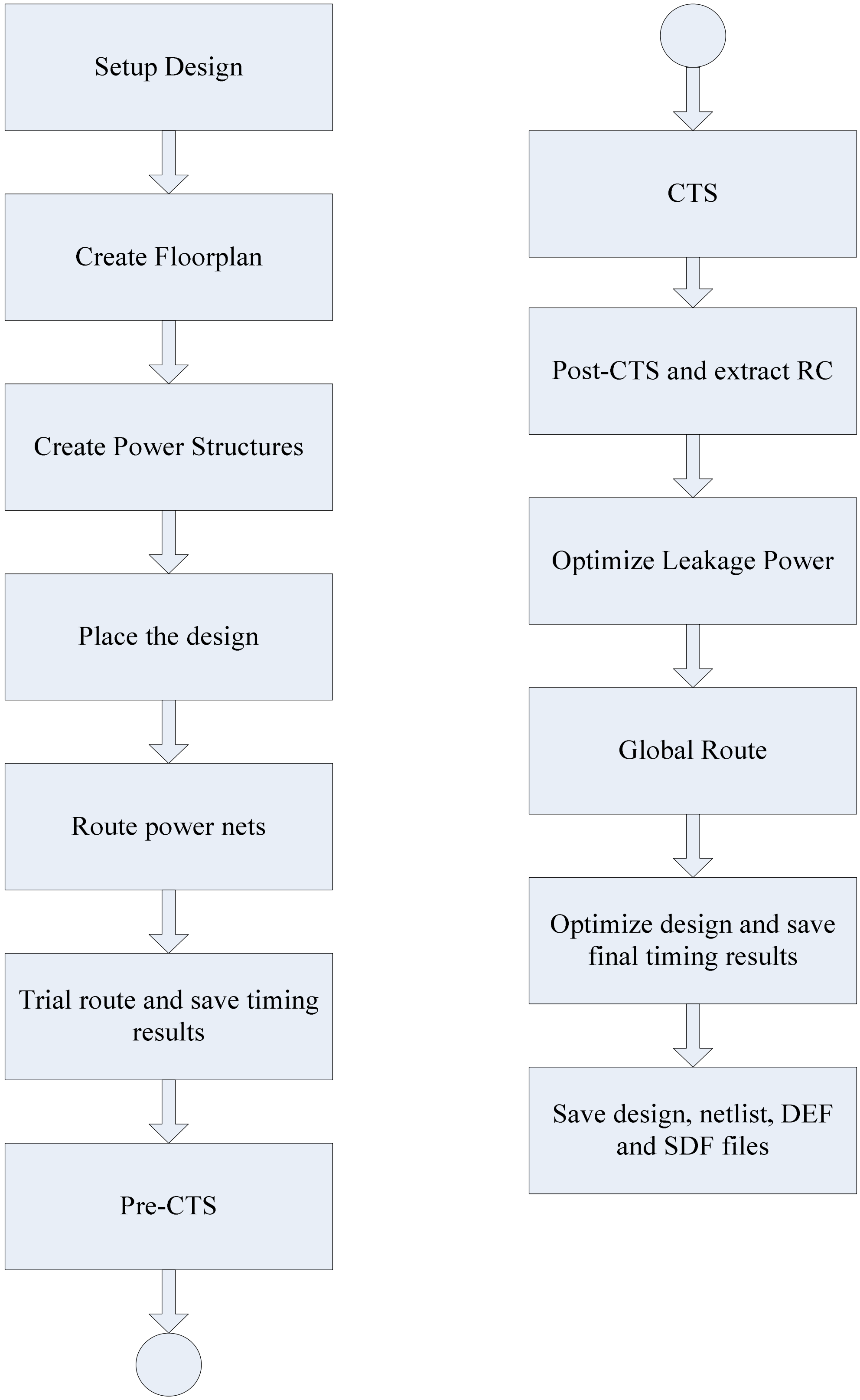


Figure 4.4: Place & Route Flow

4.4 Functional Simulation and Power Analysis

Once place and route of the design is complete and the final netlist of the design is saved, functional simulation is done on the design. The netlist which comprises of the standard cells of the technology library used is a Verilog HDL file and so NC-Verilog a HDL simulator is used for the simulation process. The simulation is carried out using an automated test bench where the design is tested for various test vectors. The results of the simulation are dumped into a vcd file. Once the simulation process is complete and successful, the design is analyzed for power. Power analysis of the design is done utilizing the results from the simulation that had been dumped into a vcd file. Performing power analysis gives the total amount of dynamic power and leakage power consumed by the design. It also gives the information on the total capacitance of the largest toggled net, total i_d and total activity during the functional simulation process. Cadence SoC Encounter is used for performing power analysis on the design and the results from this are saved into a report.

CHAPTER V

RESULTS AND CONCLUSION

5.1 Results

In this thesis, various implementations of adders and multipliers were implemented at the physical implementation level in all the technology libraries as specified in Table 4.3 for both unpipelined and two-stage pipelined versions. The pipelined versions of the adders and multipliers were synthesized using the compiler directive 'pipeline_design' during the synthesis process.

5.1.1 Power Analysis

Power dissipation has been a key area of concern for the design engineers implementing design in deep submicron technologies. It has been observed in deep submicron technology that there is a considerable impact of leakage power on the value of average power. Also, with higher circuit densities as the power dissipation per unit area is very high there is a need to address the problem of controlling the value of dynamic power dissipation.

Below the results for dynamic power dissipation, leakage power, Energy-Delay product for adders and multipliers in deep submicron technologies for unpipelined and two-stage pipelined versions have been presented.

Adder - Two-Stage Pipelined

Adder - Unpipelined

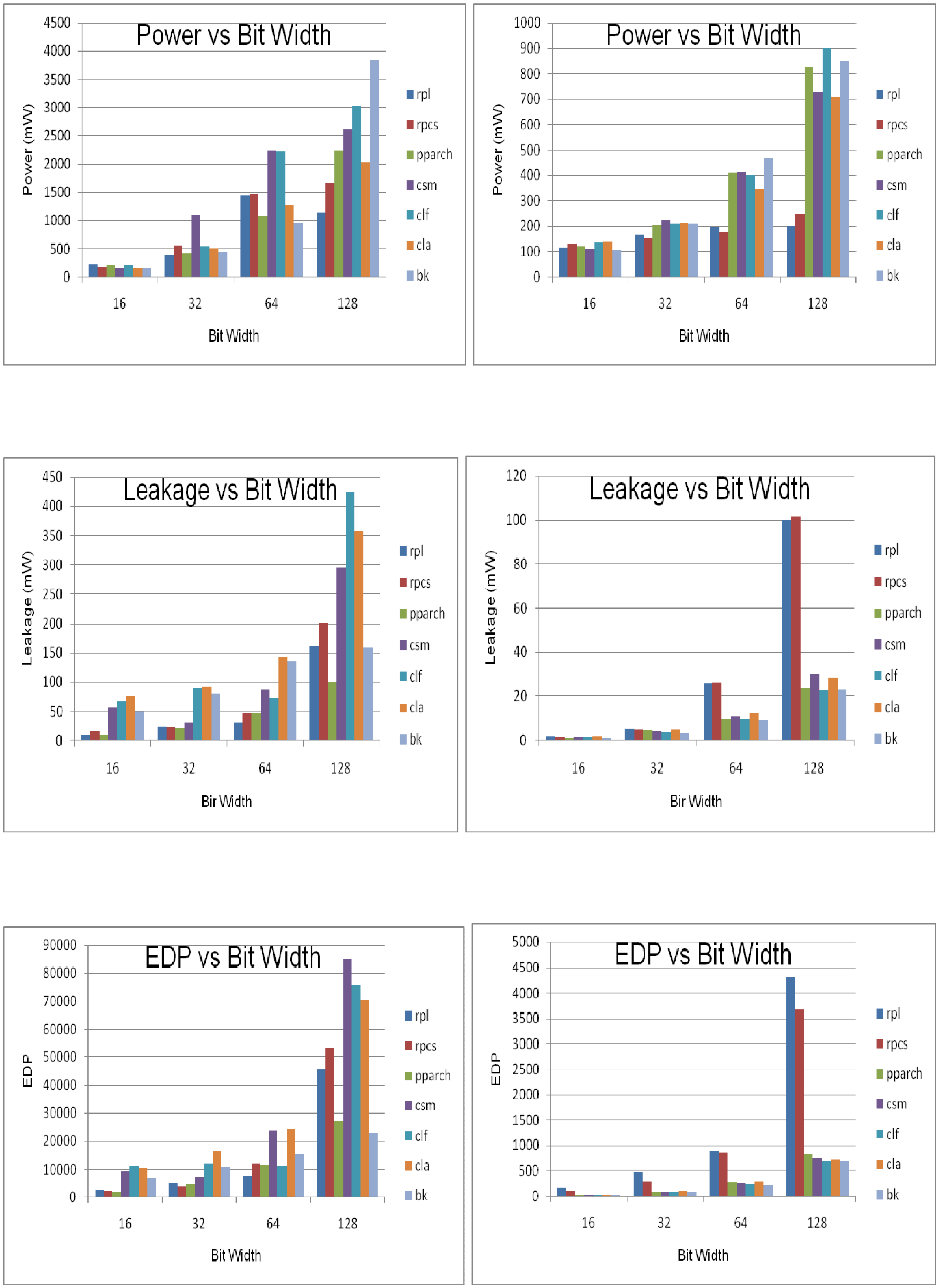


Figure 5.1 Power Results for Adder in 45nm Technology

Multiplier - Two-Stage Pipelined | Multiplier – Unpipelined

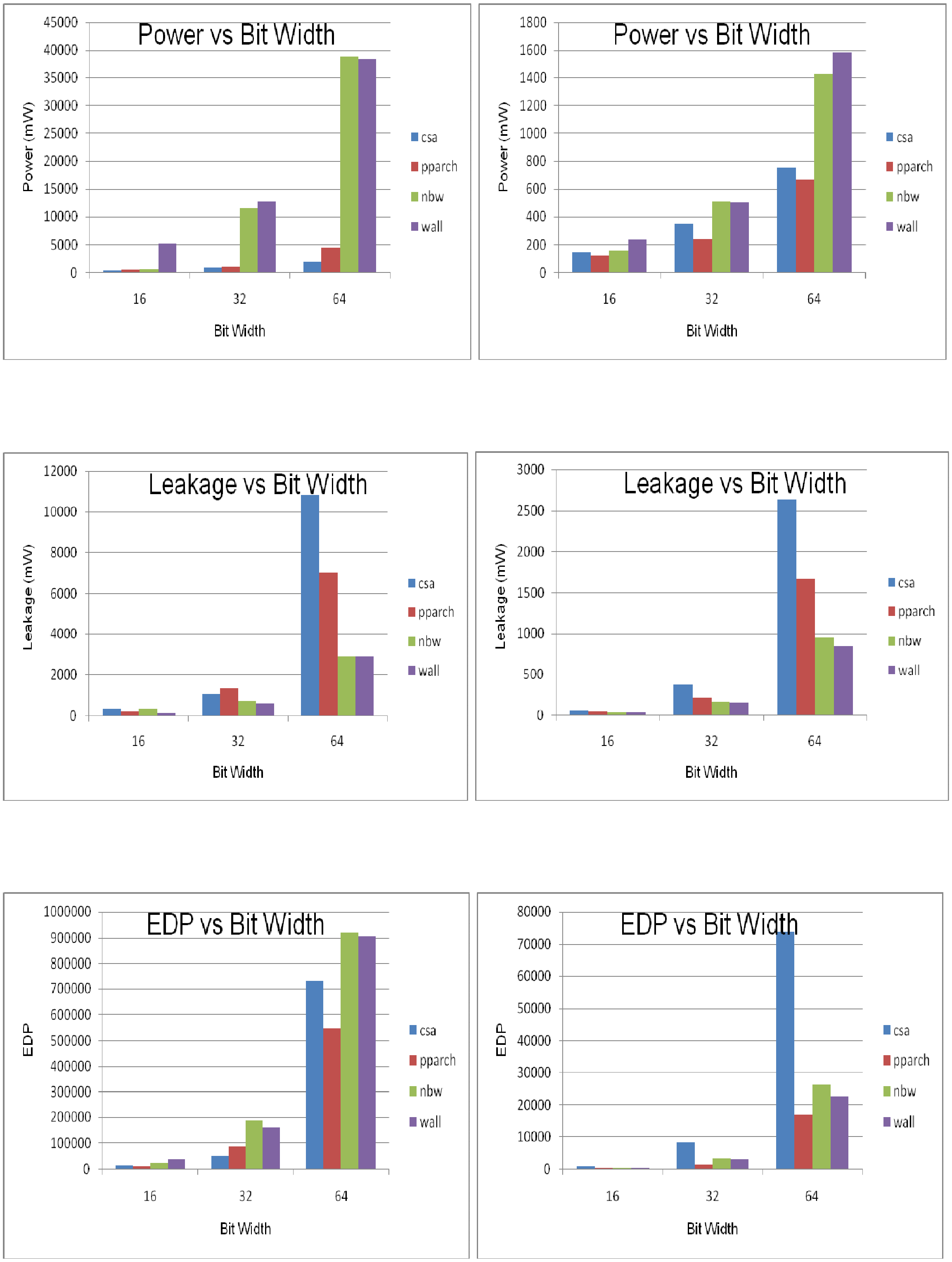


Figure 5.2 Power Results for Multiplier in 45nm Technology

Adder - Two-Stage Pipelined   Adder - Unpipelined

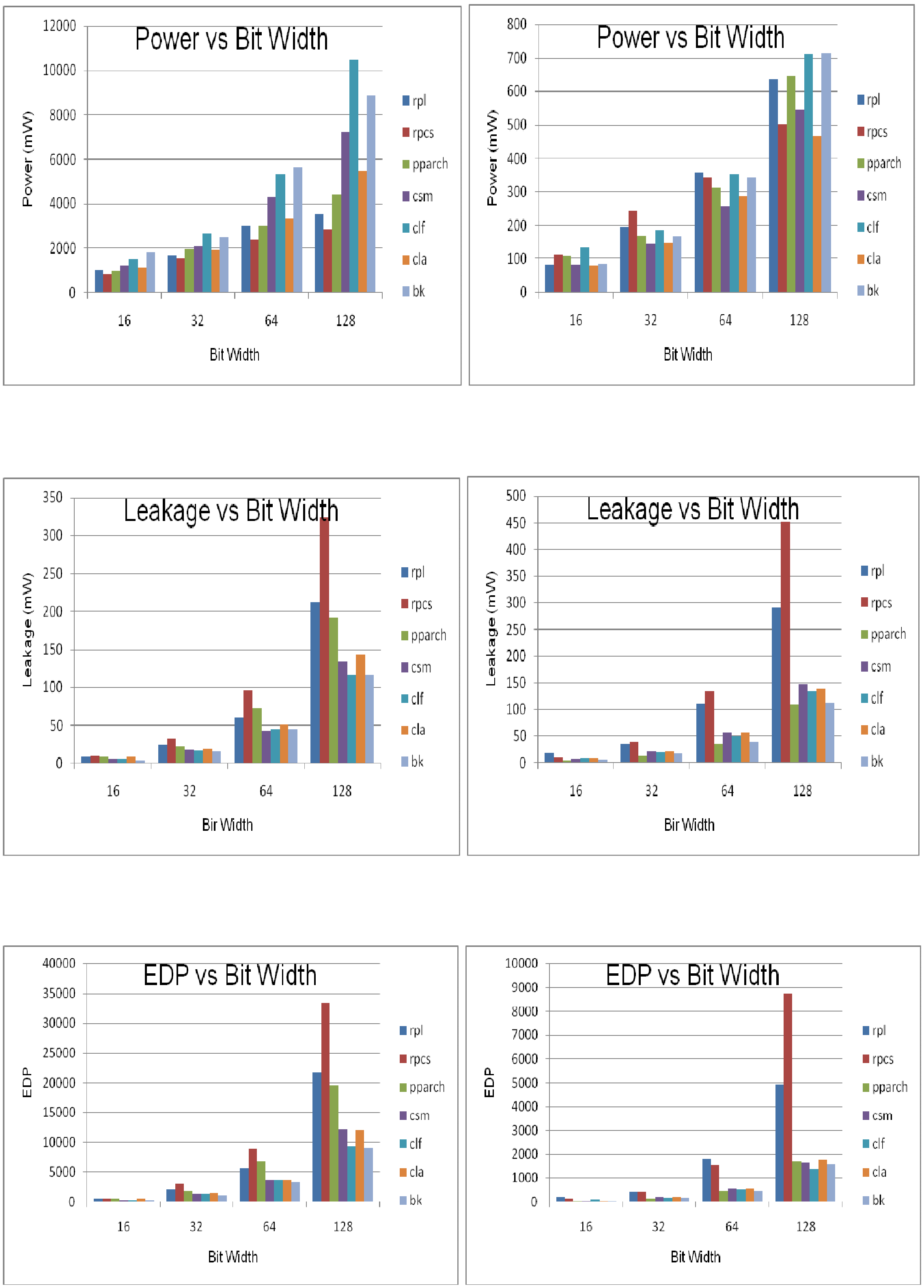


Figure 5.3 Power Results for Adder in 90nm Technology

Multiplier - Two-Stage Pipelined

Multiplier – Unpipelined

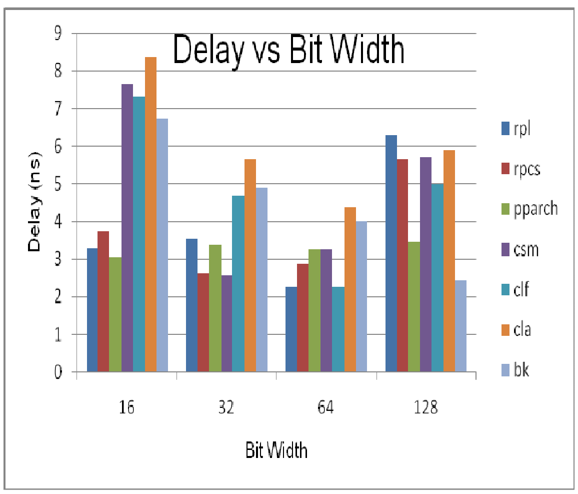
Figure 5.4 Power Results for Multiplier in 90nm Technology

5.1.2 Delay and Area

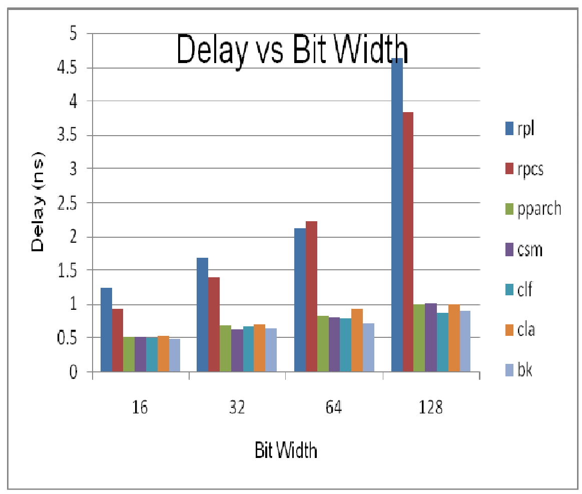
Delay and area associated with any circuit are directly related to the input bit widths. As the bit width increases the associated delay increases and so does the area.

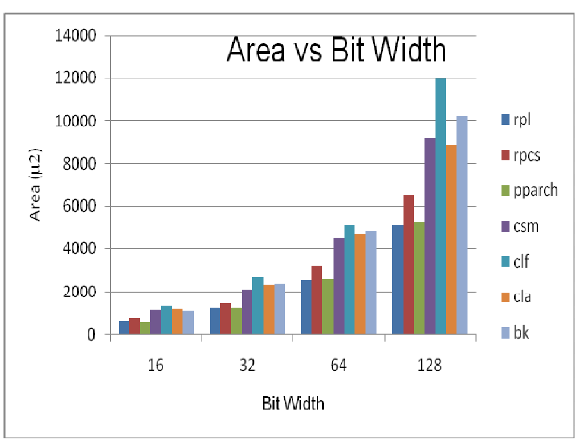
Below the results for delay, area, number of instances (gate count and register count) for adders and multipliers in deep submicron technologies for unpipelined and two-stage pipelined versions has been presented.

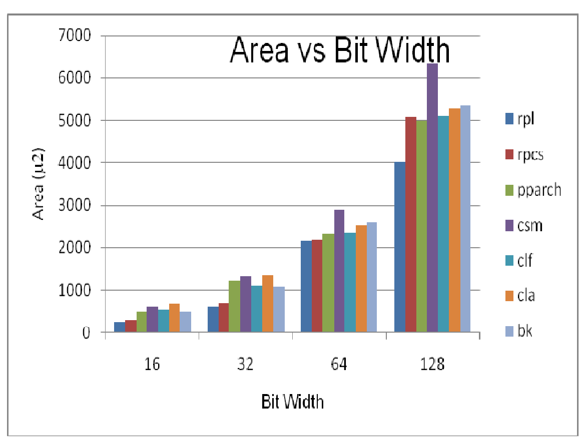
Adder - Two-Stage Pipelined



Adder - Unpipelined

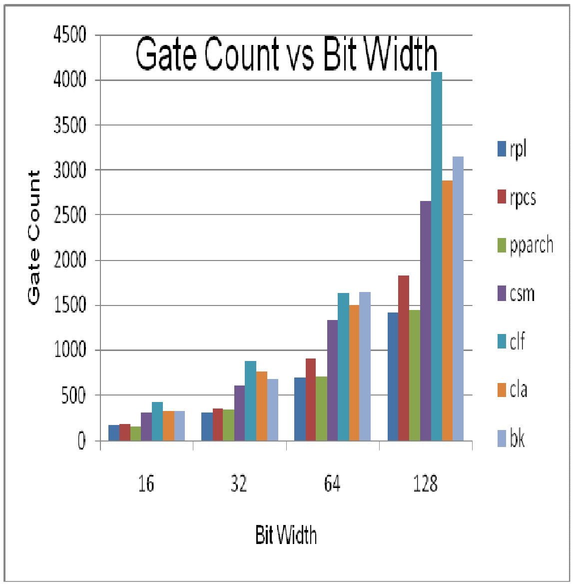


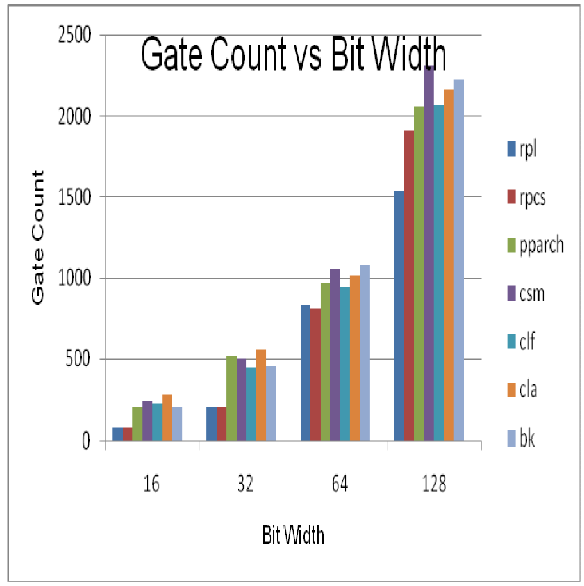




Adder - Two-Stage Pipelined

Adder – Unpipelined





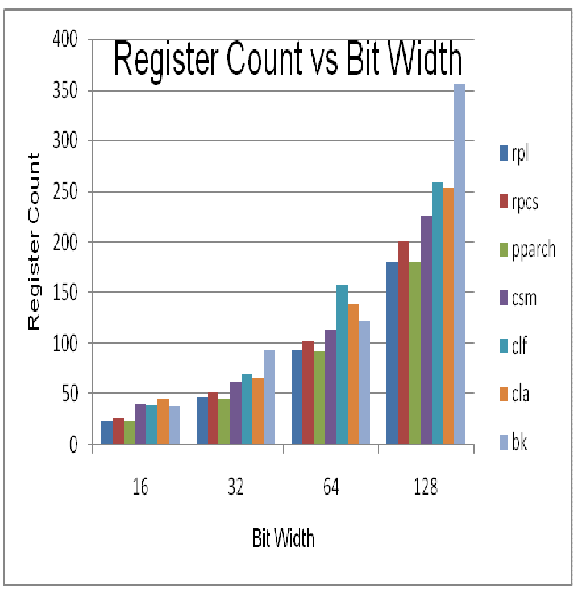
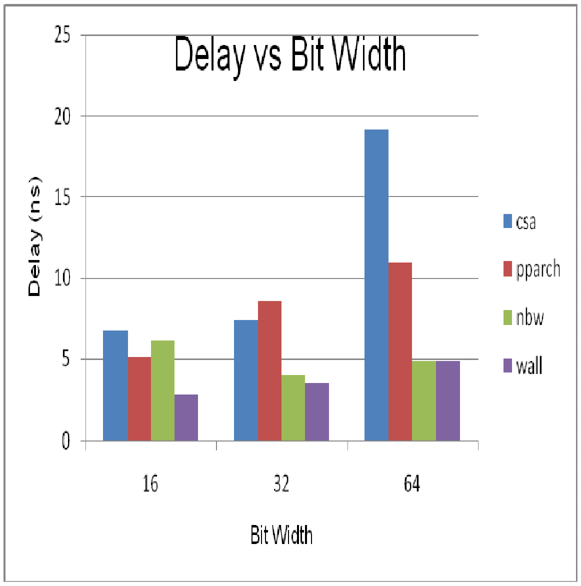
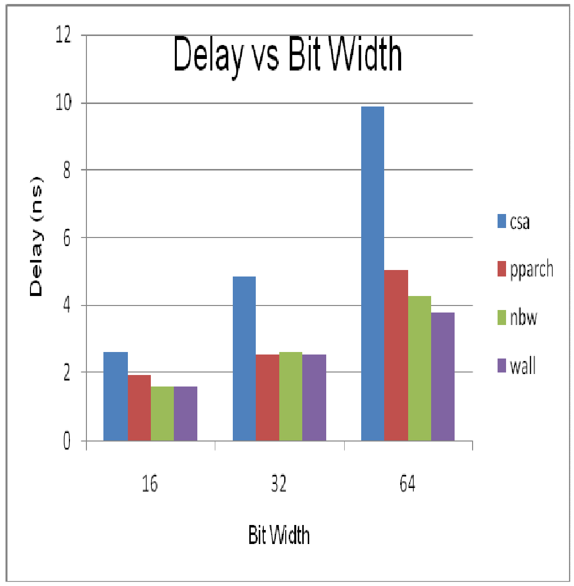


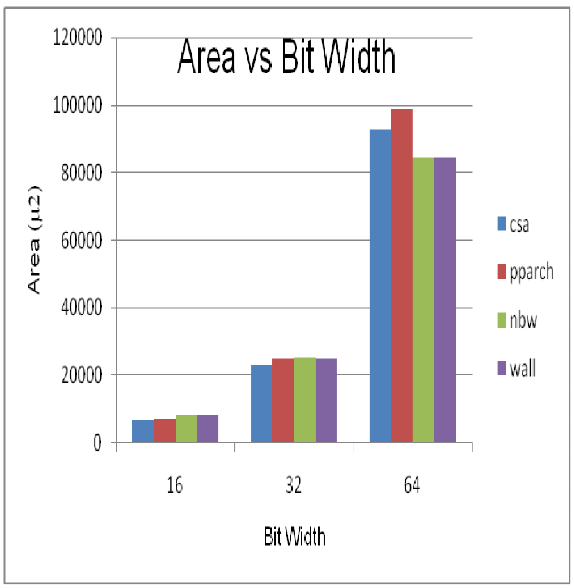
Figure 5.5: Area and Delay Results for Adder in 45nm Technology

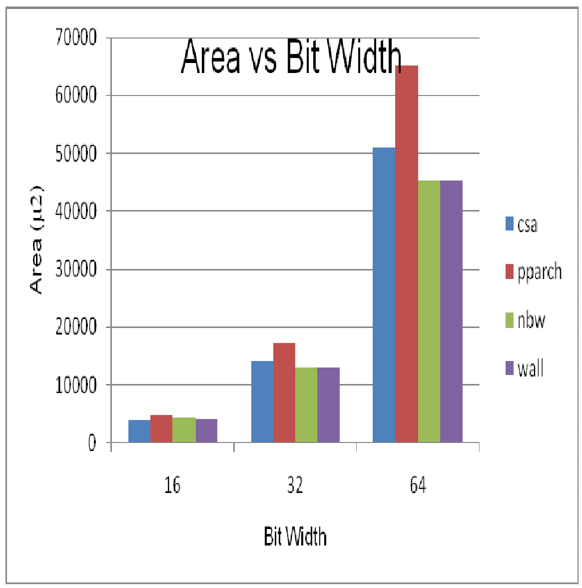
Multiplier - Two-Stage Pipelined

Multiplier – Unpipelined

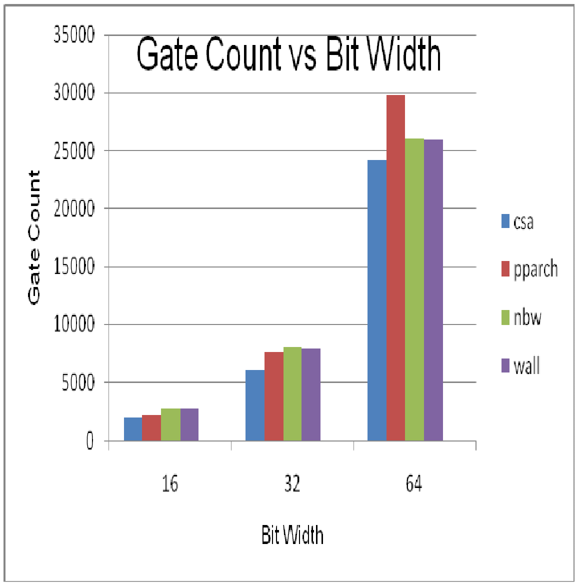




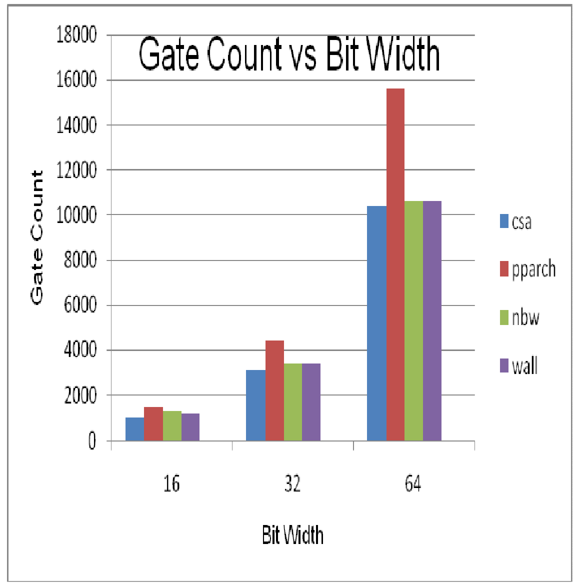




Multiplier - Two-Stage Pipelined



Multiplier – Unpipelined



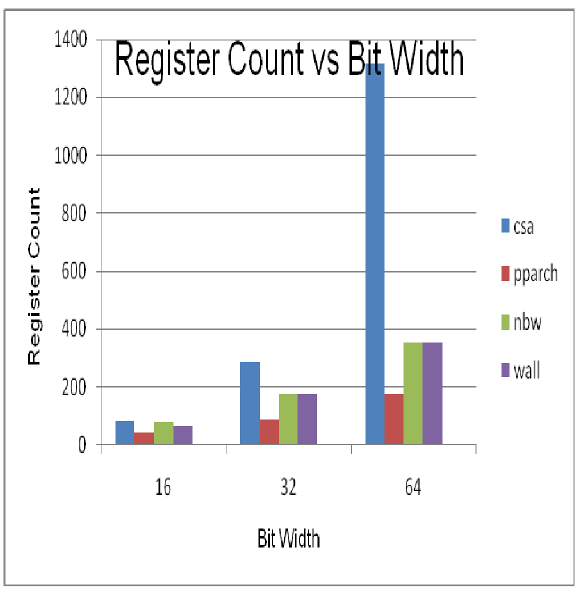
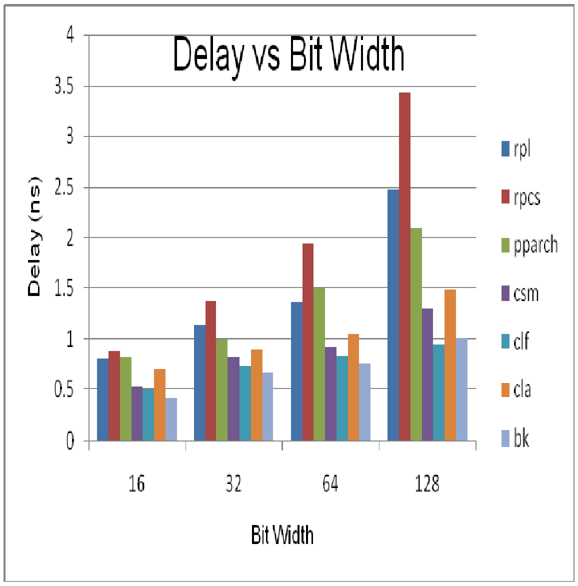
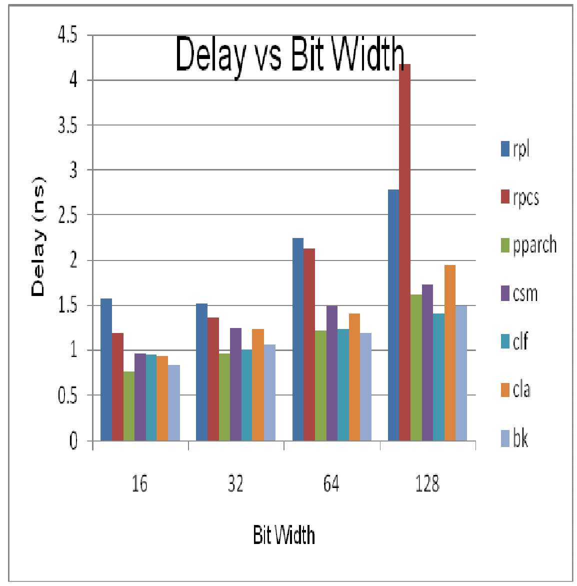


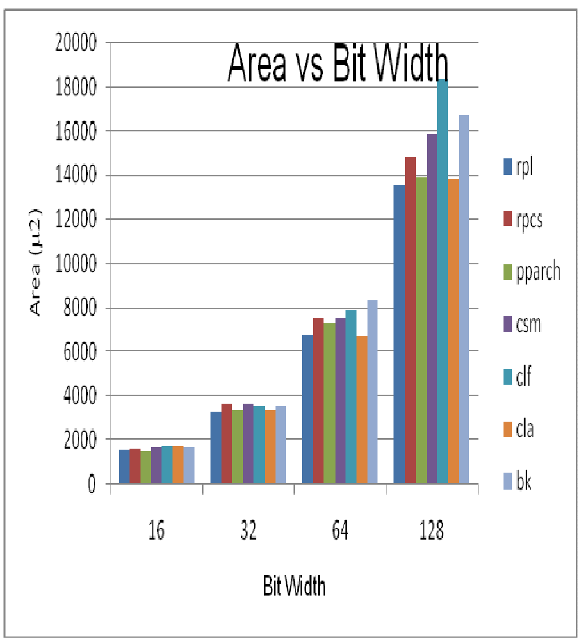
Figure 5.6: Area and Delay Results for Multiplier in 45nm Technology

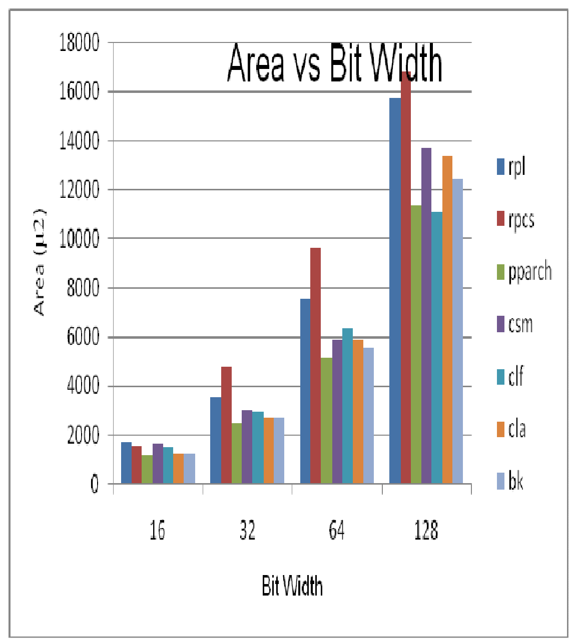
Adder - Two-Stage Pipelined

Adder – Unpipelined

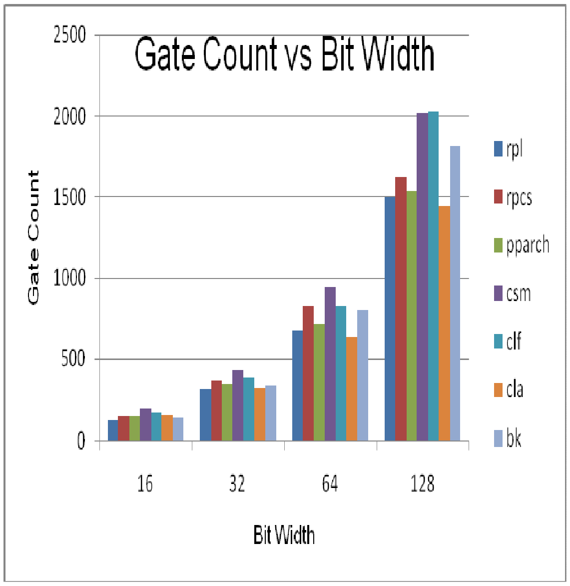




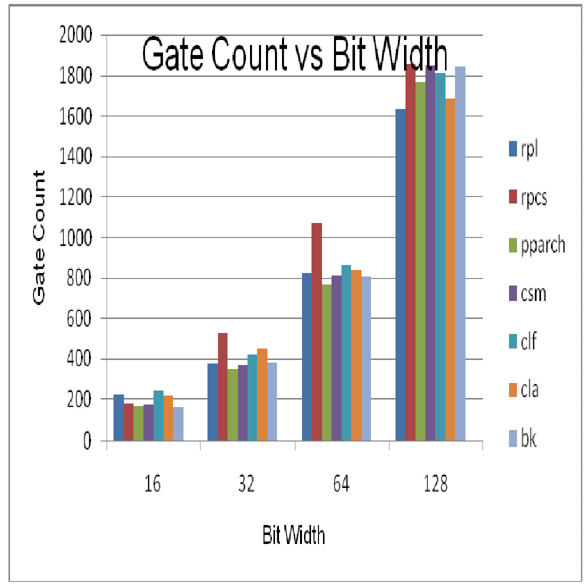




Adder - Two-Stage Pipelined



Adder – Unpipelined



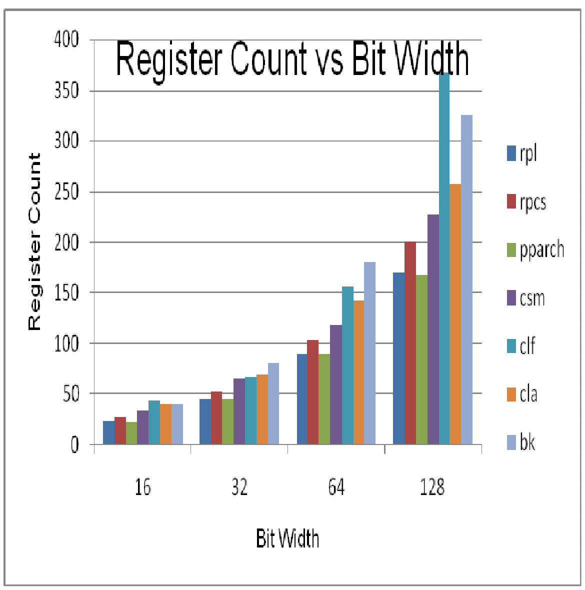
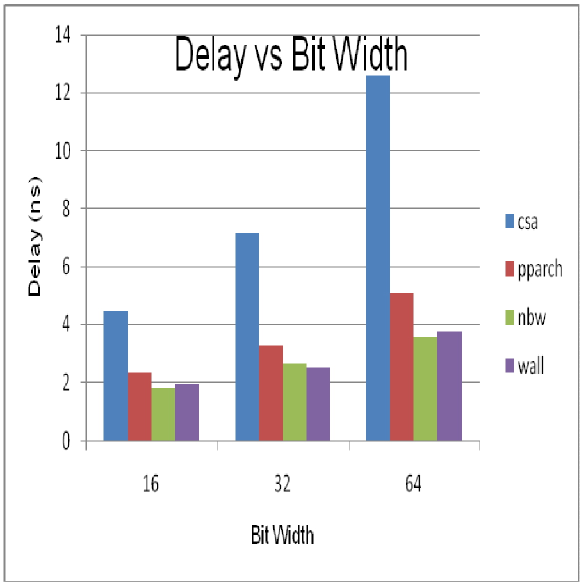
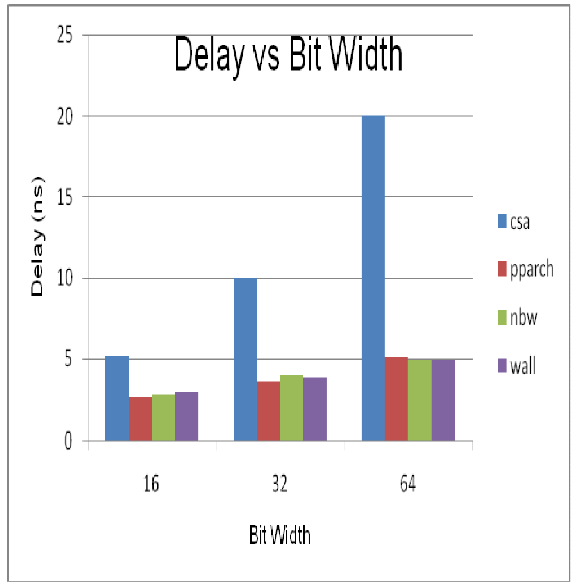


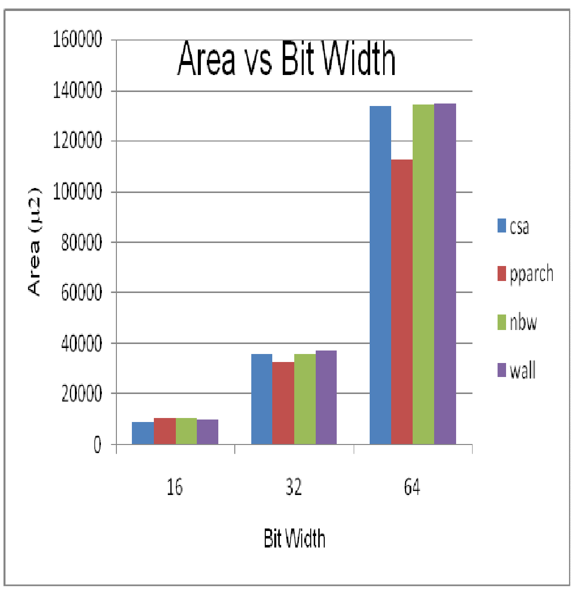
Figure 5.7: Area and Delay Results for Adder in 90nm Technology

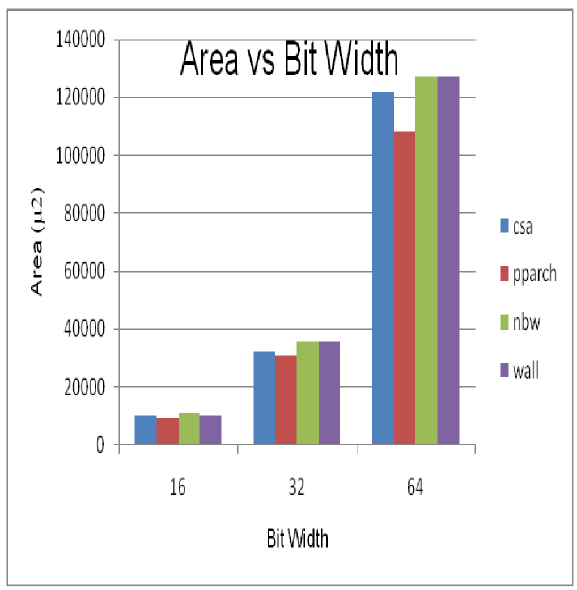
Multiplier - Two-Stage Pipelined

Multiplier – Unpipelined

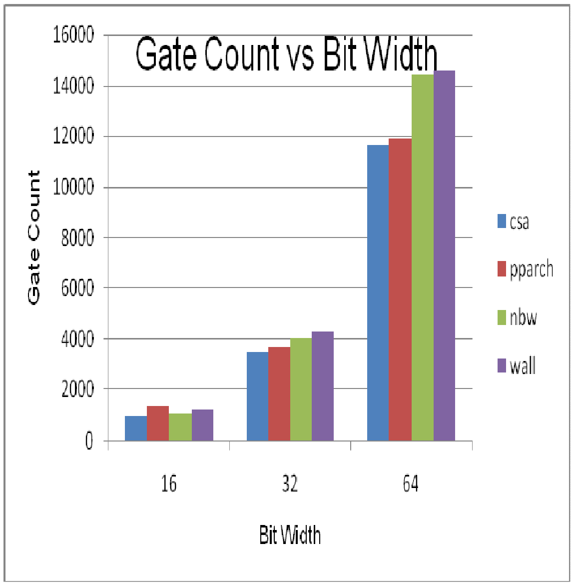




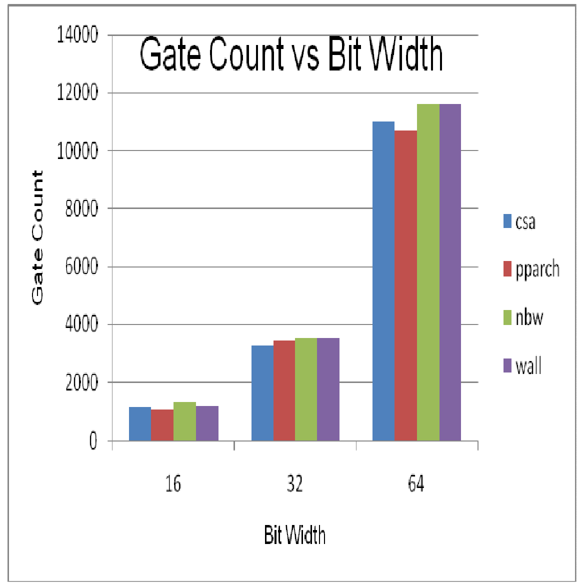




Multiplier - Two-Stage Pipelined



Multiplier – Unpipelined



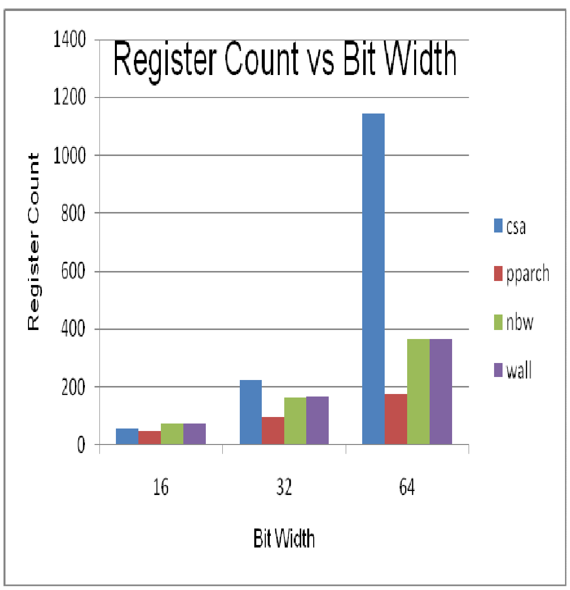


Figure 5.8: Area and Delay Results for Multiplier in 90nm Technology

With the results presented above for adders and multipliers we can observe that as the bit width increased the parameters of interest such as power, delay, area also increased. This is true with both adders and multiplier implementations. One can observe that the contribution of leakage power is very high in the designs implemented in deep submicron technologies. Observing the results one can notice that an unpipelined version of an adder or a multiplier has shown more efficient performance in terms of power dissipation and associated delay when compared to its two-stage pipelined counterpart. This is in contrary to the very concept of pipelining which assures the designer of an improved performance of the system both in terms of power dissipation and delay. This disagreement with the actual concept of pipelining can be addressed by observing the results of the gate count associated with the adder and multiplier designs. One can see that the gate count for a two-stage implementation is almost double to its counterpart in all cases. Here the gate count represents the number of standard cells instances that are needed to implement the specific implementation. Also, for a pipelined system the total number of gates is given by the sum of the gate count and the register count associated with the implementation. So, this tremendous increase in number of the gates required to implement the same function as an unpipelined counterpart has accounted for these varying results.

5.2 Conclusions

The work in this thesis is based on the analysis of the existing architectures of adders and multipliers implemented in modern day technologies. The analysis is based on the physical implementation of the designs which take into account the parasitic capacitances and also wiring delay associated. Various implementations of adders and multipliers have been analyzed at the physical implementation level for power (dynamic and leakage), delay and area in four different technologies. Utilizing the results that were obtained, implementations of adders and multipliers that offer the optimal parameters in terms of power, delay and area have been concluded. In the adder design the implementations cla and pparch offered optimal results and in the multiplier design the implementations pparch and wall offered optimal results. These results were consistent in all implemented technologies for all implemented bit widths. Depending upon the need of the application the designer can also choose from other implementations.

Further study in this topic can be made on the low-power design of these structures. Since, power has been the prime focus point for designers investigating various low-power techniques that can be used to minimize the power dissipation can be useful. Several power reduction techniques such as clock gating, controlled switching activity, capacitance reduction and use of low-voltage standard cell library can be studied and can be implemented on the adder and multiplier structures to test for their behavior.

REFERENCES

[1] Michael Keating, David Flynn, Robert Aitken, Alan Gibbons, Kaijian Shi "Low Power Methodology Manual for System-On-Chip Design", Springer, 2007

[2] Saraju P. Mohanty, Nagarajan Ranaganathan, Elias Kougianos, Priyadarsan Patra "Low-Power High-Level Synthesis for Nanoscale CMOS Circuits", Springer, 2008

[3] Vikas Kumar, Cadence Design Systems, Inc. "Low-Power CMOS Circuit Design" in http://www.powermanagementdesignline.com/howto/189500236

[4] Saeeid Tahmasbi Oskuii "Design of Low-Power Reduction-Trees in Parallel Multipliers" Ph.D. Dissertation, Norwegian University of Science and Technology, 2008

[5] Sataporn Pornpromlikit "Power-Efficient Design of 16-Bit Mixed Operand Multipliers" Master's Thesis, Massachusetts Institute of Technology, 2004

[6] James E. Stine "Digital Arithmetic Datapath Design Using Verilog HDL", Kluwer Academic Publishers, 2004

[7] Abdellatif Bellaouar, Mohamed I. Elmasry "Low-Power Digital VLSI Design: Circuits and Systems", Kluwer Academic Publishers, 1995

[8] Vojin G. Oklobdzija, "High-Speed VLSI Arithmetic Units: Adders and Multipliers", in "Design of High-Performance Microprocessor Circuits", Book Chapter, Book edited by A. Chandrakasan, IEEE Press, 2000

[9] Reto Zimmermann "Binary Adder Architectures for Cell-Based VLSI and their Synthesis" Ph.D. Dissertation, Swiss Federal Institute of Technology, Zurich

[10] Ranaganathan Panchagam "Minimization of Power Dissipation in Digital Circuits Using Pipelining and a Study of Clock Gating Technique" Master's Thesis, University of Central Florida, 2004

[11] Sean Kao, Radu Zlatanovici, Borivoje Nikolic "A 240ps 64b Carry-Lookahead Adder in 90nm CMOS", Solid-State Circuits Conference 2006, ISSC 2006, Digest of Technical Papers, IEEE International, 2006

[12] Keivan Navi, Omid Kavehei "Low-Power and High-Performance 1-Bit CMOS Full-Adder Cell", JOURNAL OF COMPUTERS, VOL. 3, NO. 2, FEBRUARY 2008

[13] Lan Wei "Implementation of Pipelined Bit-parallel Adders" Master's Thesis, Linkoping Institute of Technology, Sweden, 2003

[14] Sheng Sun, Carl Sechen "Post-Layout Comparison of High Performance 64b Static Adders in Energy- Delay Space", Computer Design 2007, ICCD 2007, 25th International Conference, 2007

[15] Vojin G. Oklobdzija, Bart R. Zeydel "Energy-Delay Characteristics of CMOS Adders" in "High-Performance Energy-Efficient Microprocessor Design", Book Chapter, Springer US, 2006

[16] Amir Ali Khatibzadeh, Kaamran Raahemifar, Majid Ahmadi "A 1.8 V 1.1 GHz Novel Digital Multiplier", Electrical and Computer Engineering, 2005 Canada Conference, 2005

[17] Nasir Mehmood, Martin Hansson, and Atila Alvandpour "An Energy-Efficient 32-bit Multiplier Architecture in 90-nm CMOS", 24th Norchip Conference, 2006

[18] Pouya Asadi and Keivan Navi "A Novel High-Speed 54×54 bit Multiplier", American Journal of Applied Sciences, 2007

[19] Ryusuke Egawa, Jubei Tada, Gensuke Goto, Tadao Nakamura "A Sophisticated Multiplier in Advanced CMOS Technologies" The 21st International Technical Conference on Circuits/Systems, Computers and Communications, 2006

[20] Dimitris Bekiaris, Kiamal Z. Pekmestzi and Chris Papachristou "A High-Speed Radix-4 Multiplexer-Based Array Multiplier" Proceedings of the 18th ACM Great Lakes symposium on VLSI, 2008

[21] Christian Schuster, Jean-Luc Nagel, Christian Piguet, and Pierre-André Farine "Leakage Reduction at the Architectural Level and Its Application to 16 Bit Multiplier Architectures" in "Integrated Circuit and System Design", Springer Berlin / Heidelberg, 2004

[22] Thomas K. Callaway and Earl E. Swartzlander, Jr "Power-Delay Characteristics of CMOS Multipliers" Proceeding of13th IEEE Symposium on Computer Arithmetic, 1997

[23] Leonardo L. de Oliveira, Cristiano Santos, Daniel Ferrão, Eduardo Costa, José Monteiro, João Baptista Martins, Sergio Bampi, Ricardo Reis "A Comparison of Layout Implementations of Pipelined and Non-Pipelined Signed Radix-4 Array Multiplier and Modified Booth Multiplier Architectures" in "VLSI-SoC: From Systems To Silicon", Springer Boston, 2007

[24] Design Compiler User Guide, Version 2002.05, Synopsys

[25] DesignWare Building Block IP User Guide, Synopsys

[26] Encounter User Guide, Product Version 6.2.2, Cadence

APPENDICES

APPENDIX-A

In this section the script files that have been used to automate the design flow process have been presented.

Synthesis script

```
set names [getenv "names"]
set bit [getenv "bit"]
set my_toplevel $names
set my_clock_pin clk
set my_clk_freq_MHz 5000
set my_input_delay_ns 0
set my_output_delay_ns 0

set type [getenv "type"]
set my_verilog_files [getenv "source"]

set OSU_FREEPDK [format "%s%s"  [getenv "OSU_FREEPDK"] "/lib/files"]
set search_path [concat  $search_path $OSU_FREEPDK]
set link_library [set target_library [concat  [list gscl45nm.db] [list dw_foundation.sldb]]]
set target_library "gscl45nm.db"
define_design_lib WORK -path ./WORK
set verilogout_show_unconnected_pins "true"

set_ultra_optimization true
set_ultra_optimization -force

analyze -f verilog $my_verilog_files

elaborate $my_toplevel

current_design $my_toplevel

link
uniquify
```

```
set my_period [expr 1000 / $my_clk_freq_MHz]

if {  [find port $my_clock_pin] == [list $my_clock_pin] } {

set clk_name $my_clock_pin
  create_clock -period $my_period $clk_name

}
if {  [find port $my_clock_pin] == [list] } {
   set clk_name vclk
  create_clock -period $my_period -name $clk_name
}
set_driving_cell  -lib_cell INVX4  [all_inputs]
set_input_delay $my_input_delay_ns -clock $clk_name [remove_from_collection
[all_inputs] $my_clock_pin]
set_output_delay $my_output_delay_ns -clock $clk_name [all_outputs]

set port_load [load_of slow/INVX4/A]
set_load $port_load [all_outputs]

set_implementation $type [list U1]

#/* compile -ungroup_all -map_effort high */
compile -map_effort high

report_resources

compile -incremental_mapping -map_effort high

report_resources

check_design
report_constraint -all_violators

set filename [format "%s%s"  $my_toplevel ".vh"]
write -f verilog -output $filename

set filename [format "%s%s"  $my_toplevel ".sdc"]
write_sdc $filename

set filename [format "%s%s"  $my_toplevel ".db"]
write -hier -output $filename

redirect timing.rep { report_timing }
redirect cell.rep { report_cell }
redirect power.rep { report_power }
```

quit

Place and Route script

```
# Setup design and create floorplan
loadConfig ../../scripts/encounter.conf
#commitConfig

# Create Initial Floorplan
floorplan -r 1.0 0.6 20 20 20 20

# Create Power structures
addRing -spacing_bottom 5 -width_left 5 -width_bottom 5 -width_top 5 -spacing_top 5 -
layer_bottom metal5 -width_right 5 -around core -center 1 -layer_top metal5 -
spacing_right 5 -spacing_left 5 -layer_right metal6 -layer_left metal6 -nets { gnd vdd }

# Place standard cells
placeDesign

# Route power nets
sroute -noBlockPins -noPadRings

# Perform trial route and get initial timing results
trialroute
buildTimingGraph
setCteReport
report_timing -nworst  10 -net > timing.rep.1.placed

# Run in-place optimization
# to fix setup problems
setIPOMode -mediumEffort -fixDRC -addPortAsNeeded
initECO ./ipo1.txt
fixSetupViolation
endECO
buildTimingGraph
setCteReport
report_timing -nworst  10 -net > timing.rep.2.ipo1

# Run Clock Tree Synthesis
createClockTreeSpec -output encounter.cts -bufFootprint buf -invFootprint inv
specifyClockTree -clkfile encounter.cts
ckSynthesis -rguide cts.rguide -report report.ctsrpt -macromodel report.ctsmdl -
fix_added_buffers
```

```
# Output Results of CTS
trialRoute -highEffort -guide cts.rguide
extractRC
reportClockTree -postRoute -localSkew -report skew.post_troute_local.ctsrpt
reportClockTree -postRoute -report report.post_troute.ctsrpt

# Run Post-CTS Timing analysis
setAnalysisMode -setup -async -skew -autoDetectClockTree
buildTimingGraph
setCteReport
report_timing -nworst  10 -net > timing.rep.3.cts

# Perform post-CTS IPO
setIPOMode -highEffort -fixDrc -addPortAsNeeded -incrTrialRoute  -restruct -topomap
initECO ipo2.txt
setExtractRCMode -default -assumeMetFill
extractRC
fixSetupViolation -guide cts.rguide

# Fix all remaining violations
setExtractRCMode -detail -assumeMetFill
extractRC
if {[isDRVClean -maxTran -maxCap -maxFanout] != 1} {
fixDRCViolation -maxTran -maxCap -maxFanout
}

endECO
cleanupECO

# Run Post IPO-2 timing analysis
buildTimingGraph
setCteReport
report_timing -nworst  10 -net > timing.rep.4.ipo2

# Add filler cells
addFiller -cell FILL -prefix FILL -fillBoundary

# Connect all new cells to VDD/GND
globalNetConnect vdd -type tiehi
globalNetConnect vdd -type pgpin -pin vdd -override

globalNetConnect gnd -type tielo
globalNetConnect gnd -type pgpin -pin gnd -override

# Run global Routing
globalDetailRoute
```

```
# Get final timing results
setExtractRCMode -detail -noReduce
extractRC
buildTimingGraph
setCteReport
report_timing -nworst  10 -net > timing.rep.5.final

# Output GDSII
#streamOut final.gds2 -mapFile gds2_encounter.map -units 1000 -mode ALL -stripes 1
delayCal -sdf final.sdf
saveNetlist -excludeLeafCell final.v
saveDesign final.enc
defOut -floorplan -netlist -routing final.def

# Output DSPF RC Data
rcout -spf final.dspf

# Run DRC and Connection checks
verifyGeometry
verifyConnectivity -type all

exit
```

Configuration file

```
set names [getenv "names"]

# Specify the name of your toplevel module
set my_toplevel  $names

##################################################
# No changes required below
##################################################

global env
set OSU_FREEPDK $env(OSU_FREEPDK)

global rda_Input
set rda_Input(ui_netlist) $names.vh
set rda_Input(ui_timingcon_file) $names.sdc
set rda_Input(ui_topcell) $names

set rda_Input(ui_netlisttype) {Verilog}
set rda_Input(ui_ilmlist) {}
```

```
set rda_Input(ui_settop) {1}
set rda_Input(ui_celllib) {}
set rda_Input(ui_iolib) {}
set rda_Input(ui_areaiolib) {}
set rda_Input(ui_blklib) {}
set rda_Input(ui_kboxlib) ""
set rda_Input(ui_timelib) "$OSU_FREEPDK/lib/files/gscl45nm.tlf"
set rda_Input(ui_smodDef) {}
set rda_Input(ui_smodData) {}
set rda_Input(ui_dpath) {}
set rda_Input(ui_tech_file) {}
set rda_Input(ui_io_file) ""
set rda_Input(ui_buf_footprint) {BUF}
set rda_Input(ui_delay_footprint) {BUF}
set rda_Input(ui_inv_footprint) {INV}
set rda_Input(ui_leffile) "$OSU_FREEPDK/lib/files/gscl45nm.lef"
set rda_Input(ui_core_cntl) {aspect}
set rda_Input(ui_aspect_ratio) {1.0}
set rda_Input(ui_core_util) {0.7}
set rda_Input(ui_core_height) {}
set rda_Input(ui_core_width) {}
set rda_Input(ui_core_to_left) {}
set rda_Input(ui_core_to_right) {}
set rda_Input(ui_core_to_top) {}
set rda_Input(ui_core_to_bottom) {}
set rda_Input(ui_max_io_height) {0}
set rda_Input(ui_row_height) {}
set rda_Input(ui_isHorTrackHalfPitch) {0}
set rda_Input(ui_isVerTrackHalfPitch) {1}
set rda_Input(ui_ioOri) {R0}
set rda_Input(ui_isOrigCenter) {0}
set rda_Input(ui_exc_net) {}
set rda_Input(ui_delay_limit) {1000}
set rda_Input(ui_net_delay) {1000.0ps}
set rda_Input(ui_net_load) {0.5pf}
set rda_Input(ui_in_tran_delay) {120.0ps}
set rda_Input(ui_captbl_file) {}
set rda_Input(ui_cap_scale) {1.0}
set rda_Input(ui_xcap_scale) {1.0}
set rda_Input(ui_res_scale) {1.0}
set rda_Input(ui_shr_scale) {1.0}
set rda_Input(ui_time_unit) {none}
set rda_Input(ui_cap_unit) {}
set rda_Input(ui_sigstormlib) {}
set rda_Input(ui_cdb_file) {}
set rda_Input(ui_echo_file) {}
```

```
set rda_Input(ui_qxtech_file) {}
set rda_Input(ui_qxlib_file) {}
set rda_Input(ui_qxconf_file) {}
set rda_Input(ui_pwrnet) {vdd}
set rda_Input(ui_gndnet) {gnd}
set rda_Input(flip_first) {1}
set rda_Input(double_back) {1}
set rda_Input(assign_buffer) {0}
set rda_Input(ui_pg_connections) [list \
                    {PIN:vdd:} \
                    {PIN:gnd:} \
                         ]
set rda_Input(PIN:vdd:) {vdd}
set rda_Input(PIN:gnd:) {gnd}
```

Power Analysis script

```
set names [getenv "names"]

restoreDesign final.enc.dat $names

setExtractRCMode -detail -noReduce
extractRC

updatePower -vcd $names.vcd -vcdTop stimulus/dut -noRailAnalysis -report
$names.power vdd

exit
```

APPENDIX-B

In this section the results for adder and multiplier implementations in submicron technologies (180nm and 250nm) have been presented. These results are in the form of excel spread sheets.

Technology	Implementation	Types	Bit	Area	Delay	Power	Leakage	Energy	EDP	Total Gates	Register Count	Gate Count
DW_180	add	rpl	16	9888	0.83	20302	0.019178	16850.7	13986.1	184	24	160
	add	rpcs	16	11418	0.439	33985	0.01172	14919.4	6549.62	191	27	164
	add	pparch	16	10194	0.634	28526	0.015345	18085.5	11466.2	196	24	172
	add	csm	16	14981	0.592	41117	0.027928	24341.3	14410	289	36	253
	add	clf	16	11331	0.472	36828	0.014512	17382.8	8204.68	224	35	189
	add	cla	16	11796	0.496	31308	0.015796	15528.8	7702.28	212	41	171
	add	bk	16	9786	0.419	26953	0.01074	11293.3	4731.89	182	34	148
	add	rpl	32	20863	0.812	46051	0.038508	37393.4	30363.4	402	48	354
	add	rpcs	32	27130	0.936	61835	0.054281	57877.6	54173.4	607	53	554
	add	pparch	32	21004	0.829	45001	0.039305	37305.8	30926.5	415	48	367
	add	csm	32	32429	0.666	94371	0.067552	62851.1	41858.8	706	55	651
	add	clf	32	23529	0.52	55245	0.032599	28727.4	14938.2	460	79	381
	add	cla	32	21812	0.704	46404	0.039491	32668.4	22998.6	417	72	345
	add	bk	32	22395	0.478	62722	0.027253	29981.1	14331	456	68	388
	add	rpl	64	41553	1.326	61979	0.126561	82184.2	108976	845	91	754
	add	rpcs	64	53545	1.689	75711	0.196701	127876	215983	1219	103	1116
	add	pparch	64	41788	1.243	67543	0.117251	83955.9	104357	851	92	759
	add	csm	64	72656	0.725	219310	0.169309	159000	115275	1589	105	1484
	add	clf	64	57667	0.643	166670	0.095537	107169	68909.7	1320	133	1187
	add	cla	64	44513	0.947	72731	0.110761	68876.3	65225.9	844	144	700
	add	bk	64	45487	0.585	97967	0.067064	57310.7	33526.8	964	128	836
	add	rpl	128	86094	2.291	80950	0.449425	185456	424880	1808	178	1630
	add	rpcs	128	109294	2.416	114160	0.56114	275811	666359	2448	202	2246
	add	pparch	128	85485	1.854	96608	0.360529	179111	332072	1759	180	1579
	add	csm	128	152824	0.882	435380	0.425353	384005	338692	3538	197	3341
	add	clf	128	123904	0.717	314210	0.227497	225289	161532	2893	292	2601
	add	cla	128	88999	1.052	151010	0.238783	158863	167124	1813	260	1553
	add	bk	128	107488	0.584	249000	0.158714	145416	84922.9	2327	294	2033

Figure B.1: Results for Two-Stage Pipelined Implementation of Adder in 180nm Technology

Technology	Implementation	Types	Bit	Area	Delay	Power	Leakage	Energy	EDP	Gate Count
DW_180	add	rpl	16	7269	1.451	1810.1	0.018618	2626.46	3810.99	214
	add	rpcs	16	7730	1.094	2166.6	0.014684	2370.26	2593.06	200
	add	pparch	16	5968	0.598	3256.1	0.006909	1947.15	1164.4	188
	add	csm	16	9619	0.696	2736.8	0.020106	1904.81	1325.75	230
	add	clf	16	6833	0.618	3481.6	0.008901	2151.63	1329.71	220
	add	cla	16	7245	0.618	3099.7	0.0089	1915.61	1183.85	241
	add	bk	16	6328	0.574	3151.6	0.007132	1809.02	1038.38	210
	add	rpl	32	12616	2.678	2040.1	0.059848	5463.39	14631	357
	add	rpcs	32	17732	1.557	3443.2	0.050805	5361.06	8347.17	477
	add	pparch	32	12344	0.76	5985.6	0.018244	4549.06	3457.29	419
	add	csm	32	23605	0.833	5560	0.0594	4631.48	3858.02	581
	add	clf	32	12440	0.803	5540.2	0.021919	4448.78	3572.37	394
	add	cla	32	13599	0.81	5071.6	0.021205	4108	3327.48	450
	add	bk	32	12466	0.733	5908	0.017169	4330.56	3174.3	416
	add	rpl	64	23791	6.592	1994.9	0.260226	13150.4	86687.4	623
	add	rpcs	64	39184	2.111	5002.5	0.160784	10560.3	22292.8	1124
	add	pparch	64	25879	0.923	10405	0.046124	9603.82	8864.33	843
	add	csm	64	52315	1.035	8172.8	0.168798	8458.85	8754.91	1244
	add	clf	64	30357	0.902	11178	0.056886	10082.6	9094.51	977
	add	cla	64	27630	0.952	8959.7	0.051933	8529.63	8120.21	885
	add	bk	64	26365	0.912	10300	0.047933	9393.6	8566.96	881
	add	rpl	128	59490	4.585	4925.6	0.535161	22583.9	103547	1806
	add	rpcs	128	70260	4.084	5553	0.561795	22678.5	92619	2012
	add	pparch	128	55120	1.093	19326	0.115858	21123.3	23087.8	1812
	add	csm	128	118325	1.28	14935	0.471782	19116.8	24469.5	2846
	add	clf	128	61256	1.128	18978	0.150419	21407.2	24147.3	1968
	add	cla	128	53110	1.18	15343	0.123345	18104.7	21363.5	1734
	add	bk	128	60223	1.088	20608	0.122509	22421.5	24394.6	2031

Figure B.2: Results for Unpipelined Implementation of Adder in 180nm Technology

Technology	Implementation	Types	Bit	Area	Delay	Power	Leakage	Energy	EDP	Total Gates	Register Count	Gate Count
DW_180	mult	csa	16	82902	1.937	98829	0.3376	191432	370804	1253	97	1156
	mult	pparch	16	87280	1.835	152800	0.346173	280388	514512	1814	47	1767
	mult	nbw	16	87581	1.213	144250	0.22375	174975	212245	1401	84	1317
	mult	wall	16	82661	1.245	207160	0.231657	257914	321103	1571	72	1499
	mult	csa	32	301178	3.895	270630	2.31659	1.05E+06	4.11E+06	3774	319	3455
	mult	pparch	32	299923	2.609	478280	1.66376	1.25E+06	3.26E+06	5976	94	5882
	mult	nbw	32	300523	1.66	672420	1.09198	1.12E+06	1.85E+06	5897	159	5738
	mult	wall	32	300625	1.665	672130	1.1027	1.12E+06	1.86E+06	5918	159	5759
	mult	csa	64	1266621	7.291	889740	18.4564	6.49E+06	4.73E+07	16034	1383	14651
	mult	pparch	64	1116642	3.802	1649300	8.90695	6.27E+06	2.38E+07	22671	185	22486
	mult	nbw	64	1063208	2.297	2156000	5.10026	4952332	1.14E+07	19999	357	19642
	mult	wall	64	1061355	2.524	2010200	5.59949	5.07E+06	1.28E+07	19863	357	19506

Figure B.3: Results for Two-Stage Pipelined Implementation of Multiplier in 180nm Technology

Technology	Implementation	Types	Bit	Area	Delay	Power	Leakage	Energy	EDP	Gate Count
DW_180	mult	csa	16	65466	3.415	3473.6	0.430768	11862.3	40509.8	996
	mult	pparch	16	68470	1.944	2543.9	0.234077	4945.34	9613.74	1033
	mult	nbw	16	64240	2.018	3132.6	0.246458	6321.59	12757	1037
	mult	wall	16	63382	2.067	3628.8	0.252133	7500.73	15504	1402
	mult	csa	32	255931	6.522	7942.8	3.03906	51802.9	337859	3358
	mult	pparch	32	251600	2.617	6392.7	1.29756	16729.7	43781.6	5045
	mult	nbw	32	267682	2.782	7554.2	1.45576	21015.8	58466	5595
	mult	wall	32	267681	2.783	7551.2	1.45634	21015	58484.7	5595
	mult	csa	64	1027695	12.606	21988	23.224	277181	3.49E+06	12747
	mult	pparch	64	1046609	3.55	13513	7.2104	47971.1	170297	21545
	mult	nbw	64	994081	3.855	16150	7.42936	62258.2	240005	19681
	mult	wall	64	994179	3.854	16167	7.42936	62307.6	240133	19681

Figure B.4: Results for Unpipelined Implementation of Multiplier in 180nm Technology

Technology	Implementation	Types	Bit	Area	Delay	Power	Leakage	Energy	EDP	Total Gates	Register Count	Gate Count
DW_250	add	rpl	16	17943	1.161	22549	0.028729	26179.4	30394.3	205	23	182
	add	rpcs	16	22143	1.204	24699	0.034001	29737.6	35804.1	275	27	248
	add	pparch	16	17664	1.214	19276	0.029789	23401.1	28408.9	203	23	180
	add	csm	16	24852	0.821	41045	0.025545	33697.9	27666	313	31	282
	add	clf	16	23694	0.7	37876	0.021567	26513.2	18559.2	279	48	231
	add	cla	16	21354	0.788	26415	0.022724	20815	16402.2	216	44	172
	add	bk	16	21102	0.691	36139	0.019504	24972	17255.7	241	35	206
	add	rpl	32	35955	1.915	28359	0.094021	54307.5	103999	432	44	388
	add	rpcs	32	44028	1.693	38665	0.096007	65459.8	110823	571	50	521
	add	pparch	32	35838	1.819	28980	0.090113	52714.6	95887.9	437	44	393
	add	csm	32	54945	0.958	88918	0.06738	85183.4	81605.7	680	63	617
	add	clf	32	52815	0.877	76730	0.063059	67292.2	59015.3	684	75	609
	add	cla	32	37968	1.223	36208	0.061714	44282.4	54157.4	425	70	355
	add	bk	32	42018	0.808	46302	0.042093	37412	30228.9	534	81	453
	add	rpl	64	74691	3.353	33557	0.323454	112517	377270	946	90	856
	add	rpcs	64	95190	2.412	62821	0.295832	151524	365476	1264	99	1165
	add	pparch	64	74781	2.174	52609	0.221791	114372	248645	949	87	862
	add	csm	64	125469	1.065	182300	0.174703	194150	206770	1600	126	1474
	add	clf	64	130941	1.025	164220	0.185935	168325	172533	1732	205	1527
	add	cla	64	86046	1.556	70137	0.185071	109133	169811	1044	141	903
	add	bk	64	115842	0.96	156860	0.158352	150586	144563	1551	166	1385
	add	rpl	128	154047	4.322	57471	0.891888	248390	1.07E+06	1951	176	1775
	add	rpcs	128	182763	4.96	60645	1.17507	300799	1.49E+06	2350	201	2149
	add	pparch	128	162234	5.742	47124	1.29643	270586	1.55E+06	2104	173	1931
	add	csm	128	240954	1.17	277870	0.347736	325108	380376	2779	262	2517
	add	clf	128	283737	0.931	384180	0.371553	357672	332993	3845	407	3438
	add	cla	128	160662	2.094	103270	0.461141	216247	452821	1969	267	1702
	add	bk	128	281397	1.12	302250	0.441493	338520	379142	3703	465	3238

Figure B.5: Results for Two-Stage Pipelined Implementation of Adder in 250nm Technology

Technology	Implementation	Types	Bit	Area	Delay	Power	Leakage	Energy	EDP	Gate Count
DW_250	add	rpl	16	8808	2.652	1518.6	0.033314	4027.33	10680.5	117
	add	rpcs	16	9738	2.555	1487.8	0.034237	3801.33	9712.4	107
	add	pparch	16	10860	1.089	2918.1	0.015155	3177.81	3460.64	185
	add	csm	16	15156	1.386	2132.2	0.028682	2955.23	4095.95	172
	add	clf	16	12249	1.2	2534.1	0.021162	3040.92	3649.1	204
	add	cla	16	14946	1.2	2369.1	0.023708	2842.92	3411.5	246
	add	bk	16	10704	1.075	2548.2	0.014631	2739.31	2944.76	183
	add	rpl	32	26724	3.289	1974.2	0.123068	6493.14	21355.9	414
	add	rpcs	32	25419	3.36	2226.7	0.117264	7481.71	25138.5	333
	add	pparch	32	25992	1.276	4848.4	0.042852	6186.56	7894.05	442
	add	csm	32	34668	1.691	3446.9	0.079046	5828.71	9856.35	387
	add	clf	32	25878	1.349	4438.4	0.049069	5987.4	8077	418
	add	cla	32	26163	1.597	3518.8	0.054792	5619.52	8974.37	427
	add	bk	32	22146	1.355	4338.3	0.037029	5878.4	7965.23	387
	add	rpl	64	60516	5.202	2567.3	0.464148	13355.1	69473.2	945
	add	rpcs	64	57138	4.984	2727.7	0.402762	13594.9	67757	791
	add	pparch	64	48426	1.587	8061.8	0.094666	12794.1	20304.2	838
	add	csm	64	77616	2.076	5712.4	0.213392	11858.9	24619.1	866
	add	clf	64	51528	1.665	7434.4	0.123237	12378.3	20609.9	831
	add	cla	64	53151	1.894	5926.9	0.135023	11225.5	21261.1	864
	add	bk	64	49878	1.601	8104	0.096809	12974.5	20772.2	871
	add	rpl	128	103491	15.475	1776.6	2.32651	27492.9	425453	1555
	add	rpcs	128	123126	10.101	2860.7	1.7705	28895.9	291877	1754
	add	pparch	128	108297	1.879	14586	0.252068	27407.1	51497.9	1852
	add	csm	128	171012	2.306	10992	0.518919	25347.6	58451.6	1898
	add	clf	128	114198	1.984	13687	0.314325	27155	53875.5	1847
	add	cla	128	96987	2.161	10868	0.274425	23485.7	50752.6	1569
	add	bk	128	106122	1.92	14638	0.247507	28105	53961.6	1851

Figure B.6: Results for Unpipelined Implementation of Adder in 250nm Technology

Technology	Implementation	Types	Bit	Area	Delay	Power	Leakage	Energy	EDP	Total Gates	Register Count	Gate Count
DW_250	mult	csa	16	154440	4.526	83069	0.988524	375970	1.70E+06	1813	87	1726
	mult	pparch	16	147618	3.34	122010	0.731928	407513	1.36E+06	1904	46	1858
	mult	nbw	16	180261	2.217	139800	0.590032	309937	687130	2356	86	2270
	mult	wall	16	180480	2.442	161380	0.656825	394090	962368	2536	81	2455
	mult	csa	32	595152	8.403	252560	6.97945	2.12E+06	1.78E+07	6511	363	6148
	mult	pparch	32	527592	5.631	333330	4.50035	1.88E+06	1.06E+07	6900	91	6809
	mult	nbw	32	566262	3.205	464600	2.65031	1489043	4.77E+06	7346	181	7165
	mult	wall	32	568053	3.126	460750	2.60052	1.44E+06	4.50E+06	7342	181	7161
	mult	csa	64	2317194	16.088	813450	51.5701	1.31E+07	2.11E+08	24310	1451	22859
	mult	pparch	64	1896498	7.15	1060900	20.2545	7585435	5.42E+07	24610	179	24431
	mult	nbw	64	1897812	4.188	1294100	11.3972	5.42E+06	2.27E+07	23215	428	22787
	mult	wall	64	1899465	4.278	1278100	11.5934	5.47E+06	2.34E+07	23455	428	23027

Figure B.7: Results for Two-Stage Pipelined Implementation of Multiplier in 250nm Technology

Technology	Implementation	Types	Bit	Area	Delay	Power	Leakage	Energy	EDP	Gate Count
DW_250	mult	csa	16	103794	7.235	2247.5	1.10435	16260.7	117646	919
	mult	pparch	16	119685	3.564	2386.8	0.626444	8506.56	30317.4	1471
	mult	nbw	16	120069	3.994	2075.4	0.703463	8289.15	33106.9	1399
	mult	wall	16	112578	4.035	2447.9	0.66295	9877.28	39854.8	1328
	mult	csa	32	399018	13.922	4226.8	8.10706	58845.5	819247	3277
	mult	pparch	32	400446	5.694	4008.7	3.35155	22825.5	129968	4453
	mult	nbw	32	375786	5.377	4470.1	2.88422	24035.7	129240	3804
	mult	wall	32	375966	5.383	4765.1	2.88949	25650.5	138077	3804
	mult	csa	64	1525113	28.987	11230	64.3685	325524	9.44E+06	11116
	mult	pparch	64	1279794	7.095	7365.2	13.0186	52256.1	370757	11466
	mult	nbw	64	1331481	7.884	7235	14.885	57040.7	449709	12083
	mult	wall	64	1320645	7.455	8341.5	13.9431	62185.9	463596	11810

Figure B.8: Results for Unpipelined Implementation of Multiplier in 250nm Technology

VITA

VIJAYA CHANDRA KURAPATI

Candidate for the Degree of

Master of Science

Thesis: ANALYSIS OF IP BASED IMPLEMENTATIONS OF ADDERS AND MULTIPLIERS IN SUBMICRON AND DEEP SUBMICRON TECHNOLOGIES

Major Field: Electrical and Computer Engineering

Biographical:

Education:
Completed the requirements for the Master of Science in Electrical and Computer Engineering at Oklahoma State University, Stillwater, Oklahoma in December, 2008.

Education:
Completed the requirements for the Bachelor of Technology in Electrical and Electronics Engineering from Jawaharlal Nehru Technological University, Hyderabad, India in May, 2006.

Name: VIJAYA CHANDRA KURAPATI Date of Degree: December, 2008

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: ANALYSIS OF IP BASED IMPLEMENTATIONS OF ADDERS AND MULTIPLIERS IN SUBMICRON AND DEEP SUBMICRON TECHNOLOGIES

Pages in Study: 64 Candidate for the Degree of Master of Science

Major Field: Electrical and Computer Engineering

Scope and Method of Study:

Datapath is at the heart of the microprocessor whose performance is a key factor which determines the performance of the processor. Adders and multipliers are the key elements in the datapath which usually are a measure of the performance of the datapath. So, with scaling of MOS transistors down into the deep submicron regime, it is necessary to investigate the performance of these key elements at such small device sizes. This thesis focuses on investigating the performance of existing architectures of adders and multipliers in the submicron and deep submicron technologies at the physical implementation level. Also, an effort has been made to investigate the performance of pipelined implementations of these architectures. Verilog HDL instantiations of adders and multipliers that are available with the DesignWare Building Block IP of Synopsys have been utilized in this thesis. The entire process of the design right from synthesis of the design down to power analysis of the design has been carried out using various EDA tools and has been automated using scripts written in TCL.

Findings and Conclusions:

Various architectures of adders and multipliers available with the DesignWare Building Block IP were implemented in different technologies for various bit widths. Adders and multipliers were implemented in unpipelined and two-stage pipelined configurations. These design implementations were analyzed for key parameters of total dynamic power, leakage power, Energy-Delay product, delay and area at various bit widths. Using the results obtained optimal implementations of adders and multipliers for before mentioned key parameters were summarized. These results were consistent for all implemented bit widths in all implemented technologies. Also, the leakage power was seen to contribute a higher percentage to the value of the average power dissipation in deep submicron technologies when compared to submicron technologies.

ADVISER'S APPROVAL: Dr. James E. Stine, Jr.