DESIGN AND IMPLEMENTATION OF FASTER AND LOW POWER MULTIPLIERS

A THESIS SUBMITTED IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

Bachelors of Technology
In
Electronics & Communication Engineering

By
PARTHA SARATHI MOHANTY

Department Of Electronics and Communication Engineering

National Institute Of Technology

Rourkela

2009

DESIGN AND IMPLEMENTATION OF FASTER AND LOW POWER MULTIPLIERS

A THESIS SUBMITTED IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

Bachelors of Technology
In
Electronics & Communication Engineering

By
PARTHA SARATHI MOHANTY

Under the guidance of

Prof. K.K.Mahapatra
Electronics and Communication Engineering
National Institute of Technology,Rourkela

Department Of Electronics and Communication Engineering

National Institute Of Technology

Rourkela

2009

National Institute Of Technology
Rourkela

CERTIFICATE

This is to certify that the thesis entitled "**DESIGN AND IMPLEMENTATION OF FASTER AND LOW POWER MULTILPLIERS**" submitted by Mr. Partha Sarathi Mohanty, Final year student of Electronics and communication Engineering, Roll No.:10509019 in partial fulfillments for the requirements for the award of Bachelor of Technology Degree in Electronics and Communication Engineering at National Institute of Technology, Rourkela is an authentic work carried out by him under my supervision and guidance.

To the best of my knowledge, the matter embodied in thesis has not been submitted to any other university/ Institute for the award of any degree or Diploma.

Date:

Prof. K.K.Mahapatra
Dept. of E.C.E
National Institute of Technology
Rourkela - 769008

ACKNOWLEDGEMENT

I would like to articulate my profound gratitude and indebtedness to my project guide Prof.Dr.K.K.Mahapatra who has always been a constant motivation and guiding factor throughout the project time in and out as well. It has been a great pleasure for me to get an opportunity to work under him and complete the project successfully.

I wish to extend my sincere thanks to Prof.Dr.S.K.Patra, Head of our Department, for approving our project work with great interest.

I would like to mention Mr. J.K.Das for his cooperation and constantly rendered assistance.

An undertaking of this nature could never been attempted with our reference to and inspiration from the works of others whose details are mentioned in references section. I acknowledge my indebtedness to all of them. Last but not the least, my sincere thanks to all my friends who have patiently extended all sorts of help for accomplishing this undertaking.

Partha Sarathi Mohanty
Roll No – 10509019
Department of ECE
N.I.T, Rourkela

TABLE OF CONTENTS

ABSTRACT

A multiplier is one of the key hardware blocks in most digital and high performance systems such as FIR filters, digital signal processors and microprocessors etc. With advances in technology, many researchers have tried and are trying to design multipliers which offer either of the following- high speed, low power consumption, regularity of layout and hence less area or even combination of them in multiplier. Thus making them suitable for various high speed, low power, and compact VLSI implementations. However area and speed are two conflicting constraints. So improving speed results always in larger areas. So here we try to find out the best trade off solution among the both of them.

Generally as we know multiplication goes in two basic steps. Partial product and then addition. Hence in this paper we have first tried to design different adders and compare their speed and complexity of circuit i.e. the area occupied. And then we have designed Wallace tree multiplier then followed by Booth's Wallace multiplier and have compared the speed and Power consumption in them.

While comparing the adders we found out that Ripple Carry Adder had a smaller area while having lesser speed, in contrast to which Carry Select Adders are high speed but posses a larger area. And a Carry Look Ahead Adder is in between the spectrum having a proper trade off between time and area complexities.

After designing and comparing the adders we turned to multipliers. Initially we went for Parallel Multiplier and then Wallace Tree Multiplier. In the mean time we learned that delay amount was considerably reduced when Carry Save Adders were used in Wallace Tree applications. Then we turned to Booths Multiplier and designed Radix-4 modified booth multiplier and analyzed the performance of all the multipliers.

After that we turned to different methods of power optimization, of which we could only complete a few like we went for designing different recoding schemes and their corresponding partial product generator scheme. After that we designed these recoders and PP generators and found out the time delays and area covered and power consumed by each scheme. We took into consideration that since all the PP generators take a huge amount of area we need to go for simplest of the designs for them and also side by side we need to ensure that we don't have much switching actions in the circuit.

After this we even modified one of the recoding schemes to lower the delay and power required by the circuit.

The result of our project helps us to make a proper choice of different multipliers in fabricating in different arithmetic units as well as making a choice among different adders in different digital applications according to requirements. All the programs and results have been given in the following sections.

Further work on Low Power Techniques on different multipliers needs to be done in order to make us choose a proper multiplier in accordance with the requirements by making the best possible trade off choice between Speed and Power in different circumstances.

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1 Motivation

1.2 Power Optimization

1.3 Low Power Multiplier Design

1.4 Language and tools used

1.5 Research Approach

1.1 MOTIVATION

As the scale of integration keeps growing, more and more sophisticated signal processing systems are being implemented on a VLSI chip.These signal processing applications not only demand great computation capacity but also consume considerable amount of energy. While performance and Area remain to be the two major design tolls, power consumption has become a critical concern in today's VLSI system design[]. The need for low-power VLSI system arises from two main forces. First, with the steady growth of operating frequency and processing capacity per chip, large currents have to be delivered and the heat due to large power consumption must be removed by proper cooling techniques. Second, battery life in portable electronic devices is limited. Low power design directly leads to prolonged operation time in these portable devices.

Multiplication is a fundamental operation in most signal processing algorithms. Multipliers have large area, long latency and consume considerable power. Therefore low-power multiplier design has been an important part in low-power VLSI system design. There has been extensive work on low-power multipliers at technology, physical, circuit and logic levels. A system's performance is generally determined by the performance of the multiplier because the multiplier is generally the slowest element in the system. Furthermore, it is generally the most area consuming. Hence, optimizing the speed and area of the multiplier is a major design issue. However, area and speed are usually conflicting constraints so that improving speed results mostly in larger areas. As a result, a whole spectrum of multipliers with different area-speed constraints has been designed with fully parallel.

Fully Parallel Multipliers at one end of the spectrum and fully serial multipliers at the other end. In between are digit serial multipliers where single digits consisting of several bits are operated on. These multipliers have moderate performance in both speed and area. However, existing digit serial multipliers have been plagued by complicated switching systems and/or irregularities in design. Radix 2^n multipliers which operate on digits in a parallel

fashion instead of bits bring the pipelining to the digit level and avoid most of' the above problems. These structures are iterative and modular. The pipelining done at the digit level brings the benefit of constant operation speed irrespective of the size of' the multiplier. The clock speed is only determined by the digit size which is already fixed before the design is implemented.

1.2 POWER OPTIMIZATION

Power refers to number of Joules dissipated over a certain amount of time whereas energy is the measure of the total number of Joules dissipated by a circuit.

In digital CMOS design, the well-known *power-delay product* is commonly used to assess the merits of designs. In a sense, this can be shown as *power* × *delay* = *(energy/delay)* × *delay* = *energy,* which implies delay is irrelevant [].

1.3 LOW-POWER MULTIPLIER DESIGN

Multiplication consists of three steps: generation of partial products or (PPG), reduction of partial products (PPR), and finally carry-propagate addition (CPA).In general there are sequential and combinational multiplier implementations. We only consider combinational case here because the scale of integration now is large enough to accept parallel multiplier implementations in digital VLSI systems. Different multiplication algorithms vary in the approaches of PPG, PPR, and CPA. For PPG, radix-2 is the easiest. To reduce the number of PPs and consequently reduce the area/delay of PP reduction, one operand is usually recoded into high-radix digit sets. The most popular one is the radix-4 digit set **{**-2,-1, 0, 1, 2**}.** For PPR, two alternatives exist: reduction by rows , performed by an array of adders, and reduction by columns, performed by an array of counters. The final CPA requires a fast adder scheme because it is on the *critical path*. In some cases, final CPA is postponed if it is advantageous to keep redundant results from PPG for further arithmetic operations.

1.4 LANGUAGE AND TOOLS USED

We used **XILINX ISE v 10.2** for our programming. We considered **VHDL** as our primary language. For test bench waveforms also we also used Xilinx to write our own test benches. Model Synthesis Map report all features in Xilinx helped us a lot.

We used **Xilinx's XPower Estimator (XPE)** tool in order to calculate power consumed in any arithmetic circuit. For calculation of power using **Xilinx's XPE** we need to generate the **map report** file in XILINX which will be saved in the same directory with an extension "**.mrp**". But in the later part of the project we used **SYNOPSIS** tool for finding out Power and delay and Area calculations

1.5 RESEARCH APPROACH

The basic motive of our project was to study and develop an Efficient Fast and Low Power Multiplier. As the name suggests we had to go for faster and low power factor optimization simultaneously. We know that the basic building block of a multiplier is ADDER circuit. Hence we turned our focus into **The ADDERS** first. We studied the area occupied and the time delay consumed by different adders and found out a proper relation between time and area complexity of all the adders under consideration. We generated a factor **Area-Delay product** which helped us to properly understand the Area and Delay trade-off perfectly and hence choose the best adder for appropriate circumstances.

Then we turned our focus into Multipliers. In Multipliers we studied different multipliers writing programs, verifying waveforms and then finally calculating number of CLBs, LUTs required along with Power consumed in the circuit. After knowing all this we also calculated delay for different multipliers which helped us to determine the best multiplier. Radix-4 Booth Multiplier was best Multiplier among all with less power consumption and proper Area Delay trade-off. Our future work will be to optimize power Consumed by different multipliers there by reducing number of gates used and area occupied by them.

Chapter 2

THE ADDERS

THE ADDERS

Addition is the most common and often used arithmetic operation on microprocessor, digital signal processor, especially digital computers. Also, it serves as a building block for synthesis all other arithmetic operations. Therefore, regarding the efficient implementation of an arithmetic unit, the binary adder structures become a very critical hardware unit.

In any book on computer arithmetic, someone looks that there exists a large number of different circuit architectures with different performance characteristics and widely used in the practice. Although many researches dealing with the binary adder structures have been done, the studies based on their comparative performance analysis are only a few.

In this project, qualitative evaluations of the classified binary adder architectures are given. Among the huge member of the adders we wrote VHDL (Hardware Description Language) code for Ripple-carry, Carry-select and Carry-look ahead to emphasize the common performance properties belong to their classes. In the following section, we give a brief description of the studied adder architectures.

With respect to asymptotic delay time and area complexity, the binary adder architectures can be categorized into four primary classes as given in Table 2.1. The given results in the table are the highest exponent term of the exact formulas, very complex for the high bit lengths of the operands.

The first class consists of the very slow ripple-carry adder with the smallest area. In the second class, the carry-skip, carry-select adders with multiple levels have small area requirements and shortened computation times. From the third class, the carry-look ahead adder and from the fourth class, the parallel prefix adder represents the fastest addition schemes with the largest area complexities.

Complex (A)	*Delay (T)*	*Product (AxT)*	*Adder Class Schemes*
O(n)	*O(n)*	*O(n2)*	*Ripple-carry (1)*
O(n)	$O(n^{1/*1+1})$	$O(n^{1+2/+1})$	*Carry select (2)* *Carry-skip (2)* *Carry-Inc (2)*
O(n)	*O(logn)*	*O(nlogn)*	*Carry look ahead(3)*

**1 denotes the LEVEL number*

TABLE 2.1

Categorization Of adders w.r.t delay time and capacity

2.1 Ripple Carry Adders (RCA)

The well known adder architecture, ripple carry adder is composed of cascaded full adders for n-bit adder, as shown in figure.1.It is constructed by cascading full adder blocks in series. The carry out of one stage is fed directly to the carry-in of the next stage. For an n-bit parallel adder it requires n full adders.

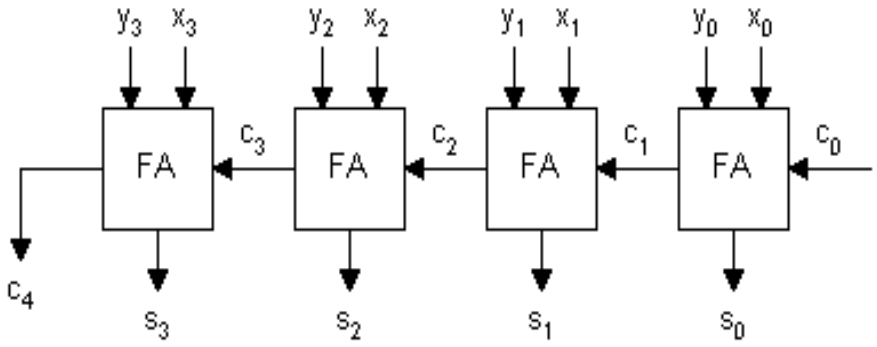


FIGURE 2.1 A 4-bit Ripple Carry Adder

- Not very efficient when large number bit numbers are used.
- Delay increases linearly with bit length.

2.1.1 *Delay*

Delay from Carry-in to Carry-out is more important than from A to carry-out or carry-in to SUM, because the carry-propagation chain will determine the latency of the whole circuit for a Ripple-Carry adder. Considering the above worst-case signal propagation path we can thus write the following equation.

For a **k-bit RCA** worst case path delay is

$$\mathbf{T_{RCA\text{-}k\ bit} = T_{FA}(x_0, y_0\ c_0) + (k\text{-}2)^* T_{FA}(C_{in}\ Ci) + T_{FA}(C_{in}\ S_{k-1})}\ .$$

2.1.2 *Logic equations*

$g_i = a_i b_i$ $\qquad$ $p = a_i$ **xor** b_i.

$C_{i+1} = g_i + p_i c_i$ $\qquad$ $Si = p_i$ **xor** c_i.

2.1.3 *Complexity and Delay for n-bit RCA structure*

$$\mathbf{A_{RCA} = O(n) = 7n}$$

$$\mathbf{T_{RCA} = O(n) = 2n}$$

2.2 Carry Select Adders (CSLA)

In Carry select adder scheme, blocks of bits are added in two ways: one assuning a carry-in of 0 and the other with a carry-in of 1.This results in two precomputed sum and carry-out signal pairs **($s^0_{i-1:k}$, c^0_i ; $s^1_{i-1:k}$, c^1_i) ,** later as the block's true carry-in (c_k) becomes known , the correct signal pairs are selected. Generally multiplexers are used to propagate carries.

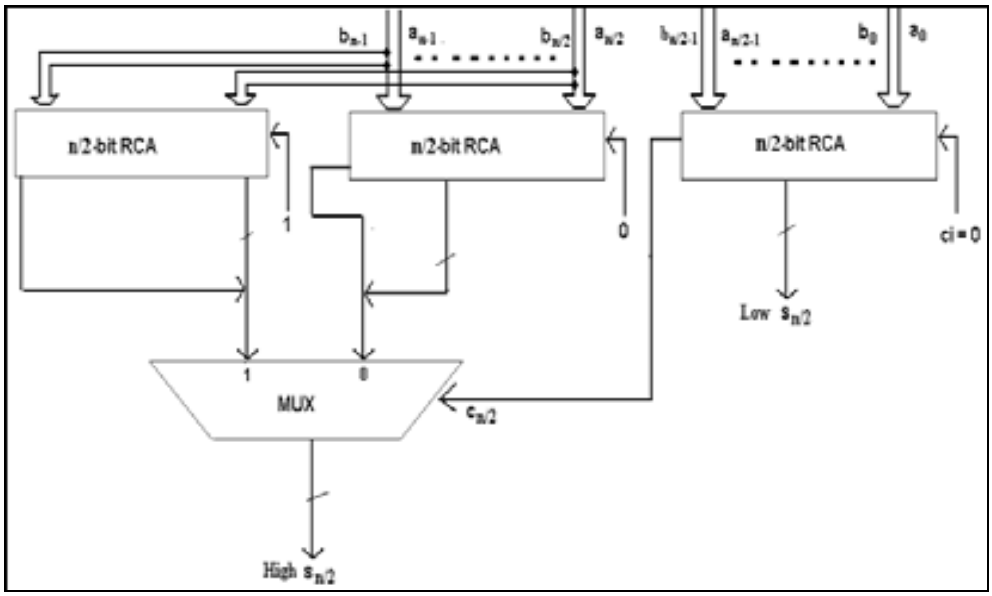


FIGURE 2.2 A Carry Select Adder with 1 level using n/2- bit RCA

- Because of multiplexers larger area is required.
- Have a lesser delay than Ripple Carry Adders (**half delay of RCA**).
- Hence we always go for Carry Select Adder while working with smaller no of bits.

2.2.1 *Logic equations*

$$s_{i-1:k} = \bar{c}_k \, s^0_{i-1:k} + c_k \, s^1_{i-1:k}$$

$$c_i = \bar{c}_k \, c^0_i + c_k \, c^1_i$$

2.2.2 *Complexity and Delay for n-bit CSLA structure*

$$\mathbf{A_{CSLA} = O\,(n) = 14n}$$

$$\mathbf{T_{CSLA} = O\,(n^{1/*l+1}) = 2.8n^{1/2}.}$$

2.3 Carry Look Ahead Adders (CLA)

Carry Look Ahead Adder can produce carries faster due to carry bits generated in parallel by an additional circuitry whenever inputs change. This technique uses carry bypass logic to speed up the carry propagation.

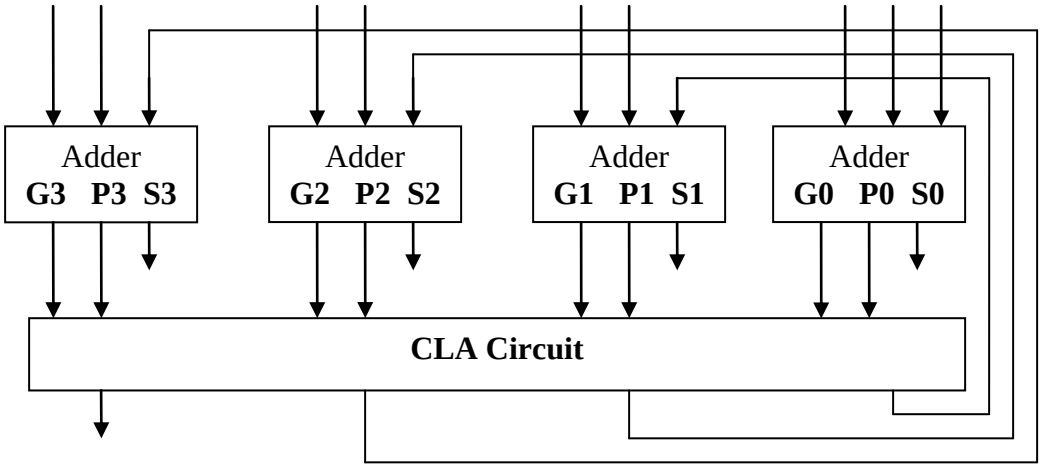


FIGURE 2.3 4-BIT CLA Logic equations

Let $\mathbf{a_i}$ and $\mathbf{b_i}$ be the augends and addend inputs, $\mathbf{c_i}$ the carry input, $\mathbf{s_i}$ and $\mathbf{c_{i+1}}$, the sum and carry-out to the $\mathbf{i^{th}}$ bit position. If the auxiliary functions, $\mathbf{p_i}$ and $\mathbf{g_i}$ called the *propagate* and *generate* signals, the sum output respectively are defined as follows.

$$p_i = a_i + b_i \qquad g_i = a_i b_i$$

$$s_i = a_i \ \mathbf{xor}\ b_i \ \mathbf{xor}\ c_i \qquad c_{i+1} = g_i + p_i c_i \ .$$

- As we increase the no of bits in the Carry Look Ahead adders, the complexity increases because the no. of gates in the expression $\mathbf{C_{i+1}}$ increases. So practically its not desirable to use the traditional CLA shown above because it increase the Space required and the power too.
- Instead we will use here Carry Look Ahead adder (less bits) in levels to create a larger CLA. Commonly smaller CLA may be taken as a 4-bit CLA. So we can define ***carry look ahead*** over a group of 4 bits. Hence now we redefine terms ***carry generate*** as [***Group Generated Carry***] $\mathbf{g_{[\,i,i+3\,]}}$ and ***carry propagate*** as [***Group Propagated Carry***] $p_{[\,i,i+3\,]}$ which are defined below.

2.3.1 *Redefined Equations:*

$$g_{[i,i+3]} = g_{i+3} + g_{i+2}\,p_{i+3} + g_{i+1}\,p_{i+2}\,p_{i+3} + g_{[i}\,p_{i+1}\,p_{i+2}\,p_{i+3}$$

$$p_{[i,i+3]} = p_i\,p_{i+1}\,p_{i+2}\,p_{i+3}$$

Now the modified block diagram for the Carry Look ahead Adder (8-bit) using levels (of 4-bit CLA) will be as block diagram below.

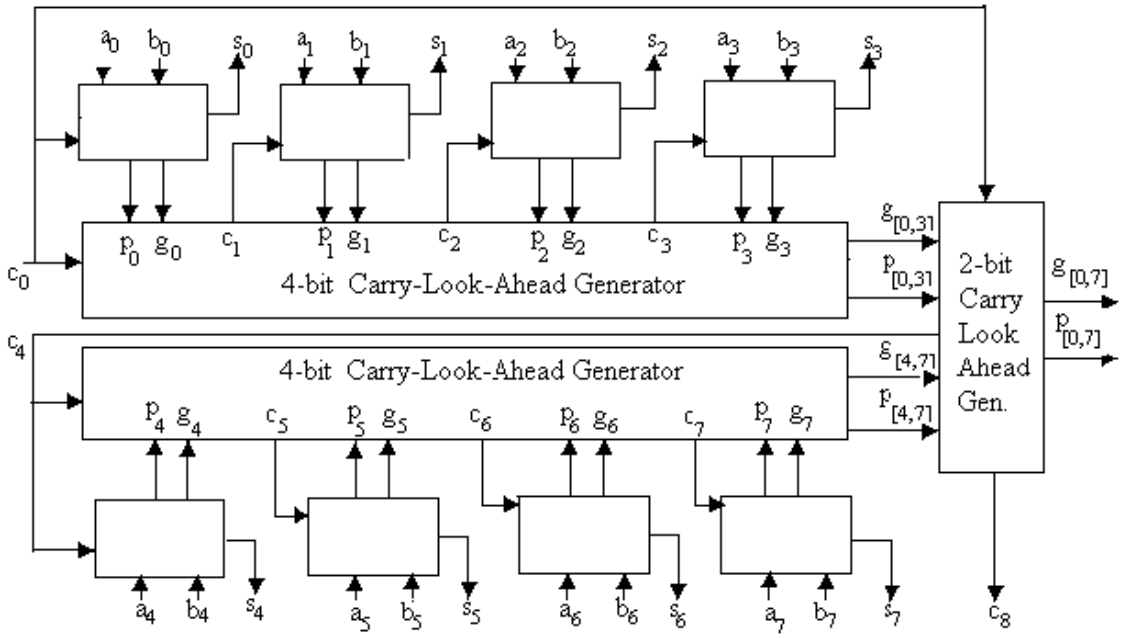


FIGURE 2.4 8-BIT CARRY LOOK AHEAD GENERATOR (using 2-bit CLA)

2.3.2 *Complexity and Delay for n-bit CLA structure*

$$A_{CLA} = O(n) = 14n$$

$$T_{CLA} = O(\log n) = 4\log_2 n.$$

2.4 ANALYSIS OF ADDERS

In our project we compared 3- different adders ***Ripple Carry Adders***, ***Carry Select Adders*** and ***the Carry Look Ahead Adders***. The basic purpose of our experiment was to know the time and power trade-offs between different adders whish will give us a clear picture of which adder suits best in which type of situation during design process. Hence below we present both the theoretical and practical comparisons of all the three adders whish were taken into consideration.

Adder Name	**Complex (A_x)**	**Area for n-bit**
Ripple Carry Adder(RCA)	O(n)	**7n**
Carry Select Adder(CSLA)	O(n)	**14n**
Carry Look Ahead Adder(CLA)	O(n)	**4n**

Table 2.1 Theoretical Comparison of Area Occupied (A_x)

Adder Name	**Complex (T)**	**Delay for n-bit**
Ripple Carry Adder(RSA)	O(n)	**2n**
Carry Select Adder(CSLA)	$O(n^{1/*\|+1})$	$\mathbf{2.8(n)^{1/2}}$
Carry Look Ahead Adder(CLA)	$O(\log_2 n)$	$\mathbf{4\log_2 n}$

Table 2.2 Theoretical Comparison of Time Required (T)

Adder Name	**Delay for n-bit**	**Area for n-bit**	**Area Delay Product**
Ripple Carry Adder(RCA)	**2n**	**7n**	$\mathbf{14n^2}$
Carry Select Adder(CSLA)	$\mathbf{2.8(n)^{1/2}}$	**14n**	$\mathbf{39.6(n)^{3/2}}$
Carry Look Ahead Adder(CLA)	$\mathbf{4\log_2 n}$	**4n**	$\mathbf{16n \log_2 n}$

Table 2.3 Theoretical Area Delay Product (AxT)

Adder Name	**Complex (A)**	**Delay for 8-bit)**
Ripple Carry Adder(RCA)	O(n)	**20.8 ns**
Carry Select Adder(CSLA)	$O(n^{1/*l+1})$	**12.8ns**
Carry Look Ahead Adder(CLA)	$O(\log_2 n)$	**17.6ns**
Carry Look Ahead Adder (**using 2 4-bits in levels)**		**14.8ns (close to CSLA)**

Table 2.4 Comparison Of Time Required (Simulated Value)

2.5 DISCUSSIONS

As can be seen above we have stated the theoretical comparison of **AREA** required and both the theoretical and simulated value of **TIME** required. The values stated above are the values for **8-bit adders.** So analyzing the above facts we reached at the following conclusions about different adders and intelligent use of them in different circumstances according to the SPACE TIME trade-off. The results can be summarized as follows.

- Regarding the ***circuit area complexity*** in the adder architectures, the ***ripple-carry adder*** (RCA) in the first class is the most efficient one, but the ***carry select adder*** (CSLA) in the fourth class with highest complexity is the least efficient one.

- Considering the ***circuit delay time***, ***Carry Select*** Adder (CSLA) is the fastest one for every n-bit length, so has the shortest delay. Otherwise, ***Ripple Carry Adder*** (RCA) is the slowest one, due to the long carry propagation.

- We defined a term ***Area-Delay Product*** which gave us the clear picture of the *space-time tradeoff.* It is worthy to note that while we consider all the adders discussed above ***Ripple Carry adders*** and ***Carry Select Adders*** are the two sides of the spectrum. Because, while Ripple Carry Adders have a ***smaller area and lesser speed***, in contrast to which Carry Select adders have high speed (***nearly twice the speed f Ripple Carry Adders*****)** and occupy a larger area. But Carry Look Ahead Adder (CLA) has a proper balance between both the Area occupied and Time required. Hence among the three, ***Carry Look Ahead Adder*** has the ***least AREA DELAY PRODUCT.*** Hence we should use Carry Look ahead Adders when it comes to optimization with both Area and Time. For an instance, the last stage of the Wallace tree Adder in Booth multiplier is a Carry look Ahead Adder.

Chapter 3

THE MULTIPLIERS

3.1 Wallace Tree Multipliers

3.2 Booth's Multiplier (radix-2)

3.3 Booth's Multiplier (radix-4)

3.4 Analysis of Multipliers

3.1 THE WALLACE TREE MULTIPLIER

The Wallace tree multiplier is considerably faster than a simple array multiplier because its height is logarithmic in word size, not linear. However, in addition to the large number of adders required, the Wallace tree's wiring is much less regular and more complicated. As a result, Wallace trees are often avoided by designers, while ***design complexity*** is a concern to them.

Wallace tree styles use a ***log-depth*** tree network for reduction. Faster, but irregular, they trade ease of layout for speed. Wallace tree styles are generally avoided for low power applications, since excess of wiring is likely to consume extra power.

While subsequently faster than Carry-save structure for large bit multipliers, the Wallace tree multiplier has the disadvantage of being very irregular, which complicates the task of coming with an efficient layout.

The Wallace tree multiplier is a high speed multiplier. The summing of the partial product bits in parallel using a tree of carry-save adders became generally known as the "**Wallace Tree".** Three step processes are used to multiply two numbers.

- Formation of bit products.
- Reduction of the bit product matrix into a two row matrix by means of a carry save adder.
- Summation of remaining two rows using a faster Carry Look Ahead Adder (CLA).

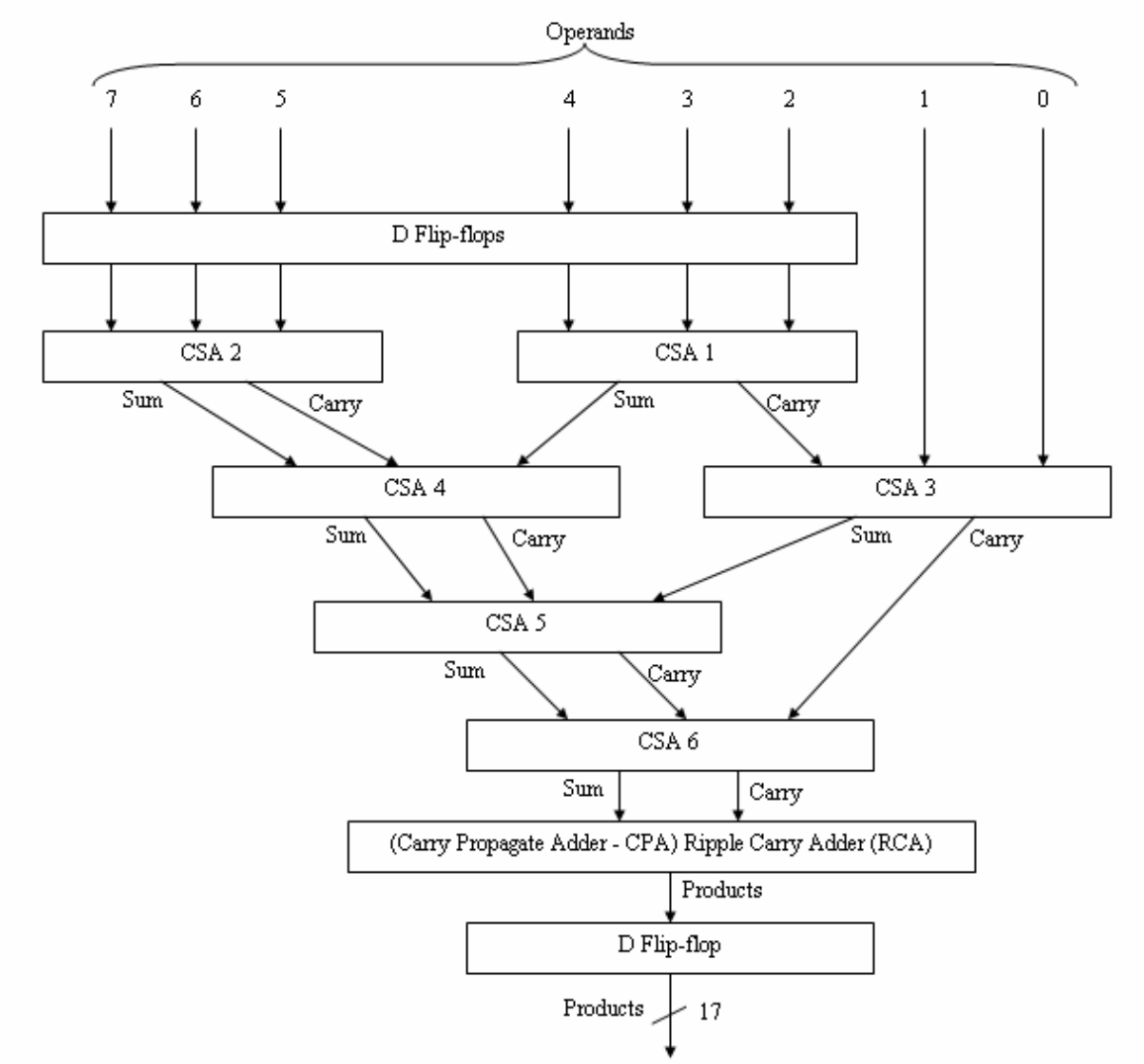


Figure 3.1 Wallace Tree Block Diagram

In order to design an **n-bit Wallace tree Multiplier** (Generic: =N) an algorithm was derived from the flow diagram developed below. The flow diagram below shows the intermediate state reductions of the multipliers are being done by Carry save adders and half adders while the final step additions being done by a Carry Look Ahead Adder. The flow diagram was done in Microsoft Excel sheet and Paint.

After generating the flow diagram for 8-bit × 8-bit we generalized the algorithm for n-bit and hence we designed a **GENERIC WALLACE TREE.** The Code for the same is in Appendix for programs. The algorithm developed for designing the Generic Wallace tree is given below.

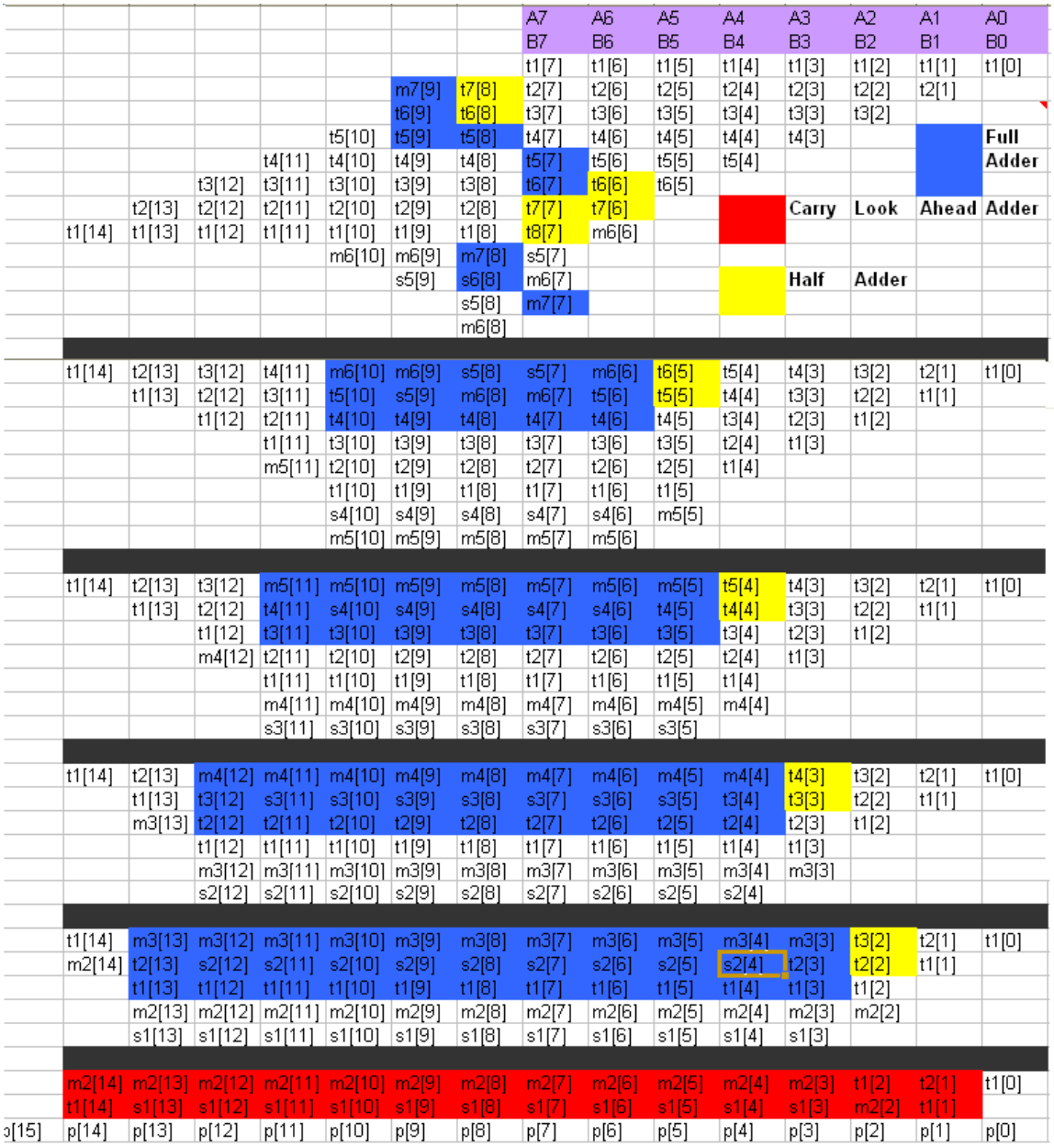


Figure 3.2 8-bit × 8-bit Wallace Tree Multiplier (Logarithmic Depth Hierarchy)

3.1.1 Algorithm For Wallace Tree Multiplier (Generic: N)

```
Input A: in STD_LOGIC_VECTOR (N-1 downto 0);
Input B: in STD_LOGIC_VECTOR (N-1 downto 0);
Output C: out STD_LOGIC_VECTOR (2*N-1 downto 0));

------Let half_adder & full_adder be two components
type array_signal is array(N downto 1) of
        STD_LOGIC_VECTOR(2*N-1 DOWNTO 0);

signal t,m,s :array_signal;
signal p,d :STD_LOGIC_VECTOR(2*N-1 downto 0);

for I in 0 to N-1
        for J in 0 to N-I-1
                t(I+1)(I+J)<=A(J) and B(I);
        end
end
for I in N-1 downto 0
        for J in N-I to N-1
                t(N-I)(I+J)<=A(J) and B(I);
        end
end

---Initial Half adder additions
HA1: half_adder port map(t(N-2)(N),t(N-1)(N),s(N-2)(N),m(N-1)(N+1));
HA2: half_adder port map(t(N-1)(N-1),t(N)(N-1),m(N-1)(N-1),m(N-1)(N));

for I in 1 to N-2
    for J in (-I) to (I)
        if(I < N-2)
                if(J=I or J=(-I))
full_adder port map(m(N-I)(N+J),t(N-I-1)(N+J),t(N-I-2)(N+J),
                                        s(n-I-2)(N+J),m(N-I-1)(N+J+1));
```

```
                    end
                    if((J /= I and J /= (-I)))
full_adder port map(m(N-I)(N+J),s(N-I-1)(N+J),t(N-I-2)(N+J),
                                        s(n-I-2)(N+J),m(N-I-1)(N+J+1));
                    end
              end
             if(I = N-2)
                    if (J = I)
full_adder port map(m(N-I)(N+J),t(N-I-1)(N+J),d(N+J),p(N+J),p(N+J+1));
                    end
                    if (J = (-I))
    full_adder port map(m(N-I)(N+J),t(N-I-1)(N+J),d(N+J),p(N+J),d(N+J+1));
                    end
                    if ((J/=I and J/=(-I)))
    full_adder port map(m(N-I)(N+J),s(N-I-1)(N+J),d(N+J),p(N+J),d(N+J+1));
                    end
              end
     end
              if(I < N-2)
    half_adder port map(t(N-I)(N-I-1),t(N-I-1)(N-I-1),m(N-I-1)(N-I-1),
                                        m(n-I-  1)(N-I));
              end
             if(I = N-2) generate
    half_adder port map(t(N-I)(N-I-1),t(N-I-1)(N-I-1),p(N-I-1),d(N-I));
     end
  end
p(0)<=t(1)(0);
C<=p;
```

3.2 THE BOOTH'S MULTIPLIER

Though Wallace Tree multipliers were faster than the traditional Carry Save Method, it also was very irregular and hence was complicated while drawing the Layouts. Slowly when multiplier bits gets beyond 32-bits large numbers of logic gates are required and hence also more interconnecting wires which makes chip design large and slows down operating speed

Booth multiplier can be used in different modes such as **radix-2, radix-4, radix-8** etc. But we decided to use **Radix-4 Booth's Algorithm** because of number of Partial products is reduced to ***n/2***.

3.2.1 BOOTH MULTIPLICATION ALGORITHM (radix – 4)

One of the solutions realizing high speed multipliers is to enhance parallelism which helps in decreasing the number of subsequent calculation stages. The Original version of Booth's multiplier (Radix – 2) had two drawbacks.

- The number of add / subtract operations became variable and hence became inconvenient while designing Parallel multipliers.

- The Algorithm becomes inefficient when there are isolated 1s .

These problems are overcome by using Radix 4 Booth's Algorithm which can scan strings of three bits with the algorithm given below. The design of Booth's multiplier in this project consists of four Modified Booth Encoded (MBE), four sign extension corrector, four partial product generators (comprises of 5:1 multiplexer) and finally a Wallace Tree Adder. This Booth multiplier technique is to increase speed by reducing the number of partial products by half. Since an 8-bit booth multiplier is used in this project, so there are only four partial products that need to be added instead of eight partial products generated using conventional multiplier. The architecture design for the modified Booths Algorithm used in this project is shown below.

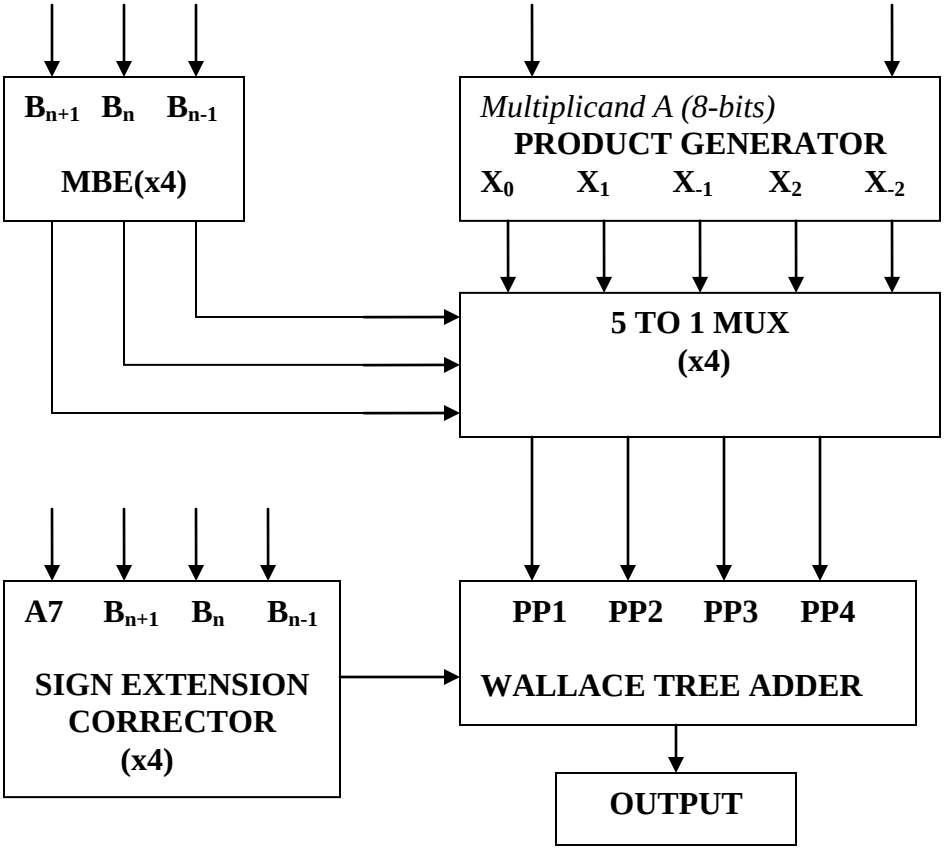


Figure 3.3 Architecture of designed Booth Multiplier in the Project.

3.2.2 MODIFIED BOOTH ENCODER (MBE)

Modified Booth encoding is most often used to avoid variable size partial product arrays. Before designing a MBE, the multiplier B has to be converted into a Radix-4 number by dividing them into three digits respectively according to Booth Encoder Table given afterwards. Prior to convert the multiplier, a zero is appended into the Least Significant Bit (LSB) of the multiplier. The figure above shows that the multiplier has been divided into four partitions and hence that mean four partial products will be generated using booth multiplier approach I nstead of eight partial products being generated using conventional multiplier.

$$Z_n = -2 * B_{n+1} + B_n + B_{n-1}$$

Lets take an example of converting an 8-bit number into a Radix-4 number. Let the number be **-36 = 1 1 0 1 1 1 0 0**. Now we have to append a '**0**' to

the LSB. Hence the new number becomes a 9-digit number, that is **1 1 0 1 1 1 0 0** 0. This is now further encoded into Radix-4 numbers according to the following given table. Starting from right we have **0*Multiplicand, -1*Multiplicand, 2*Multiplicand, -1*Multiplicand.**

B_{n+1}	B_n	B_{n-1}	Z_n	Partial Product	1M	2M	3M
0	0	0	0	0	1	1	0
0	0	1	1	1×Multiplicand	0	1	0
0	1	0	1	1×Multiplicand	0	1	0
0	1	1	2	2×Multiplicand	1	0	0
1	0	0	-2	-2×Multiplicand	1	0	1
1	0	1	-1	-1×Multiplicand	0	1	1
1	1	0	-1	-1×Multiplicand	0	1	1
1	1	1	0	0	1	1	0

Table 3.1
Modified Booth Encoder's table to generate M, 2M, 3M control signal

Table 3.1 shows B_{n+1}, B_n and Bn_{-1} which are three bits wide binary numbers of the multiplier Bin which B_{n+1} is the most significant bit (MSB) and B_{n-1} is the least significant bit (LSB). Z_n is representing the Radix-4 number of the 3-bit binary multiplier number. For example, if the 3-bit multiplier value is "111", so it means that multiplicand A will be 0.And it's the same for others either to multiply the multiplicand by -1, -2 and so on depending on 3 digit number. And thing to note is generated numbers are all of 9-bit.

From the table 4.1, the **M, 2M** and **3M** are the elect control signals for the partial product generator. It will determine whether the multiplicand is multiplied by 0,-1, 2 or -2. **M** and **2M** are designed as an active low circuit which means if let's say the multiplicand should be multiplied by 1 then the **M** select signal will be set to low "0" whereas If the multiplicand should be multiplied by 2 then the **2M** select signal will be set to low "0". The **3M** is representing the sign bit control signal and its active high circuit which means if the multiplicand should be multiplied by -1 or -2, then the sign, **3M** will be set to high "1".

3.2.3 PARTIAL PRODUCT GENERATOR (PPG)

Partial product generator is the combination circuit of the product generator and the 5 to 1 MUX circuit. Product generator is designed to produce the product by multiplying the multiplicand A by 0, 1, -1, 2 or -2. A 5 to 1 MUX is designed to determine which product is chosen depending on the M, 2M, 3M control signal which is generated from the MBE. For product generator, multiply by zero means the multiplicand is multiplied by “0”.Multiply by “1” means the product still remains the same as the multiplicand value. Multiply by “-1” means that the product is the two’s complement form of the number. Multiply by “-2” is to shift left one bit the two’s complement of the multiplicand value and multiply by “2” means just shift left the multiplicand by one place.

3.2.4 SIGN EXTENSION CORRECTOR

Sign Extension Corrector is designed to enhance the ability of the booth multiplier to multiply not only the unsigned number but as well as the signed number. As shown in Table 4.2 when bit 7 of the multiplicand A(A7) is zero(unsigned number) and B_{n+1} is equal to one, then sign E will have one value (become signed number for resulted partial product). It is the same when the bit 7 of the multiplicand A (A7) is one (signed number) and B_{n+1} is equal to zero, the sign E will have a new value. However when both the value of A7 and B_{n+1} are equal either to zero or one, the sign E will have a value zero(unsigned number). For the case when all three bits of the multiplier value B_{n+1}, B_n and B_{n-1} are equal to zero or one, the sign E will direct have a zero value independent to the A7 value. The table for the Sign Extension Corrector is shown below.

TABLE 3.2 (A) Sign E when A& is Zero

A7	B_{n+1}	B_n	B_{n-1}	E
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0

TABLE 3.2 (B) Sign E when A& is One

A7	B_{n+1}	B_n	B_{n-1}	E
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

3.2.5 WALLACE TREE ADDER

Wallace tree has been used in this project in order to accelerate multiplication by compressing the number of partial products. This design is done using half adders; Carry save adders and the Carry Look Ahead adders to speed up the multiplication. As shown in the figure below, since there are four sign extension values generated namely sign **1E, 2E, 3E** and **4E** for the partial product **PP1, PP2, PP3** and **PP4** respectively. The arrangement of total four partial product s is shown in the figure below. The second partial product had to be shifted left by two bits before adding to the first partial product. Hence the third will be shifted left by four where as for fourth it will be shifted left by six. Hence after proper arrangement all the four partial products will be added along with the sign extension.

					1BE	1E	1E	PP08	PP07	PP06	PP05	PP04	PP03	PP02	PP01	PP00	Partial Product 1	
				1	2BE	PP18	PP17	PP16	PP15	PP14	PP13	PP12	PP11	PP10			Partial Pr	oduct 2
		1	3BE	PP28	PP27	PP26	PP25	PP24	PP23	PP22	PP21	PP20					Partial Pr	oduct 3
1	4BE	PP38	PP37	PP36	PP35	PP34	PP33	PP32	PP31	PP30							Partial Pr	oduct 4

Figure 3.4 Partial Product Initial Arrangement

	1	4BE	PP38	PP37	PP36	1BE	1E	1E	PP08	PP07	PP06	PP05	PP04	PP03	PP02	PP01	PP00	
			1	3BE		1	2BE	PP18	PP17	PP16	PP15	PP14	PP13	PP12	PP11	PP10		
					PP28	PP27	PP26	PP25	PP24	PP23	PP22	PP21	PP20					
						PP35	PP34	PP33	PP32	PP31	PP30							
	1	4BE	1S12	1S11	1S10	1S9	1S8	1S7	1S6	1S5	1S4	1S3	1S2	1S1	1S0	PP01	PP00	
		1C12	1C11	1C10	1C9	1C8	1C7	1C6	1C5	1C4	1C3	1C2	1C1	1C0				
						PP35	PP34	PP33	PP32	PP31	PP30							
	1	2S12	2S11	2S10	2S9	2S8	2S7	2S6	2S5	2S4	2S3	2S2	2S1	2S0	1S0	PP01	PP00	CLA
	2C12	2C11	2C10	2C9	2C8	2C7	2C6	2C5	2C4	2C3	2C2	2C1	2C0					
D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	2S0	1S0	PP01	PP00	

Figure 3.5 Wallace Tree Multiplication Method

First of all, the partial product initial arrangement is rearranged into first stage as shown in figure above. It can be seen like a tree shape here. The stage from **PP36** till **1** from the 4th partial product is moved to the first row and **3BE** together with **1** is moved up to the row partial product 2. After rearrangement, the first three rows will be added using ***half adder*** and ***carry save adders.*** The fourth partial product will not be added first but will be sent directly to the second stage. Hence, there total up to nine carry save adders and four half adders.

For second stage, the summation of the first half adders in right hand side of the first stage is examined. After the summation is done to add up **PP02** and **PP10**, The SUM (1S0) will be generated in the same column as the second stage shows where as the CARRY (1C0) will be shift left into next level of summation. In this stage, the bit **PP30-PP35** is finally being added using ***carry save adder***. At this stage, bit **4BE** is also being added by using ***half adders*****.** Hence, there are total six carry save adders and seven half adders needed in this stage.

In third stage, it is a final stage adder and since there are only remaining two inputs to be added instead of three, thus ***carry look ahead*** is used to perform the final summation based on the **Sum** and **Cout** signal in which had been propagated by the second stage. ***13-bit carry look ahead*** had been designed to be used in this ***Wallace tree final stage***. The bit PP00 and PP01 are directly sent to the output without going through any gate level. Hence, Wallace tree adder will have a 17 bit length output including the carry from the final bit.

3.2.6 BOOTH MULTIPLIER BY AN EXAMPLE

Let us see an example demonstrating the whole procedure of Booth multiplier (Radix -4) using Wallace Tree and Sign Extension Correctors.Let us take Example of calculation of (34×-42).

Multiplicand A = 34 = 00100010
Multiplier B = -42 = 11010110 (2's Complement form)

A×B = 34 × -42 = -1428

First of all, the multiplier had to be converted into radix number as in Figure below. The first partial product determined by three digits LSB of multiplier that are B1, B0 and one appended zero. This 3 digit number is 100 which mean the multiplicand A has to multiply by -2.To multiply by -2, the process takes two's complement of the multiplicand value and then shift left one bit of that product. Hence, the first partial product is 110111100. All of the partial products will have nine bits length.

Next, the second partial product is determined by bits B3, B2, B1 which indicated have to multiply by 2. Multiply by 2 means the multiplicand value has to shift left one bit. So, the second partial product is 001000100. The third partial product is determined by bits B5, B4, B3 in which indicated have to multiply by 1. So, the third partial product is the multiplicand value namely 000100010. The forth partial product is determined by bits B7, B6, B5 which indicated have to multiply by -1. Multiply by -1 means the multiplicand has to convert to two's complement value. So, the forth partial product is 111011110.

Figure below shows the arrangement for all four partial products to be added using Wallace tree adder method. 1E, 1BE 2E, 3E and 4E is obtained based on the Table 4.2. The way on how this sign E is arranged has been shown in Wallace Tree Multiplication Method above. The Wallace tree for the Example is given below.

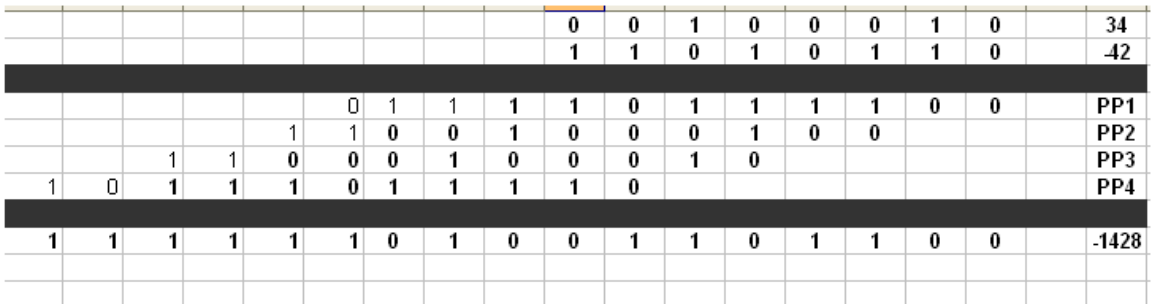


Figure 3.6 Method showing How Partial Products Should Be Added

To prove the output result is correct:

11111101001101100 =

$2^0(0) + 2^1(0) + 2^2(1) + 2^3(1) + 2^4(0) + 2^5(1) + 2^6(1) + 2^7(0)$

$+ 2^9(1) + 2^{10}(0) + 2^{11}(-1)$

= 4 + 8 + 32 + 64 + 512 – 2048

= -1428

3.4 ANALYSIS OF MULTIPLIERS

In our project we had to compare different multiplier on the basis of their **speed** and **power** parameters. We used Xilinx ISE version 10.2 for our simulation of different multipliers and knowing their delays. We analyzed Array Multipliers, Wallace Tree Multipliers and Booth Multiplier (Radix-2 and Radix-4) and analyzed their speed and power consumption using the above.

PARAMETERS	VALUES
Number Of Slices	**96**
Number Of 4-input LUTs	**178**
Number Of Bonded Input	**32**
Number Of Bonded Output	**32**
Power	**134mW**

Table 3.3 Array Multiplier

PARAMETERS	VALUES
Number Of Slices	**72**
Number Of 4-input LUTs	**130**
Number Of Bonded Input	**32**
Number Of Bonded Output	**32**
Power	**124mW**

Table 3.4 Booth Multiplier (Radix – 2)

PARAMETERS	VALUES
Number Of Slices	**69**
Number Of 4-input LUTs	**125**
Number Of Bonded Input	**32**
Number Of Bonded Output	**32**
Power	**87mW**
Delay	**25.435 ns**

Table 3.5 Wallace Tree Multiplier

PARAMETERS	VALUES
Number Of Slices	**96**
Number Of 4-input LUTs	**178**
Number Of Bonded Input	**32**
Number Of Bonded Output	**32**
Power	**79mW**
Delay	**26.645 ns**

Table 3.6 Booth Multiplier (Radix – 4)

If we compare the above values among each other we can observe that the **Array Multiplier** is the worst case multiplier consuming highest amount of power. Then comes the **Radix – 2 booth multiplier** which consumes lesser power than array multiplier. The **Wallace Tree multiplier** and **Booth Multiplier Radix-4** have nearly same amount of delay while **Radix-4 Booth** consuming lesser power than the other. Hence we reach to a conclusion that **Booth Radix-4 Multiplier** is best for situations requiring **Low power Applications.**

Chapter 4

LOW POWER OPTIMIZATIONS

4.1 Optimization of multiple recoding schemes for low power

4.2 High Level Comparison

4.3 Delay in various recoding schemes

4.4 New Recoding schemes

4.5 Comparison of multipliers with different recoding and PP generator schemes

4.1 Optimization of Multiplier Recoding schemes for Low Power

The multiplier operand Y is often recoded into a radix higher than 2 in order to reduce the number of partial products. The most common recoding is radix – 4 recoding with digit set **{-2, -1, 0, 1, 2}.** For a series of consecutive 1's, the recoding algorithm converts them 0's surrounded by a 1 and a (-1), which has potential of reducing switching activity. At the binary level, there are many design possibilities. The traditional design objectives are small delay and small area. The power issues of different designs have not been addressed well. In this chapter, we focus on the effects of radix-4 recoding schemes in multipliers and optimize their designs for low power. Here, we give an overview and analysis of several known recoding schemes and their designs and compare the Timing and power consumption by them.

Intuitively, radix-4 multipliers could consume less power than their radix-2 counterparts as recoding reduce the number of PPs to half. However, the extra recoding logic and the more complex PP generation logic may present significant overheads. In addition, recoding introduces extra unbalanced signal propagation paths because of the additional delay on the paths from operand *Y* to the product output. We have showed that Wallace tree multipliers consumed less power than Booth-recoded radix-4 multipliers although the radix-2 scheme had twice as many PPs as the radix-4 scheme. This leads us to believe that the design of recoders and PP generators plays an important role in the overall power consumption in multipliers.

4.1.1 Parallel Recoding Schemes

In parallel recoding, at least three signals are needed to represent the digit set **{**-2,-1, 0, 1, 2**}**. To achieve area/delay tradeoff, additional signals are often used. We classify existing schemes of radix-4 recoding design by the number of control signals. There are three classes: three-signal schemes, four-signal schemes, and five-signal schemes. They are different in one or more

aspects of parallel/serial recoding, control signals, zero handling, and logic organization.

THREE - SIGNAL SCHEMES

4.1.1.1

In three-signal schemes, one standard approach [] is the **THREE_SIGNAL_1** scheme in Table 4.1. The signal *cor* is the correction bit for negative numbers. The following switching expressions are deduced:

$$neg_i = y_{2i+1}$$

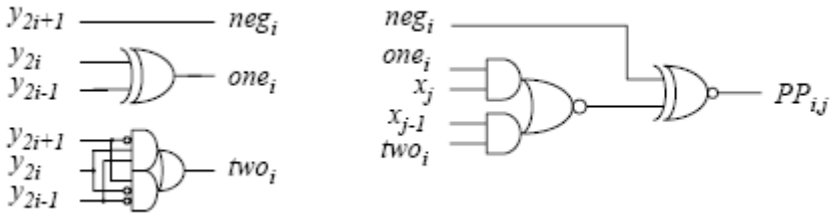
$$two_i = y'_{2i+1}y_{2i}y_{2i-1} + y_{2i+1}y'_{2i}y'_{2i-1}$$

$$one_i = y_{2i} \ XOR \ y_{2i-1}$$

$$cor_i = y_{2i+1}$$

Y_{2i+1}	Y_{2i}	Y_{2i-1}	Partial Product	neg_i	two_i	one_i	cor_i
0	0	0	0	0	0	0	0
0	0	1	1×Multiplicand	0	0	1	0
0	1	0	1×Multiplicand	0	0	1	0
0	1	1	2×Multiplicand	0	1	0	0
1	0	0	-2×Multiplicand	1	1	0	1
1	0	1	-1×Multiplicand	1	0	1	1
1	1	0	-1×Multiplicand	1	0	1	1
1	1	1	0	1	0	0	1

Table 4.1: Recoder for the scheme THREE_SIGNAL_1



RECODER **PP GENERATOR**

4.1.1.2

In three-signal schemes, one standard approach is the **THREE_SIGNAL_2** scheme in Table 4.2. The signal *cor* is the correction bit for negative numbers. The following switching expressions are deduced:

$$neg_i = y_{2i+1}(y_{2i}y_{2i-1})'$$

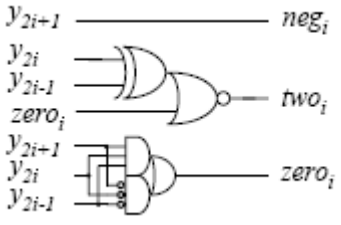
$$two_i = y'_{2i+1}y_{2i}y_{2i-1} + y_{2i+1}y'_{2i}y'_{2i-1}$$

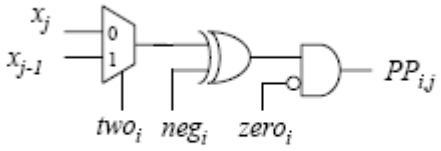
$$zero_i = y_{2i+1}y_{2i}y_{2i-1} + y'_{2i+1}y'_{2i}y'_{2i-1}$$

$$cor_i = neg_i$$

Y_{2i+1}	Y_{2i}	Y_{2i-1}	Partial Product	neg_i	two_i	$zero_i$	cor_i
0	0	0	0	0	0	1	0
0	0	1	1×Multiplicand	0	0	0	0
0	1	0	1×Multiplicand	0	0	0	0
0	1	1	2×Multiplicand	0	1	0	0
1	0	0	-2×Multiplicand	1	1	0	1
1	0	1	-1×Multiplicand	1	0	0	1
1	1	0	-1×Multiplicand	1	0	0	1
1	1	1	0	0	0	1	0

Table 4.2: Recoder for the scheme THREE_SIGNAL_2





RECODER **PP GENERATOR**

4.1.1.3

In three-signal schemes, one standard approach is the **THREE_SIGNAL_3** scheme in Table 4.3. The signal *cor* is the correction bit for negative numbers. And the other three signals are *pos, neg,two* The following switching expressions are deduced:

$$neg_i = y_{2i+1}(y_{2i}y_{2i-1})'$$

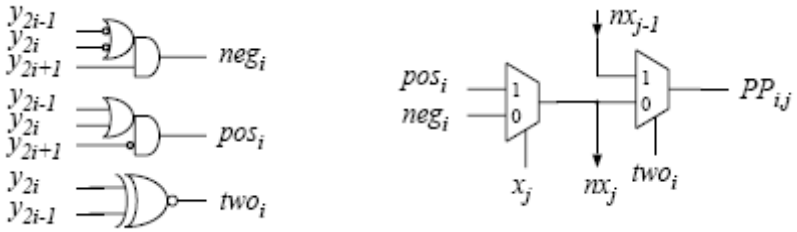
$$pos_i = y'_{2i+1}(y_{2i}+y_{2i-1})$$

$$two_i = (y_{2i} \ XOR \ y_{2i-1})'$$

$$cor_i = neg_i$$

Y_{2i+1}	Y_{2i}	Y_{2i-1}	Partial Product	neg_i	pos_i	two_i	cor_i
0	0	0	0	0	0	1	0
0	0	1	1×Multiplicand	0	1	0	0
0	1	0	1×Multiplicand	0	1	0	0
0	1	1	2×Multiplicand	0	1	1	0
1	0	0	-2×Multiplicand	1	0	1	1
1	0	1	-1×Multiplicand	1	0	0	1
1	1	0	-1×Multiplicand	1	0	0	1
1	1	1	0	0	0	1	0

Table 4.3: Recoder for the scheme THREE_SIGNAL_3



RECODER **PP GENERATOR**

4.1.1.4 FOUR - SIGNAL SCHEMES

In Four-signal schemes we have an approach is the **FOUR_SIGNAL_1** scheme in Table 4.4. The signals here are as follows: *neg, tmp1, tmp2, one, two, zero, cor*, The following switching expressions are deduced

$$neg_i = y_{2i+1}$$

$$tmp1_i = y_{2i+1} \; XOR \; y_{2i-1}$$

$$tmp2_i = y_{2i+1} \; XOR \; y_{2i}$$

$$one_i = y_{2i} \; XOR \; y_{2i-1}$$

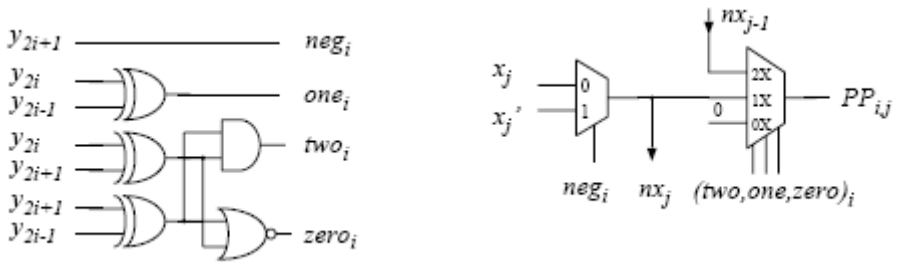
$$two_i = tmp1_i * tmp2_i$$

$$zero_i = (tmp1_i + tmp2_i)'$$

$$cor_i = y_{2i+1} \; zero_i'$$

Y_{2i+1}	Y_{2i}	Y_{2i-1}	Partial Product	neg_i	two_i	one_i	$zero_i$	cor_i
0	0	0	0	0	0	0	1	0
0	0	1	1×Multiplicand	0	0	1	0	0
0	1	0	1×Multiplicand	0	0	1	0	0
0	1	1	2×Multiplicand	0	1	0	0	0
1	0	0	-2×Multiplicand	1	1	0	0	1
1	0	1	-1×Multiplicand	1	0	1	0	1
1	1	0	-1×Multiplicand	1	0	1	0	1
1	1	1	0	1	0	0	1	0

Table 4.4: Recoder for the scheme FOUR_SIGNAL_1



RECODER **PP GENERATOR**

4.1.2 Serial Recoding Schemes

For serial radix-4 recoding, there is not much previous work because its application is limited to linear array multipliers although it has the potential of power saving. However, the use of four possible values instead of five does not necessarily simplify the design because the digit set is not symmetric. The equation derived is shown below.

$neg_i = y_{2i+1}\ (y_{2i}+C_i)$

$one_i = y_{2i}$ XOR C_i

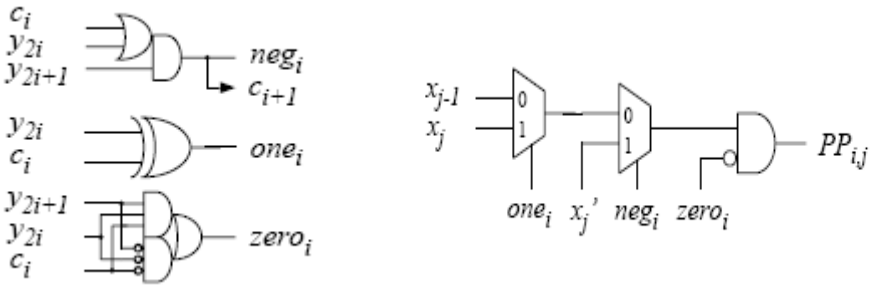
$zero_i = y_{2i+1}\ y_{2i}\ C_i + y_{2i-1}'\ y_{2i}'\ C_i\ '$

$cor_i = y_{2i+1}\ * one_i$

$C_{i+1} = neg_i$

Y_{2i+1}	Y_{2i}	C_i	Partial Product	neg_i	one_i	$zero_i$	cor_i	c_{i+1}
0	0	0	0	0	0	1	0	0
0	0	1	1×Multiplicand	0	1	0	0	0
0	1	0	1×Multiplicand	0	1	0	0	0
0	1	1	2×Multiplicand	0	0	0	0	0
1	0	0	-2×Multiplicand	0	0	0	0	0
1	0	1	-1×Multiplicand	1	1	0	1	1
1	1	0	-1×Multiplicand	1	1	0	1	1
1	1	1	0	1	0	1	0	1

Table 4.5: Recoder for the scheme SERIES_SIGNAL_1



RECODER **PP GENERATOR**

4.2 High Level Comparison

When we go for high level comparisons we have two parameters for us when we consider different types of recoding schemes. It is the recoder and the partial product generator. The delay of REC and PPG is estimated roughly as equivalent XOR2 gate delay (*TXOR*2) without considering fan-out and signal transition directions. Simple gates such as AND2 have the delay of 0.5*TXOR*2. Two-level AND/OR gates and MUX21 have the same delay as XOR2.

Among above discussed recoding schemes, **THREE_SIGNAL_1** has both the simplest recoder and the simplest PP generator. But doesn't have a unique zero handling.

After analyzing it was found out that in an m × n bit radix-4 multiplier, the total area of recoders is $\mathbf{n/2}A_{REC1b}$ while the total area of PP generators is $\mathbf{n/2 * (m+1)}A_{PPG1b}$, where A_{REC1b} is area of 1-bit recoder cell area and A_{PPG1b} is area of 1-bit partial product generator cell area. Hence since partial product generators (PPG) occupy larger areas than the recoders, we need to simplify PP generators for power saving. In addition, unique zero handling is desired in order to reduce the number of unnecessary switching activities.

4.3 Delay in various recoding schemes

The delay found in the above mentioned recoding schemes are given in the table presented below.

Schemes	*Delay(in ns)*
Three_signal_1	**3.1**
Three_signal_2	**4.2**
Three_signal_3	**2.75**
Four_signal_1	**2**
Series_signal_1	**3.5**

Table 4.6: Delay for different schemes

4.4 New Recoding schemes

For new parallel recoding schemes we know we always have **THREE_SIGNAL_1** having simplest recoder and PP generator. So we can think to keep the simplicity of the PP generator because it occupies most area and can think of increasing the complexity of recoder to add the power efficient zero handling. So the new scheme can be shown to be as follows.

$$neg_i = y_{2i+1}(y_{2i}y_{2i-1})'$$

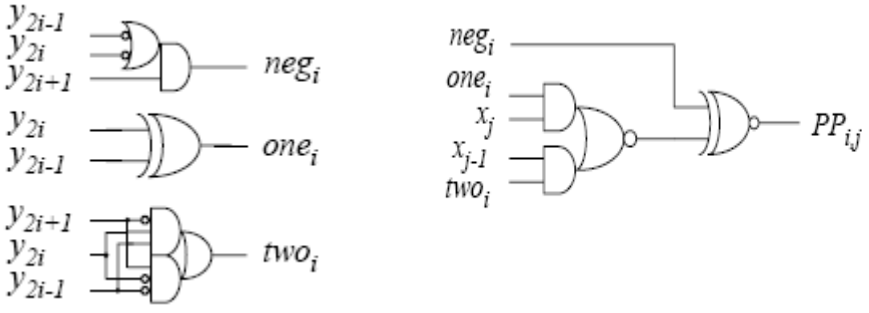
$$two_i = y'_{2i+1}y_{2i}y_{2i-1} + y_{2i+1}y'_{2i}y'_{2i-1}$$

$$one_i = y_{2i} \ XOR \ y_{2i-1}$$

$$cor_i = neg_i$$

Y_{2i+1}	Y_{2i}	Y_{2i-1}	Partial Product	neg_i	two_i	one_i	cor_i
0	0	0	0	0	0	0	0
0	0	1	1×Multiplicand	0	0	1	0
0	1	0	1×Multiplicand	0	0	1	0
0	1	1	2×Multiplicand	0	1	0	0
1	0	0	-2×Multiplicand	1	1	0	1
1	0	1	-1×Multiplicand	1	0	1	1
1	1	0	-1×Multiplicand	1	0	1	1
1	1	1	0	0	0	0	0

Table 4.7: Recoder for the scheme NEW_THREE_SIGNAL_1



RECODER **PP GENERATOR**

4.5 Comparison Of Multipliers with Different Recoding Schemes

SCHEME	POWER (mW)	DELAY(ns)
Three_signal_1	**29.72**	**11.00**
Three_signal_2	**30.80**	**11.15**
Three_signal_3	**27.88**	**10.73**
Four_signal_1	**29.82**	**10.93**
Series_signal_1	**31.43**	**10.97**
New_Three_signal_1	**27.32**	**10.31**

Table 4.8: Power and delays of multiplier with above discussed schemes of recoders and PP generators.

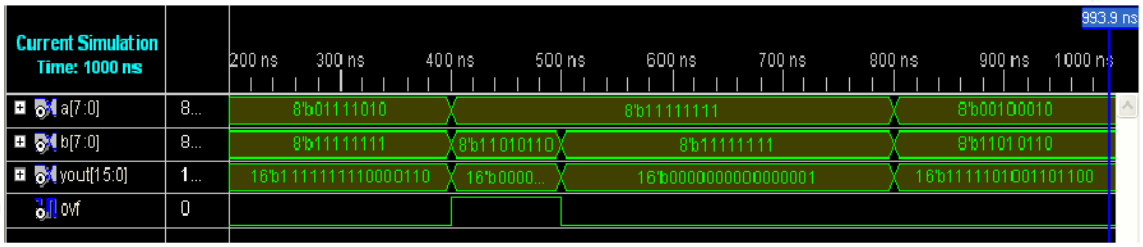
In the above as we can see we have calculated the Power and Delays by different recoding schemes. We have also included the new proposed recoding scheme which was modified version of some previous recoding scheme.

From above only we can say that parallel recoding schemes always consume less power than serial recoding schemes and hence are used very less these days. But if we look at the modified parallel recoding scheme it has much less power consumed than the previous schemes and also has less delay in the circuit as compared to the other schemes. Hence **New_Three_signal_1** can be used as a recoder scheme when it comes to lesser power and lesser delay applications, which are generally the basic motto requirements of today's integration industry.

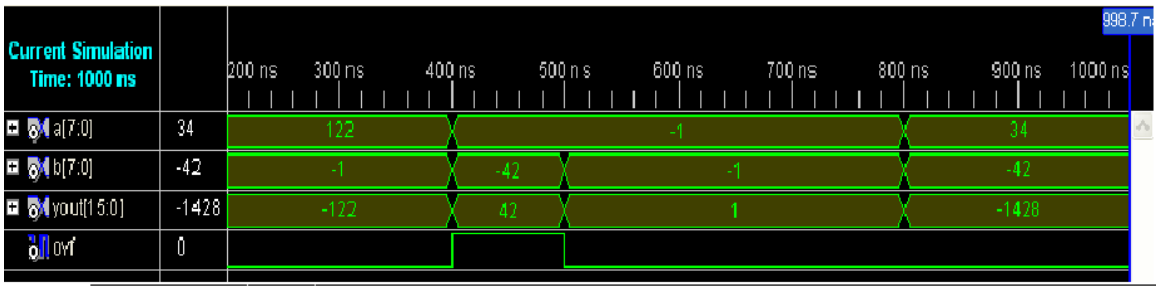
Chapter 5

OUTPUT WAVEFORMS

BOOTH MULTIPLIER OUTPUT

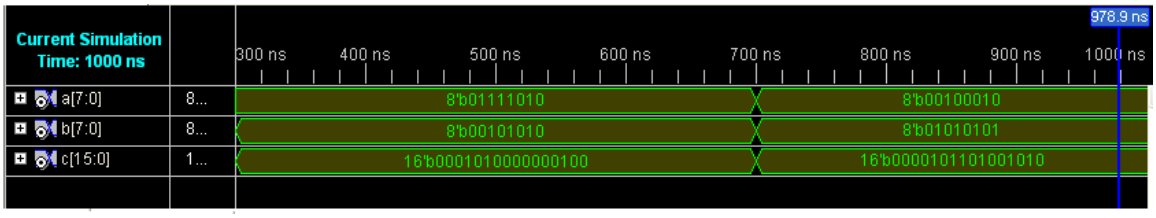


Binary Form output

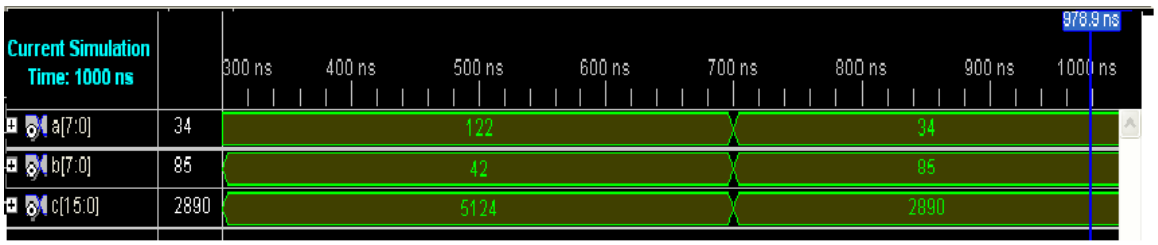


Signed Form Output

WALLACE TREE MULTIPLIER OUTPUT



Binary Form Output



Signed Form Output

Chapter 6

6.1 Conclusion

After going through all the hard work and facing problems, this project managed to complete its objectives that are to study different Multiplier and learn the Power and Time trade off among them so that we can design **Efficient Faster Low Power Multiplier**.

We studied about different adders among compared them by different criteria like Area, Time and then Area-Delay Product etc. so that we can judge to know which adder was best suited for situation. After comparing all we came to a conclusion that Carry Select Adders are best suited for situations where Speed is the only criteria. Similarly Ripple Carry Adders are best suited for Low Power Applications. But Among all the Carry Look Ahead Adder had the least Area-Delay product that tells us that, it is suitable for situations where both low power and fastness are a criteria such that we need a proper balance between both as is the case with our Project.

Coming to Multipliers we studied different Multipliers starting from Array Multiplier to Wallace Tree, Booth Multipliers, both Radix-2 and Radix-4.We found that parallel multipliers are much better than the serial multipliers due to less area consumption and hence the less power consumption. Comparing Radix-2 and Radix-4 booth multipliers we found that radix-4 consumes less power than radix-2, because radix-4 uses almost half number of iterations than radix-2.We saw Wallace tree having nearly same delay as of radix-4 multipliers where as consuming a little more power than the former.

After all this then we tried to improve power efficiency of circuits. Hence we went for studying different recoding schemes along with their Partial Product generators and study time and power required by them in a multiplication process. After studying them we went to modify one of the recoding schemes to find a proper combination of recoder and PP generator such that we will have simplest PP generator as these take maximum area in a cell area and then take care of zero handling as it handles most of the switching activities. Hence we ended up creating a better recoding scheme.

6.2 FUTURE WORK

As an attempt to develop arithmetic algorithm and architecture level optimization techniques for low-power multiplier design, the research presented in this dissertation has achieved good results and demonstrated the efficiency of high level optimization techniques. However, there are limitations in our work and several future research directions are possible.

One possible direction is radix higher-than-4 recoding. We have only considered radix-4 recoding as it is a simple and popular choice. Higher-radix recoding further reduces the number of PPs and thus has the potential of power saving.

Another possible direction can be representation of Arguments such as in sign-magnitude or 2’s compliment form which in any case would prove better according to situation and require less power and consume less time.

6.3 REFERENCES

[1] *K.H. Tsoi, P.H.W. Leong, "Mullet - a parallel multiplier generator," fpl, pp.691-694, International Conference on Field Programmable Logic and Applications, 2005., 2005*

[2] *S. Tahmasbi Oskuii, P. G. Kjeldsberg, and O. Gustafsson, "Transition activity aware design of reduction-stages for parallel multipliers," in Proc. 17th Great Lakes Symp. On VLSI, March 2007, pp. 120–125.*

[3] *Ayman A. Fayed, Magdy A. Bayoumi, "A Novel Architecture for Low-Power Design of Parallel Multipliers," wvlsi, pp.0149, IEEE Computer Society Workshop on VLSI 2001, 2001*

[4] *M. 0. Lakshmanan, Alauddin Mohd Ali, "High Performance Parallel Multiplier Using Wallace-Booth Algorithm," IEEE International Conference on Semiconductor Electronics, pp. 433-436, 2002.*

[5] *Design Compiler User Guide. Synopsys, Inc., Nov. 2000.*

[6] *Z. Huang, "High-Level Optimization Techniques for Low-Power Multiplier Design," PhD dissertation, Univ. of California, Los Angeles, June 2003.*