FPGA Implementation of Booth Encoded Multi-Modulus $\{2^n -1, 2^n, 2^n +1\}$ RNS Multiplier

A Thesis Report

submitted in partial fulfillment of the requirements

for the award of degree

of

Master of Engineering

in

Electronics and Communication Engineering

Submitted By

Saguna Goel

(801461025)

Under the supervision of:

Ms. Sakshi Bajaj & Ms. Amanpreet Kaur

Asst. Professors, ECED

ELECTRONICS AND COMMUNICATION ENGINEERING

DEPARTMENT

THAPAR UNIVERSITY

(Established under the section 3 of UGC Act, 1956)

PATIALA – 147004 (PUNJAB)

June 2016

DECLARATION

I hereby declare that the work which is being presented in the thesis entitled, **"FPGA Implementation of Booth encoded multi-modulus {2^n-1, 2^n, 2^n+1} RNS Multiplier"**, in partial fulfilment of the requirements for the award of degree of Master of Engineering in Electronics and Communication Engineering submitted in Electronics & Communication Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of **Ms. Sakshi Bajaj and Ms. Amanpreet Kaur** and refers other researcher's works which are duly listed in the reference section.

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

Date:- 7/7/2016

(Saguna Goel)

Roll no: 801461025

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Date:- 7/7/2016

Ms. Sakshi Bajaj &

Ms.Amanpreet Kaur

Asst. Professors, ECED

Thapar University, Patiala

Countersigned by

Dr. Sanjay Sharma

Professor and Head (ECED)

Thapar University, Patiala

Dr. S. S. Bhatia

Dean of Academic Affairs

Thapar University, Patiala

ACKNOWLEDGEMENT

I would like to express my most sincere appreciation and deep sense of gratitude and indebtedness to my guides **Ms. Sakshi Bajaj and Ms. Amanpreet Kaur,** assistant professors, Electronics & Communication Engineering Department, Thapar University, Patiala for their continuous indefatigable guidance, which paved me on to the path to carry this project. I am highly indebted to them for their painstaking efforts and invaluable suggestions during the period of work.

I am also thankful to **Dr. Sanjay Sharma**, Professor and Head and **Dr. Amit Kumar Kohli**, PG Coordinator, Electronics and Communication Engineering Department, Thapar University, Patiala for their valuable advice and helped in all possible ways for completion of my work.

I am also very thankful to the entire faculty and staff members for the direct-indirect help, cooperation and support.

Date: - 7/7/2016

Place: Patiala

(Saguna Goel)

ABSTRACT

A novel number system, Residue Number System (RNS) deals with a small set of integers making the arithmetic easier to realize. RNS is a non weighted number system without any significant ordering of digits which makes it suitable for fault tolerant applications. The research work aims at designing a fast efficient RNS based Booth encoded multiplier since multiplication, a very important arithmetic operation, is finding a great use these days in various applications (especially DSP, multimedia and image processing). Due to advances in technology and increasing needs for high speed calculations, it has become really necessary and important to design a fast efficient multiplier. With advances in technology, many researchers have tried and are trying to design multipliers which offer either to the following-high speed, low power consumption, regularity of layout and hence less area or even combination of them in multiplier. However area and speed are two conflicting constraints. So improving speed results always in larger areas.

A productive method to speed up the multiplication is reduction in the Partial Product (PP) array. Various techniques exist for the reduction of Partial Products. But according to research, Booth's algorithm is the most efficient and most widely used. This algorithm effectively multiplies the operands for partial product generation. In this research , an RNS based Booth encoded multi-modulus $\{2^n-1, 2^n, 2^n+1\}$multiplier has been designed using Verilog HDL, synthesized and simulated on Xilinx 14.5 and targeted on Spartan 3E FPGA device.

TABLE OF CONTENTS

LIST OF ACRONYMS

PP	Partial Product
LSB	Least Significant Bit
DSP	Digital Signal Processing
VLSI	Very Large Scale Integration
SPM	Serial Parallel Multiplier
PPG	Partial Product Generation
PPR	Partial Product Reduction
PPA	Partial Product Accumulation
CSA	Carry Save Adder
HDL	Hardware Descriptive Language
RNS	Residue Number System
CRT	Chinese Remainder Theorem
MRC	Mixed Radix Conversion
CPU	Central Processing Unit
FIR	Finite Impulse Response
IIR	Infinite Impulse Response
DR	Dynamic Range
ADC	Analog to Digital Converter
DAC	Digital to Analog Converter
NB	Non Binary
RB	Redundant Binary
FPGA	Field Programmable Gate Array
RSA	Right Shift Accumulate
RRNS	Redundant Residue Number System
DFT	Discrete Fourier Transform
GCD	Greatest Fourier Transform

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 Motivation

Multiplication, a process of repetitive addition, is gaining importance in various applications, especially in the field of Digital Signal Processing (DSP) for running complicated high speed computations [1]. Due to advances in technology and increasing needs for high speed calculations, it has become really necessary and important to design a fast efficient multiplier. A productive method to speed up the multiplication is reduction in the Partial Product (PP) array. Multiplication process involves three basic steps: 1) Generation of Partial Products (PPG) 2) Reduction in Partial Products (PPR) and 3) Addition of Partial products. Multipliers are a prime activity in most processors and generally have large area consumption and high latency. They are slowest components of a system, hence, the realization of a system is mainly regulated by realization of the multiplier. Therefore, upgrading area and speed is a major issue while designing a multiplier.

A number of techniques have been proposed to drive the speed of multiplication [3-8].Also various algorithms to reduce number of Partial Products have been proposed. Booth encoding is one such technique and is most frequently used. Booth encoding increases the multiplication speed and thus makes the arithmetic fast.Radix-4 Booth encoded technique is the most efficient among all techniques as it reduces the number of partial products by an aspect of two [2].Radix-8 reduces the number of Partial Products by an aspect of three as compared to that in Radix-2, but the generation of partial products in Radix-8 is a complex process. This is so because with the increase in radix, the number of multiples and also the hard multiples increase [9].Thus, Radix-8 and more higher order radices are not cost effective and are rarely used [10]. The first and foremost step in multiplication is the production of partial products. It is very important to reduce the partial product array for faster multiplication. Booth‘s algorithm is the mostly used technique to perform the same.

RNS is used to express a large number in terms of a set of smaller integers. Basically it is used for fault tolerant computations and faster arithmetic. Three main attributes of RNS which make it suitable for these implementations are: First, it is a non-weighted

number system, hence any fault in any digit does not affect the other digit. Second, there is no significant ordering of digits in RNS. Therefore, any faulty digit can be easily removed at the cost of reduction in Dynamic Range. Third, carry propagation which is a limiting factor in addition and multiplication operations, is absent in RNS. The conversion between conventional number system and RNS is also possible. For processing the data in RNS, data needs to be converted to RNS first. The process of converting conventional numbers to RNS is known as Forward conversion. Similarly, converting back data from RNS to conventional numbers is known as Backward conversion. Reverse conversion algorithms are based either on Chinese Remainder Theorem (CRT) or Mixed-Radix Conversion (MRC).In every reverse convertor, a large modulo adder is required at last stage, which makes the system complex.

Pipelining has been used for increasing the design efficiency further. Pipeline, as the name indicates, is a combination of various processing units connected in series. The input to every processing unit is the output from the previous processing unit, just like a pipeline. Pipelining, thus, is an excellent leading technique wherein processing of next instruction begins before the previous one has actually ended. Therefore parallel execution of units occurs making the processing faster. Memory units such as flips flops are usually employed in pipelining process. Pipelining offers various advantages like-First, reduction in processor cycle time, making the proceeding fast. Second, Arithmetic Logic Unit of CPU can be designed faster. Third, overall performance of the design is improved. Apart from advantages, pipelining offers few disadvantages too- First, execution of simultaneous instructions at a time might increase branch delays. Second, a pipelined block generally do not have a stable instruction bandwidth.

1.2 Objective

The present work focuses on designing a fast multiplier. Booth's algorithm has been used for the reduction of Partial Products (PPs), the first step in multiplication. The goal is to analyze the behaviour of Booth encoded multiplier in Radix-2, Radix-4 and Radix-8 and to find the best among these three on the basis of area utilization and speed. The main aim is to implement Residue number system (RNS) in Booth multiplier for the best chosen radix and further observe the behaviour of multiplier. The application of pipelining in these designs of the multiplier will also be discussed.

1.3 Thesis Orientation

The work has been divided into six chapters. The first chapter focuses on the motivation and main objective of the work. The second chapter discusses the literature studied. Third chapter is detailed about RNS multiplication. It involves RNS architecture, advantages and disadvantages of RNS, various arithmetic operations involved in RNS. Fourth chapter is a discussion about various types of multipliers. Main focus is on Booth Multiplication Algorithm for Radix-2, Radix-4 and Radix-8. In fifth chapter, simulation and synthesis results are presented. Sixth chapter is a conclusion of the work done and also the future scope of the work done is outlined.

CHAPTER 2
LITERATURE REVIEW

Many papers were studied on the recent researches and developments in the direction of making the multipliers more efficient. A brief summary is as follows:

H. L. Garner [1] Investigation and development of properties of a new system, known as Residue Number System (RNS)/Residue Code was done. It was shown that RNS is of great interest due to its faster arithmetic operations. Multiplication and addition could both be done in the same time as taken by addition alone. The major difficulty in RNS arithmetic was found in division, scaling and also in determining relative magnitude of numbers. Residue code finds a great use in applications which require high speed calculations and implementations. A brief discussion to the properties of congruences was shown. Also, it was shown that residue code can easily be evolved in terms of congruences.

K.Y Khoo et al. [2] proposed that a little simple modification done in Booth multiplication algorithm could be used for increasing the chances of zero coded digit. The chances of zero in bits of Partial Products (PPs) of a Booth multiplier were increased and therefore, average number of transitions in bits of Partial Products was reduced by a percentage of 3.75 compared to conventional algorithm of Booth multiplier, for any input taken randomly. Moreover, it was also shown that chances of transition of carry bits in adder circuits of PPs is related to chances of transition of PPs directly and was decreased by a percentage of 3.75 to 7 approximately. Simulations on HSPICE showed that encoding proposed was able to lower the power dissipation by more than 4% for a Wallace Tree multiplier core and a two's complement 16 ×16 linear array.

R. Conway et al. [3] introduced a novel architecture for the implementation of a transposed RNS filter. A group of most common moduli set of the form $\{2^n-1, 2^n, 2^n+1\}$ was used. The number of moduli in the set was not confined to 3,but, it can be any number consisting only of relatively prime moduli of this form that fulfill the requirement of Dynamic Range (DR). With use of an effective algorithm for

generation of carry save trees, various readings of area utilization and time delays for different values of n were retrieved. Using these values a moduli set of RNS was chosen, which could reduce the area delay product in a single stage of transpose FIR. It was found that in comparison to use of the common 3 modulus set such modulus sets offered a significant improvement in speed. With the same dynamic range, the cost comparison was done between RNS filter stage and an equivalent 2's complement filter.RNS stage proposed, using restricted moduli, was observed and shown to provide a noteworthy overall area- delay product gain for a number of different values of dynamic ranges. The gain in percentage ranged from 35% to 60% for a 16-tap filter for a dynamic range between 20–40 b.

R. C. Ismail et al. [4] proposed the methods needed to implement a fast and efficient performance complex number parallel multiplier. The designs were framed using Booth encoded Radix-4 algorithm and Modified Wallace tree. These techniques thus have the ability to constrict the terms of partial products in the ratio 3:2 and reduce generation of partial products to 11:2.Due to these advantages these techniques are selected. Also, these are used for speeding up the multiplication operation. In addition to these carry save-adders (CSA) have been used to boost the speed of addition operation in the system. The designing of system was done using VHDL code. Simulation and synthesis was done using ModelSim XE II 5.8c and Xilinx ISE 6.1.The system was implemented later on Xilinx Virtex-It Pro FPGA board for the proof.

D. Adamidis et al. [6] proposed a new design for units capable of performing modulo 2^n-1(diminished-1) and 2^n+1 multiplication (or sum-of squares) depending on the value of control. The Residue Number System generally finds its use in multimedia applications and Digital signal processing (DSP) applications. In such systems, multiplication and sum of squares are the commonly used operations and the moduli set commonly used is moduli of the forms of 2^n-1(diminished-1) and 2^n+1. These operations were performed using consecutive machine cycles or/and explicit design units. The area utilization of these operations, for having distinct units, is very large. Moreover, a number of machine cycles are required for solutions based on employing a multiplier-adder pair for the performance of sum of squares operation. Two architectures of units capable of performing either the jX - YjR or the jX2 + Y2jR

operation (depending on the value of a control signal) were proposed. It was observed that R can either have the 2^n-1 or the 2^n + 1 form .Diminished-1 representation was used in the latter case.

Y. He et al. [7] proposed an energy-efficient RB (Redundant Binary) and high-speed multiplier design based on a new covalent RB Booth encoding algorithm. The main idea behind it was to polarize two neighbouring Booth-encoded digits into a contrary pair to recover the effective reduction rate of RB partial product without the overhead in conversion from NB-to-RB. The method so proposed fully utilize the characteristics of the negative-positive coding in compliment form of RB number for generating an RB partial product (PP) directly from the two consecutive Booth-encoded digits. Thus, it shared the same benefits as that of RB Booth encoder for the avoidance of error compensation vector and easy generation of hard multiples are the two problems which were encountered by RB multiplier with a simple normal binary Booth encoding. Six multipliers (RB) with different techniques of Booth encoding techniques were designed for evaluation process. The synthesis results showed that the RB multiplier so designed based on CRBBE-4 is the most energy-efficient and fastest one among its contenders for the existing power-of-two computing word lengths.

Z. Li et al. [8] verified the design of an 8×8 bit multiplier (signed/unsigned) with proper perfect results. Work was done to dispose off negative Partial Product (PP) on the basis of Radix-4 Booth algorithm. Without any increase in any new PP, it advanced a skilful method neglecting additive arithmetic of “plus 1”. Disposing off the negative PP, lead to an increase in chip resources and shortening of key path, which improved the performance of multiplier.

R. Muralidharan et al. [9] proposed a Radix-4 booth encoded, diminished-1 multiplier with efficient area utilization. This approach was able to minimize number of encoder and decoder blocks in booth multiplier, needed for generating Partial Products(PPs).The correction factor was split into two sections-first multiplier independent static bias and second, multiplier dependent dynamic bias. Dynamic bias was generated using hardwiring of outputs of Booth encoding block. Sum of static and dynamic bias was reduces to an easy simple binary form (with alternate zeroes

and ones).The multiplier proposed was successful in achieving power reduction of 18.5% and an area saving of 27% for n=40 compared to 2^n+1 multiplier (non-encoded).The average power reduction and area saving of 62% and 52% respectively for same value of n was observed. This behaviour of proposed multiplier was compared to an existing modulo 2^n+1 Booth multiplier on the basis of gate model. It was found that the multiplier proposed was both cost and power efficient with a little compromise in speed.

M. Sharma et al. [10] addressed the problem of removal of partial products by calculating two's complement, neglecting an extra block for adding 1 and generation of lengthy carry chains. The mechanism for reduction of partial products was implemented and the number of partial products reduced from n/2+1 to n/2 compared to a modified booth algorithm. Also, using Verilog HDL, possibility of using same hardware to behave as a multiplier and as a single two's complement computer as created. In the proposed multiplier, there is no restriction to the number of bits to be multiplied. The proposed multiplier was synthesized o Xilinx with maximal frequency of operation as 14.5 MHz and delay of 7.013ns.

S. Shrivastava et al. [11] presented the design and implementation of Radix-2 Booth Multiplier and its comparison to Radix-4 Booth encoded multiplier. The comparison is done with the use of RTL schematic. Various techniques required for area improvement and effective enhancement of speed of circuit were discussed. The RTL schematics were implemented on FPGA CPLD kit for proper output. Radix-4 Booth multiplier showed higher speed and lower area compared to Radix-2 Booth multiplier.

N. Ghazali et al. [12] designed a multiplier that generate product between multiplier and multiplicand in shorter time execution by using hybrid modified Booth encoded algorithm and carry save adder techniques .The maximum time saving that could be achieved was upto 18.29%.Also management to reduce number of logic element in field programmable gate array (FPGA) was done. In comparison to basic multiplier architecture number of logic elements was reduced to about 12.27%.

D. Chandel et al. [13] designed a multiplier which reduces maximum apex of an array of Partial Products (PPs), which further simplified the PP reduction tree in

denominations of regularity and delay of the design. This was of great interest for multipliers needing small bit length for effective performance. It was also made clear that radix-8 multiplier has more advantages compared to radix-2 and radix-4 in terms of the requirement of transistors (radix-8 requires less transistors comparatively).Due to this less power is consumed.

B. Sreekanth et al. [14] a different modern approach for modulo 2^n-1 was proposed. Identical to binary multiplication, the Partial Products were generated using AND gates. For reduction in compression speed of column size (from N to two), Wallace tree was used. An algorithm (for optimization of delay) of the Wallace tree, for effective utilization of unequal delay of full adder was evolved. The scheme proposed, provided a significant performance in speed and requirement of hardware. Because of easy hardware implementation of arithmetic operations in special $\{2^n-1, 2^n, 2^n+1\}$ moduli set, these were preferred compared to generic moduli.Radix-8 multiplier, for which the delay is adaptable to delay in RNS multiplication, was proposed. The delay of 2^n-1 multiplier was made scalable, by controlling word length of ripple carry adder (employed for generation of hard multiples in Radix-8).

R. Muralidharan et al. [15] Radix-2 and Radix-4 multi modulus multipliers (Booth encoded) which are able to perform efficient multiplication for special set of moduli i.e. $\{2^n-1, 2^n, 2^n+1\}$, with a match among $\{2^n-1, 2^n, 2^n+1\}$ modulo operations were proposed. The architectures of Radix-2 and Radix-4 modulo multipliers comparable to $\{2^n-1, 2^n, 2^n+1\}$ modulus multipliers was proposed. Also, the equality in operations(central to modulus multiplication)i.e. reduction in binary weights, negation of modulus, addition of two operands and multiplication in power of two for moduli set $\{2^n-1, 2^n, 2^n+1\}$was introduced.

C. P. R. Mejia et al.[16] presented the design of Radix-8 booth encoded Montgomery Multiplier for 8192-bit.Also, they designed a Montgomery multiplier based on coded-digit. It was shown that both these multipliers perform two multiplications concurrently. Both these multipliers use systolic architecture. 8142-bit multipliers have highest throughput and highest area consumption. So there is an area-throughput trade-off. These multipliers are useful in RSA Cryptosystems.

K.M. Karthik et al. [17] designed a reverse converter for RNS i.e. from RNS back to binary. Reverse conversion requires a large circuitry at an end stage which makes this conversion a difficult one. Also it has been discussed that RNS finds a great use in various DSP applications due to its fast processing (as it deals with small integers). Reverse conversion for the moduli set $\{2^n-1, 2^n, 2^n+1\}$ has been discussed. The use of parallel prefix ling adder was made for implementation of the modular adder, required at the last stage of reverse converter.

H. Jiang et al. [18] proposed different 16× 16 signed multipliers for Radix-8. Initially, for the calculation of triple of numbers (binary), a proposal of 2-bit adder consisting of XOR gate with three inputs was proposed. Also, they presented various circuits like that of error detection, compensation circuit and recovery circuit for 2-bit adder. A truncation technique for was employed for saving power and increasing speed for the proposed Radix-8 multipliers known as ABM1 and ABM2. For accelerating the speed of Partial Product addition, Wallace Tree parallel processing was employed.

V.G. Oklobdzija et al. [19] presented a method /algorithm for generating a parallel multiplier (optimized for speed). The method proposed was applicable for any size of the multiplier and was adaptable to any technology, parameters of speed for which are known. It was very easy to use this method in the compilation of silicon and in tools used for logic synthesis. It was found that the proposed method was very efficient as compared to other schemes used for comparison. This paper presents a method and an algorithm for generation of a parallel multiplier, which is optimized for speed. The number of cells used in the reduction of partial products by this method is also minimal.

S.M. Yen et al. [20] designed two new protocols based on Chinese Remainder Theorem (CRT) for preventing RSA signature/ decryption computation from a hardware defect cryptanalysis with the help of Residue Number System. Few remedies such as error detection and simple verification have been used in literature but only a very few are efficient and useful. The protocols were designed after proper testing about security. Advance concepts of error infected CRT estimation and error non infected CRT recombination were proposed.

F. Barsi et al. [21] defined a new approach to various properties (error correcting) of the Redundant Residue Number System (RRNS). Determination of a necessary and sufficient factor for correcting the error of a single digit in RRNS is done. Also, it was shown that correction of a single error infected digit in RNS may be allowed with small repetition and single redundant modulus.

CHAPTER 3

RNS MULTIPLICATION

A number system in which a large number can be expressed in terms of a set of smaller integers is known as Residue Number System (RNS). Basically it is used for fault tolerant computations and faster arithmetic. Three main attributes of RNS which make it suitable for these implementations are: First, it is a non-weighted number system, hence, any fault in any digit does not affect the other digit. Second, there is no significant ordering of digits in RNS. Therefore, any faulty digit can be easily removed, but at the cost of reduction in Dynamic Range. Third, carry propagation which is a limiting factor in addition and multiplication operations, as it increases the delay in arithmetic and adds complexity to the circuit , is absent in RNS.

3.1 Basic architecture of an RNS based processor

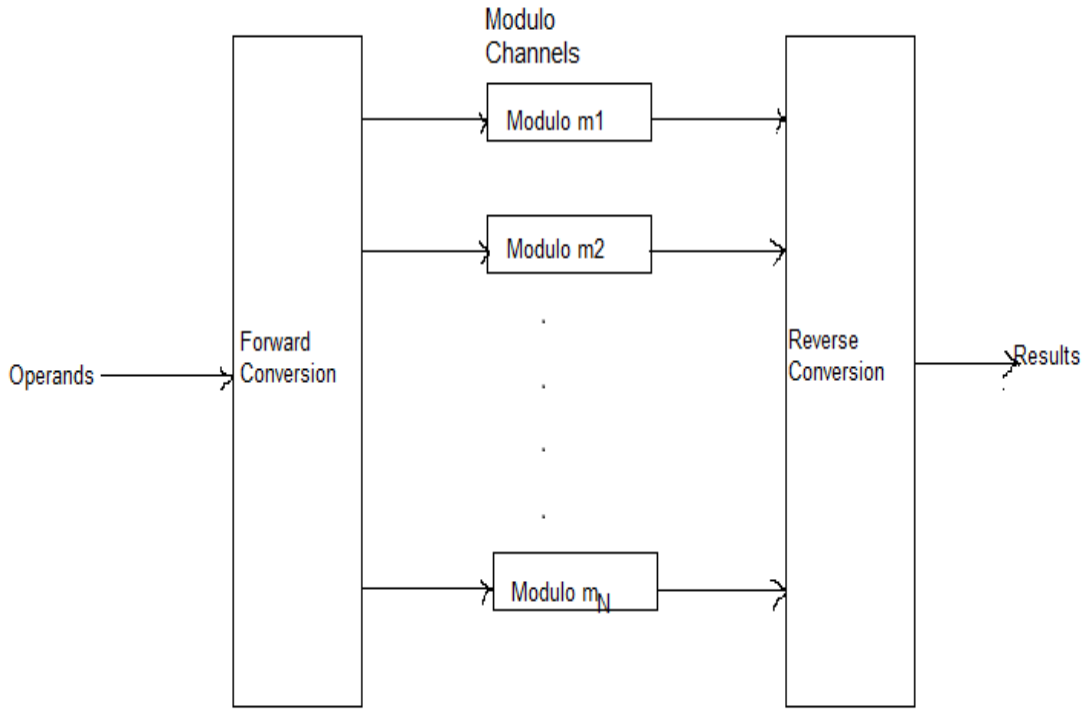


Figure 1.1: Basic architecture of an RNS based processor

Framing an RNS circuit requires conversion from analog to residue and vice versa. In the process of conversion lies the intermediate stage, which is the conversion to binary numbers. Conversion to intermediate stage makes the system complex and

increases its latency. So to frame an RNS processor, we need to frame conversion circuitry that can efficiently perform analog to digital (ADC) conversion and digital to analog (DAC) conversion. To efficiently us the properties of RNS and make them suitable for processing in RNS domain, a smooth conversion between conventional number system and RNS is required.

3.2 Representation of RNS

Linear congruences are used to represent residue numbers. It is written as

$$A \equiv q \bmod r$$

and is read as *A* is congruent to *q* modulo *r*.

Example: decimal number 11 in RNS can be written as

$$11 \equiv 8 \bmod 3$$

$$11 \equiv 5 \bmod 3$$

$$11 \equiv 2 \bmod 3$$

Here 8, 5 and 2 are residues of number 11 with respect to moduli 3.

The RNS uses positional bases, called moduli that are relatively co-prime to each other, e.g. 2, 3, 5. To convert a conventional weighted number (A) to residue system, we simply take the residue of *A* with respect to each of positional moduli. Example of RNS number shown below:

To convert the decimal number 47 to {5, 3, 2} RNS

$$R_5 = 47 mod 5 = 2$$

$$R_3 = 47 mod 3 = 2$$

$$R_2 = 47 mod 2 = 1$$

The decimal number 47 is represented by [2, 2, 1] in above mentioned RNS.

Table 3.1. Some natural numbers and their corresponding residue numbers [1]

Natural Numbers	2537	Natural Numbers	2537	Natural Numbers	2537
0	0000	7	1120	14	1240
1	1111	8	0231	15	1001
2	0222	9	1042	16	0112
3	1033	10	0103	17	1223
4	0144	11	1214	18	0034
5	1205	12	0025	19	1145
6	0016	13	1136	20	0206

The main advantage of RNS is the absence of carries between moduli in addition and multiplication. Arithmetic is closed (done completely) within each residue position, i.e. carry propagation is limited to residue position. Therefore it is possible to perform addition and multiplication on long numbers at the same speed as on short numbers, since the speed is determined by largest modulus position. But, in conventional linear weighted number system, an operation on long words is slower due to the carry propagation. The RNS is defined in terms of a set of relatively prime moduli.

If *M* denotes the moduli set, then

$$\boldsymbol{M} = \{m_1, m_2, \cdots, m_n\} \tag{1}$$

$$\mathrm{GCD}(m_i, m_j) = 1, \ for\, i \neq j \tag{2}$$

The dynamic range DR is

$$DR = m_1 m_2 \cdots m_n \tag{3}$$

$$A \xrightarrow{RNS} (a_1, a_2, \cdots, a_n) \tag{4}$$

$$B \xrightarrow{RNS} (b_1, b_2, \cdots, b_n) \tag{5}$$

$$C = a \circ b \tag{6}$$

Where $\circ$ represent some mathematical operation like addition, subtraction and multiplication

$$C \xrightarrow{RNS} (c_1, c_2, \cdots, c_n) \tag{7}$$

For processing the data in RNS, data needs to be converted to RNS first. The conversion between conventional number system and RNS is also possible.

3.3 Types of RNS

Residue number system can be divided into two categories based upon conversion. Any conventional number system can be converted to RNS and vice-versa. The two types of RNS are discussed below:

3.3.1 Forward RNS

The process of converting conventional numbers to RNS is known as Forward conversion. A special moduli set i.e. $\{2^n-1, 2^n, 2^n+1\}$ is used often for forward conversion, as it makes the process simple and faster. However, for applications requiring high dynamic range, this moduli set does not provide efficient results. Also, for high dynamic range, the moduli set needs to be large, which leads to low performance of arithmetic units in every modulo channel.

3.3.2 Backward RNS

Similarly, converting back data from RNS to conventional numbers is known as Backward conversion. Reverse conversion algorithms are based either on Chinese Remainder Theorem (CRT) or Mixed-Radix Conversion (MRC).In every reverse convertor, a large modulo adder is required at last stage, which makes the system complex. Reverse conversion is one of the most problematic operation in RNS which limits its use to a great extent. Reverse conversion process requires a large modulo adder at end stage which makes the circuitry complex.

The Chinese Remainder theorem is discussed below.

Chinese Remainder Theorem (CRT)

For a moduli set $(m_1,m_2,.....m_n)$ which contains relatively prime numbers, then any number such that $A < S$, the residue set written as $\{A \bmod m_i | 1 \leq i \leq n\}$ is solitary.
Here

$$S = \prod_{i=1}^{n} m_i \tag{8}$$

$$x_i = |A|_{mi} \tag{9}$$

The integer A, therefore, is represented as

$$A = \left| \sum_{j=1}^{n} y_j a_j \right|_S \tag{10}$$

Where y_j is a multiple of all the m_i ($i \neq j$), and $|a_j y_j|_{mj} = |A|_{mj}$, for all j [18].

3.4 Basic calculus in RNS

In this section, discussion about basic algebra existing in RNS is discussed. These properties need to be discussed for better understanding of RNS algebra. Some operations like addition, subtraction, multiplication, additive inverse, multiplicative inverse and some difficult operations like division and scaling are discussed here.

3.4.1 Addition/Subtraction

Adding/Subtracting numbers in RNS is basically adding the residues of these numbers with respect to a particular moduli. Suppose two numbers A and B are to be added in RNS. Consider the moduli set U= $\{m_1, m_2, \ldots, m_n\}$

The numbers A and B in RNS representation will be written as

A= $\{a_1, a_2, \ldots, a_n\}$ and B=$\{b_1, b_2, \ldots, b_n\}$

The addition/subtraction can now be expressed as:

$$C = A \pm B = \{ c_1, c_2, \ldots, c_n \} \tag{11}$$

Where $c_i = (a_i \pm b_i) \bmod m_i$

Addition/Subtraction is distributive in modulo operation

$$|X \pm Y|_m = ||X|_m \pm |Y|_m|_m \tag{12}$$

3.4.2 Multiplication

Similar to addition/subtraction, multiplication in RNS can be easily carried out. Multiplication in RNS is the multiplication of residues of two residues with respect to a particular moduli. Suppose two numbers A and B are to be added in RNS. Consider the moduli set U= $\{m_1, m_2, \ldots, m_n\}$

The numbers A and B in RNS representation will be written as

A= $\{a_1, a_2, \ldots, a_n\}$ and B=$\{b_1, b_2, \ldots, b_n\}$

The multiplication thus is expressed as:

$$C = A \times B = \{ c_1, c_{2,\ldots..}, c_n \} \quad (13)$$

Where $c_i = (a_i \times b_i) \bmod m_i$

Addition/Subtraction is distributive in modulo operation

$$|X \times Y|_m = ||X|_m \times |Y|_m| \quad (14)$$

3.4.3 Additive Inverse

In RNS, the additive inverse is $\bar{a}$ of a residue a, is expressed as shown below:

$$a + \bar{a} = 0 \quad (15)$$

Additive inverse is used mainly in subtraction, but, can also be used for distinct residues or for the whole system. The additive inverse also exist generally for a number. Also, this inverse is unique for any residue. It is similar to 2's compliment in binary number system representation. Simply, it can be realized through an elementary operation given as:

$$\bar{a} = |m - a|_m \quad (16)$$

3.4.4 Multiplicative Inverse

The multiplicative inverse is analogous to the reciprocal in conventional arithmetic. It is defined as follows. For a non zero integer a, if

$$|a \times a^{-1}|_m = 1 \quad (17)$$

then a^{-1} is the multiplicative inverse of a, with respect to the modulus m.

Here a and m have no common factor. Multiplicative inverse of a number cannot be determined by any general expression. For prime m, a theorem named as Fermat's Theorem may be used sometimes to determine the multiplicative inverse. This theorem states that for a prime modulus m and for a non-negative integer *'b'*

$$|b^m|_m = |b|_m \quad (18)$$

Here b is not a multiple of m.

3.4.5 Division

Division operation is one of the most difficult arithmetic operation as compare to others. It becomes even more difficult to specifically define the meaning of division in

RNS. But, defining RNS can somehow become possible according to the proceedings noticed above for addition/ subtraction and multiplication operation i.e., defining operation for the residues and extending it further to the triple of residues. But, two complications can arise immediately if we try to do so. These are as follows: first, division in residue and normal division are in consonance when after the division process, the quotient is an integer and secondly, the evidence that there is no multiplicative inverse for zero.

Division in conventional notation can be expressed by the equation given below

$$\frac{a}{b} = c \tag{19}$$

which signify that

$$a = b \times c \tag{20}$$

In residue number systems (RNS), the elementary relationship is not plain equality but congruence. Hence, for this system (RNS), this last equation does not hold significantly. Suppose the RNS equivalent of the following equation in RNS, for residues *a* and *b*, the congruence

$$b \times c \equiv a \ (mod\ m) \tag{21}$$

Multiplying both sides by the multiplicative inverse of b^{-1}, we get

$$c \equiv a \times b^{-1} \ (mod\ m) \tag{22}$$

In RNS, divergent to the analogous situation in conventional form of arithmetic, multiplication by a multiplicative inverse is not identical to division at all times.

3.4.6 Scaling

From the above discussion it is clear that division operation is difficult to implement in residue number system. But, division can be an easier process if the divisor belongs to the moduli set or is the result obtained after multiplication of several moduli. It should not be a multiple of a power of a modulus. This operation is called as a scaling operation. Scaling is identical to division operation by a power of two in conventional binary system, where the division operation is executed by making shifts in the dividend towards the right. In residue number system scaling is not very much simple but it is comparatively easier to perform compared to division in RNS. Scaling is frequently used to prevent overflow by reducing the dynamic range of RNS variables.

Scaling of a number A by a positive integer B, in RNS, w.r.t. moduli m_i is expressed as:

$$\left\lfloor \frac{A}{B} \right\rfloor_{mi} = \left| \frac{A-a}{B} \right|_{mi} \tag{23}$$

3.5 Advantages of RNS

The benefits of RNS are as discussed below:

Diminished power: If mini arithmetic sections are used to design and implement RNS processors, then switching activities can be reduced in every channel. By reduction in the switching activities, the power consumption also gets reduced. The dynamic range on the other hand gets reduced because it is proportional to the switching activities directly.

Diminished Complexity: As already discussed, a large number is broken down to small residues in RNS, the processor arithmetic units also gets reduces, thereby reducing the complexity of the system. This leads to simplification of overall design of the system.

Elevated Speed: In conventional number system carry if generated is propagated from LSB to MSB creating a delay in processing. Whereas in RNS as already discussed, a large number is broken down into smaller integers (residues), leading to absence of carry. Thus, the RNS processor becomes faster due to absence of carry propagation.

Disclosing errors and correcting them: RNS is a non weighted system. Any fault in any digit does not affect the other digit. In case there is a fault in some digit, that digit can easily be removed without altering others.

3.6 Disadvantages of RNS

The various benefits of RNS have been discussed and it was found that RNS offers great advantages, especially in terms of fast speed and reduced power .These cons make RNS suitable for implementation in various applications. Besides these advantages, RNS possess few drawbacks which limit its use in various applications. The reasons are

Firstly, not all the arithmetic operations in RNS are easy. Division, square root, Scaling, detection of sign and comparison are the most difficult operations to be implemented in RNS in comparison to the simple conventional (analog or digital) number system. Therefore, to design an Arithmetic Logic Unit (ALU) based on RNS performing all operations of arithmetics is a very difficult job.

Secondly, conversion from conventional number system to RNS or vice versa can be a very complex circuitry, which can lead to increased delay in processing i.e. lowering of speed. Thus, while designing an efficient RNS processor, it is very important to have to proper and efficient conversion circuitry.

3.7 Applications of RNS

The various advantages of RNS as discussed earlier, makes it suitable for different applications.RNS is very useful in applications demanding high speed and low power consumption. The various applications involve Digital Signal Processing (DSP)[19],processing of images[20],Right Shift Accumulate (RSA) algorithms[21],receivers used in communication[22],and is fault tolerant applications[23,24].Further in DSP,RNS is used for designing digital filters(Finite Impulse Response(FIR) or Infinite Impulse Response(IIR)).Digital filters are used in interpolation, decimation, reduction in noise, and for band splitting.

Another application in DSP of RNS is Discrete Fourier Transform (DFT) which has various applications in the field of engineering. Carry free characteristic of RNS makes it useful for fault tolerant applications. In brief RNS is useful in various applications because of its various advantages.

CHAPTER 4
BOOTH MULTIPLIER

4.1 Multipliers

Multiplier, a factor of proportionality, measures how much a variable gets altered in response to an alteration in some another variable. Example, suppose there is a change of 1 unit in variable *x*, which changes another variable *y* to by N units. Then here the multiplier is N. Multipliers are an alternative for repetitive addition and are therefore of great use. Multipliers are the basic building blocks of the microprocessor. Multipliers have various uses in various fields of microprocessor design. Multipliers are non-memory sub-blocks of microprocessors .They have the largest size and time delay that creates a great impact on the cycle time. Multipliers are frequently used and provide a great significance in Digital Signal Processing (DSP) applications for running high speed calculations. Multiplication (denoted by this cross symbol "×") is an operation of mathematics for scaling one number by the second. In standard method, if two n-bit numbers are to be multiplied then it requires n^2 multiplications. Multipliers are becoming necessary in modern electronic systems, that run complex high speed calculations and have become a basic building block especially fields of DSP applications and multiprocessors. With advancement in science and technology, many researchers have already tried and some are trying multiplier designs which can offer either of the following design objectives namely- fast speed, less consumption in power, less area requirement, a layout regularity or even combination of these in a single multiplier making them relevant for fast and compact VLSI implementations. "Add and Shift" in multiplication is a common multiplication method. In parallel multipliers, an important factor that depict the fulfillment and realization of the performance of a multiplier, is the number of partial products that are added. Multiplication involves two basic steps: 1) generating partial products, 2) adding these partial products. For reducing the number of partial products, Modified Booth algorithm is the most prominent algorithm. For reducing number of sequential addition stages and to achieve speed enhancement and improvement Wallace Tree algorithm can be used .Further, if we combine both these algorithms in one multiplier, advantage of both can be seen in a single multiplier. However, increasing in parallelism, amount of shifting that occur between partial products and intermediate

sums for addition will rise which might result in reduction of speed, growth in silicon area due to uncertainty of structure and also escalated power consumption because of increment in interconnections resulting from complex routing. On the other hand "serial-parallel" multipliers have to compromise on speed for achieving improved performance in terms of area and power consumption requirements. The selection among a serial or parallel multiplier will depend on our requirement of application. An example of simple multiplication is given below:

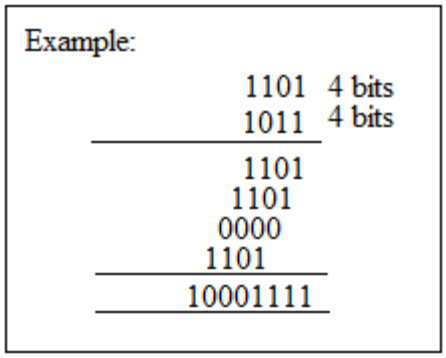


Figure 4.1: Example representing simple binary multiplication

4.2 Modulo Multiplier

In single modulo multipliers there was a need of a separate processor for performing a particular modulo operation. This resulted in more complex circuitry. Also the requirement for area increased. Moreover larger the number of processors, larger was the power dissipation encountered. So a flipside to this is multi modulus multiplier which overcomes all these problems. In multi modulus multiplier a single processor is used to perform the modular operation of the multi moduli set .This is an efficient method as it reduces area requirement, power consumption and speed up the operation. The literature studied mainly focuses on the moduli set $\{2^n-1, 2^n, 2^n+1\}$. Base 2 is chosen as it makes the calculations simple and easier. The moduli sets with 2^n are special moduli sets. $\{2^n-1, 2^n, 2^n+1\}$ is one of them. These special moduli are usually referred to as low cost moduli as forward conversion and reverse conversion process from their residue can be accomplished very easily and it does not need any complex operations like multiplicative inverse or multiplication.

4.3 Classification of Multipliers

- Serial Multipliers
- Parallel Multipliers
- Serial Parallel Multiplier

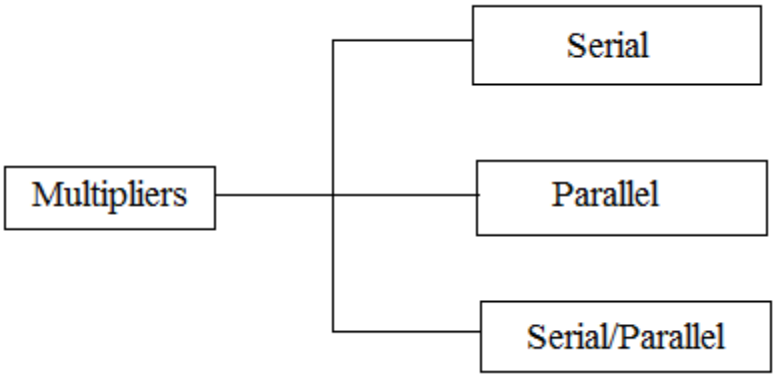


Figure 4.2: Block diagram of Multiplier Types

4.3.1 Serial Multiplier

Serial multipliers are used when we need to take special care about area and power consumption. The circuit of serial multiplier uses one adder for adding the m * n partial products. The circuit for m=n=4 is shown in the figure given below. Input multiplicand and multiplier have to be arranged in a proper special manner which can be synchronized with circuit behaviour as depicted on the figure. The inputs could also be conferred at altered rates depending on what the length of the multiplicand and multiplier is. Here two clocks are used, one for the clocking of data and another one for reset. For delay, the first order approximation of the delay is given as O (m,n). With this arrangement of circuit, the delay is given as D = [(m+1)n + 1] tf.

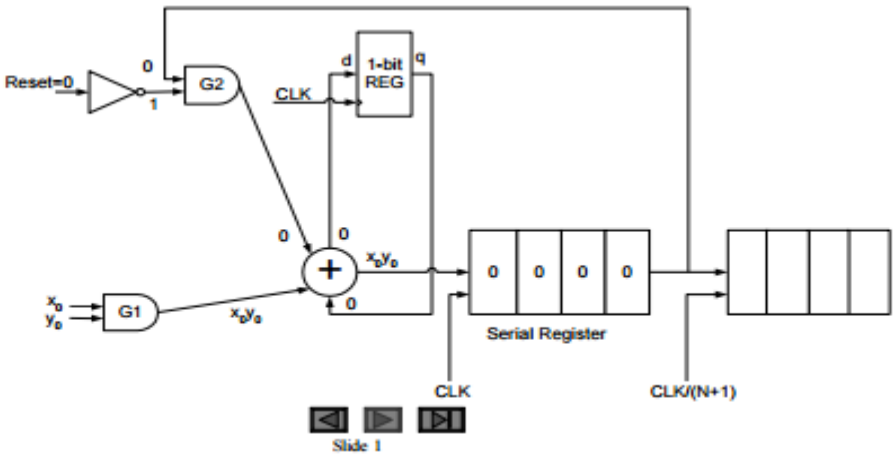


Figure 4.3: Block diagram of Serial Multiplier

As shown in the diagram, the every individual partial product is formed individually. The addition of partial products is done as intermediate values of PPs. PPs addition is

stored in the DFF, exchanged in circles and then added together with the new PPs formed. This approach is suitable only for small values of M or N.

4.3.2 Parallel Multiplier

These multipliers have a good driving characteristics. It consists of a numbers of full adders. Every full adder in the multiplier, performs same number of computations. The ripple carry propagates the input signal, by a constant and for same number of times. A very common example example of parallel multiplier is Baugh-Wooley multiplier.

4.3.2 Serial/Parallel Multiplier

The commonly used/general for the serial/parallel multiplier is represented in the figure given below. One number is made input to the circuit in parallel while another number is inputted serial. In every cycle N partial products are formed. On subsequent cycles, each cycle performs addition of one column in multiplication table of M*N partial products. After M+N cycles, the results are stored finally in output register. For M=N, the area requirement is N-1.

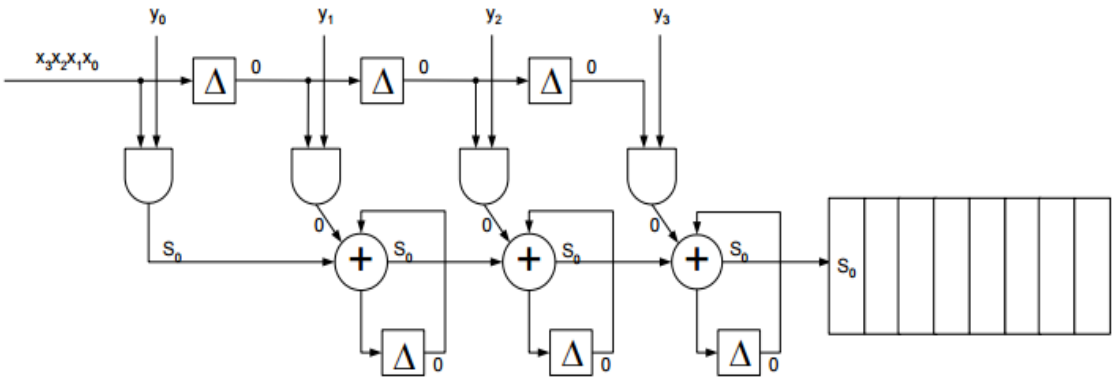


Figure 4.4: A general serial/parallel multiplier

4.4 Multiplication Algorithm

The following steps are involved in a simple multiplication algorithm:

- If the Least Significant Bit (LSB) of Multiplier is '1', then multiplicand is added into an accumulator.

- Now, Shift multiplier to the right by one bit and multiplicand to the left by one bit.

- When the multiplier bits become zero, stop the process. It is now clear that the multiplication of numbers is changed to addition of numbers. A serial adder having

less hardware is used if the Partial Products add serially. Only one combinational circuit, using a parallel multiplier, can be used to add all partial products. However it is also possible to use some compression techniques to reduce number of partial products before performing addition.

4.4.1 Booth's Multiplication Algorithm

Booth's algorithm performs the examination of adjacent pairs of bits of the N-bit multiplier Y in signed two's compliment notation , and it includes an implicit bit below the least significant bit(LSB), $y_{-1} = 0$. For each y_i bit, whereas i running from 0 to N-1, the bits y_{i-} and y_i are considered. Where these two bits become equal, the product accumulator i.e. P is left unchanged. Where $y_{i-1} = 1$ and $y_i = 0$, the multiplicand times 2^i is then added to P; and where $y_{i-1} = 0$ and $y_i = 1$, the multiplicand times 2^i is then subtracted from P. The final value of P is thus a signed product. The multiplicand and the product are not specified; typically, these both are also in two's complement representation, just like the multiplier, but any number system that supports subtraction and addition will work as well. As stated here, the determination of the order of the steps is not done. Typically, it is proceeded from LSB to MSB, starting from $i = 0$; the multiplication by 2^i is then consistently replaced by the incremental shift of the product accumulator P, to the right between steps; low bits can thus be shifted out, and hence subsequent subtractions and additions can then be done on the highest N bits of P. There are many deviations and developments on these details. The Booth's algorithm is often described as conversion of strings of 1's in the multiplier to a high-order +1 and a low order -1 at the end of the string. When the string runs through the MSB, there is no high-order +1, and the net effect is that it leads to the interpretation as a negative of the appropriate value [10].

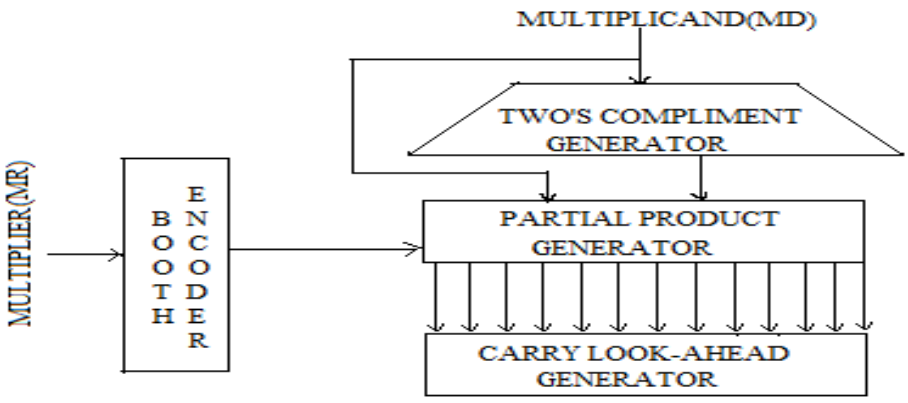


Figure 4.5: Architecture of Booth Multiplier [11]

A simple example of Booth multiplier is illustrated in table 4.1.

Table 4.1: Example representing Booth Algorithm

A	**Q**	**Q-1**	**M**		
0000	0101	0	0111	**Initial value**	
1001	0101	0	0111	A ← A-M	First cycle
1100	1010	1	0111	Shift	
0011	1010	1	0111	A ← A+M	Second cycle
0001	1101	0	0111	Shift	
1010	1101	0	0111	A ← A-M	Third cycle
1101	0110	1	0111	Shift	
0100	0110	1	0111	A ← A+M	Fourth cycle
0010	0011	0	0111	Shift	

4.5 Types of Booth Multiplier

We will discuss the following types:

- Radix-2 : pairing of two consecutive bits of the multiplier
- Radix-4 : pairing of three consecutive bits of the multiplier
- Radix-8 : pairing of four consecutive bits of the multiplier

4.5.1 Radix-2

- A number with the least difference between its consecutive bits is chosen out of the two numbers and is treated as a multiplier. For example if we have two numbers i.e., 1100 — From 1 to 1 no change, 1 to 0 one change, 0 to 0 again no change, and hence there is only one change on this number, similarly in 1010 — From 1 to 0 one change, 0 to 1 one change, 1 to 0 another change, so there are three changes on this number. Therefore, in multiplication of 12 × 10(for example), where 12 (1100) is the multiplier and 14(1010) is the multiplicand.
- Let X = 1100 (multiplier) Let Y = 1010 (multiplicand) .Take the 2's complement of Y and call it –Y = 0110.
- Load the value of X in the table.
- Load 0 for X-1 initially. Later its value should be the previous first least significant bit of X.

- Load zeroes in U and V columns which would contain the product of X and Y at the termination of operation.
- Make four rows for each cycle are we are multiplying numbers having four bits.

Table 4.2: Encoding of Radix-2 Booth Multiplier

X	X-1	ACTION
0	0	No action
0	1	Addition
1	0	Subtraction
1	1	No action

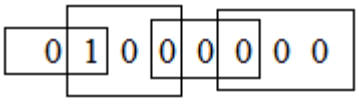
4.5.2 Radix-4

Radix-4 Booth Multiplier is similar to Radix-2 multiplier except that here three consecutive bits are paired unlike radix 2 where two consecutive bits of the multiplier were paired.

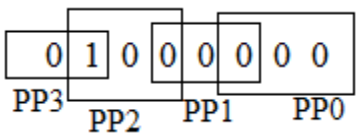
The following are the steps for its multiplication:

The Booth Algorithm for radix-4 is given below:

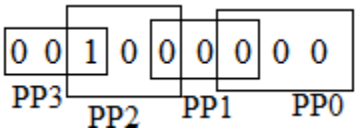
- Consider two inputs of eight bits each i.e. 32(00110001)as A and 64(01000000)as B.
- Affix a 0 to the LSB of B.
- Start pairing the bits in groups of three on B.



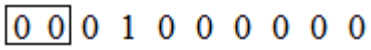
- Now, mark every group as a partial product.



- If the PPs do not complete their group of three, then add more bits i.e. if MSB is 1, add ones, if MSB is 0, add zeroes respectively to the left of MSB. Now denote this as a multiplier M.



- Affix two zeroes/ones to MSB of A depending on which bit is present at MSB of A and denote it as multiplicand N.



- Now, represent PP of M according to the rule of radix-4 encoding given in table below:

Table 4.3: Encoding of Radix-4 Booth Multiplier

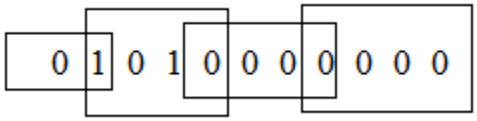
BLOCK	PARTIAL PRODUCT
000,111	0
001,010	1a
011	2a
100	-2a
101,110	-1a

4.5.3 Radix8

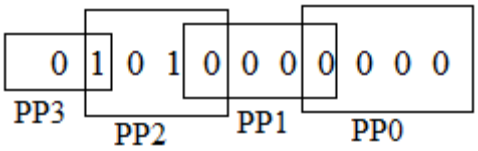
Unlike Radix-4 the Radix-8 strings 4 bits of the multiplicand '*a*' at a time.

The Booth Algorithm for radix-8 is given below:

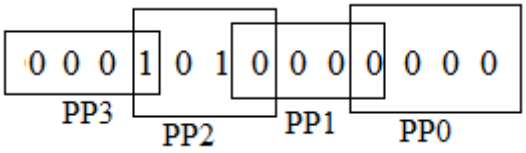
- Consider two inputs of ten bits each i.e. 135(0010000111) as A and 320(0101000000) as B.
- Affix a 0 to the LSB of B.
- Star pairing the bits in groups of four on B.



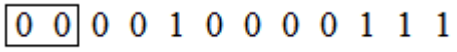
- Now, mark every group as a partial product.



- If the PPs do not complete their group of four, then add more bits i.e. if MSB is 1, add ones, if MSB is 0, add zeroes respectively to the left of MSB. Now denote this as a multiplier M.



- Affix two zeroes/ones to MSB of A depending on which bit is present at MSB of A and denote it as multiplicand N.



- Now, represent PP of M according to the rule of radix-8 encoding given in table below:

Table 4.4: Encoding of Radix-8 Booth Multiplier

BLOCK	PARTIAL PRODUCT
0000,1111	0a
0001,0010	+1a
0011,0100	+2a
0101,0110	+3a
0111	+4a
1000	-4a
1001,1010	-3a
1011,1100	-2a
1101,1110	-1a

Here

- 2a is obtained by shifting a to the left one time.
- 4a is obtained by shifting a to the left two times.
- -a is obtained by taking 2's complement of a.

- -2a is obtained by shifting –a to the left once.
- -4a is obtained by shifting -1 to the left twice.
- 3a is obtained by adding 2a and a.
- Similarly, to generate -3a we add -2a and –a.

CHAPTER 5

IMPLEMENTATION AND RESULTS

A simple Booth multiplier for Radix-2, Radix-4 and Radix-8 for the bits 8, 16 and 32 were designed using Verilog HDL, simulated and synthesized on Xilinx 14.5, targeting Spartan 3E FPGA device. Area and Delay observations were made. From the results it was observed that Radix-4 gives the best results for applications requiring fast speed and low area consumption. Later an RNS based Booth encoded multiplier was designed for Radix-4. RNS deals with small set of integers, so RNS was applied to radix-4 Booth encoded multiplier and results were observed. It was noted that this multiplier showed further improvement in area and speed. Incorporation of pipelining in these designs accelerated the speed further. The area consumption was also reduced.

5.1 Results of Booth Multiplier

The simulation results of Radix-2, Radix-4 and Radix-8 multiplier for 8, 16 and 32 bits are shown in figures 5.1, 5.2 and 5.3 respectively.

8 bit multiplier

Figure 5.1: Simulation result of 8-bit multiplier

The simulation result of two eight bit numbers when multiplied is shown in figure 5.1. Here x_1 and y_1 are the eight bit multiplicand and multiplier respectively. Multiplication of these two numbers generates a 16 bit output designated as z here. The results are shown in decimal form here. The output results of multiplication are same for each radix i.e. radix-2, radix-4 and radix-8 for the same number of bits. Only the area consumption and speed is variable.

16 bit multiplier

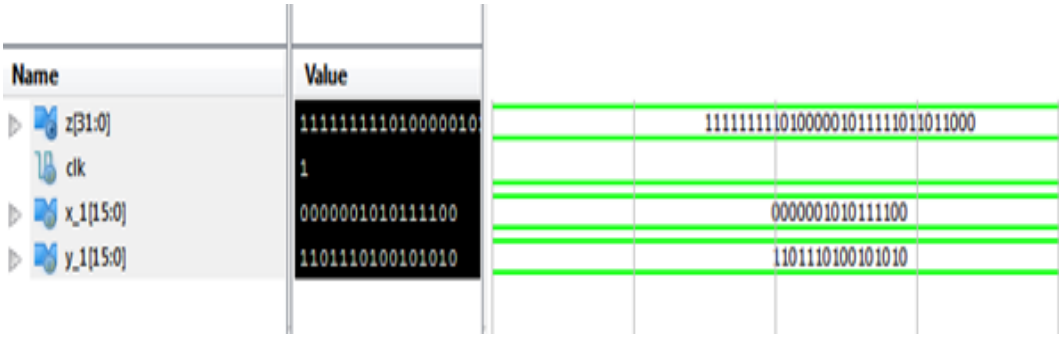


Figure 5.2: Simulation result of 16-bit multiplier

The simulation result of two sixteen bit numbers is shown in figure 5.2. Here x_1 and y_1 are two sixteen bit inputs to a multiplier. On multiplication, a thirty two bit output z is produced. Here the results are depicted in binary form.

32 bit multiplier

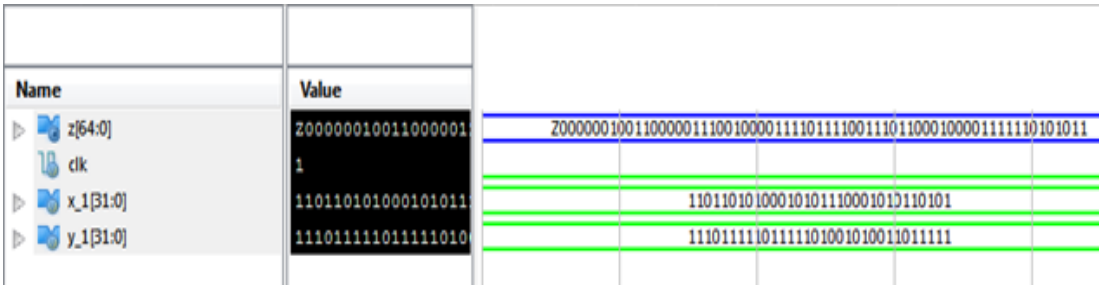


Figure 5.3: Simulation result of 32-bit multiplier

The simulation result of two thirty two bit numbers is shown in figure 5.3. Here x_1 and y_1 are two thirty two bit inputs to a multiplier. On multiplication, a sixty four bit output z is produced. The results shown here are in binary form.

Table 5.1: Area and Delay report of Radix-2, Radix-4 and Radix-8 Booth encoded multiplier

		Radix 2		Radix 4		Radix 8	
Device Utilization Summary	Bits	Used/ Available	Utilization (%)	Used/ Available	Utilization (%)	Used/ Available	Utilization (%)
Number of LUT's	8	192/4656	4	63/4656	1	120/4656	2
	16	696/4656	14	237/4656	5	403/4656	8
	32	2674/4656	57	468/4656	10	725/4656	15
Timing Summary							
Maximum Combinati -onal Delay(ns)	8	30.113		20.225		15.608	
	16	44.302		29.417		36.560	
	32	60.558		30.668		48.358	

The comparison of the area consumed in different radices for the same number of bits is done in the table 5.1. Also the comparative study of time taken for multiplication i.e. final time delay is done for same bits in different radices. We found out that as the number of bits increase in same radix, the area consumption and delay increases. But for the same number of bits, Radix-2 consumes maximum area and takes maximum time to generate an output. The area utilization and delay for Radix-8 are less than that of Radix-2 but more than that of Radix-8. Therefore Radix-4 provides the best use in areas requiring fast speed and low area.

The device utilization and timing report for the three radices i.e.Radix-2, 4, 8 for the same number of bits as shown in table 5.1 is presented graphically in Figure.5.4 and Figure.5.5 respectively.

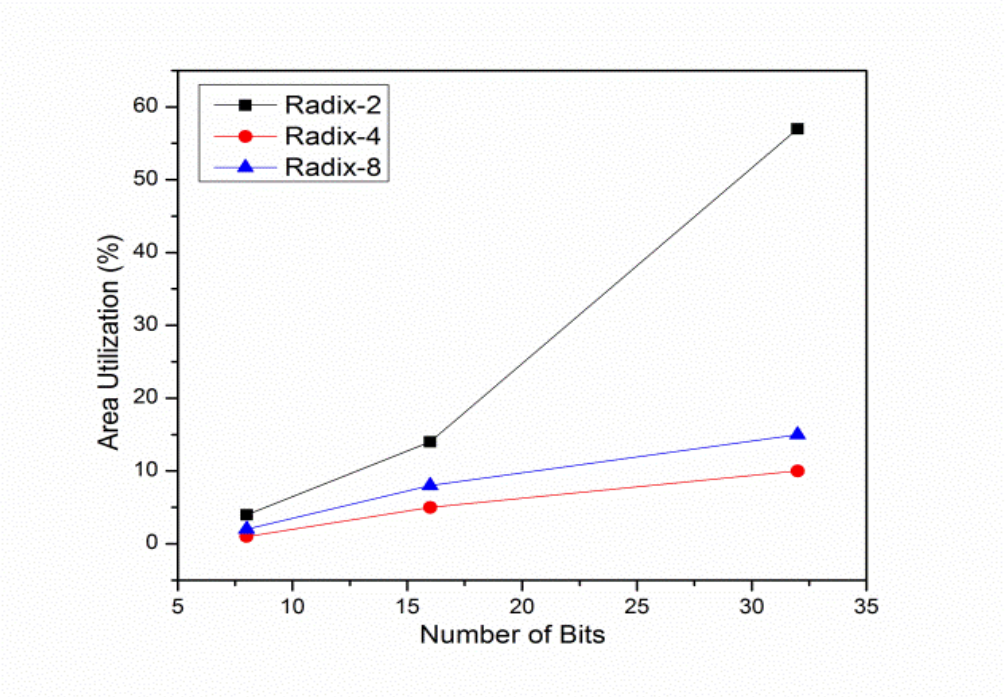


Figure 5.4: Comparison between Radix-2, Radix-4 and Radix-8 on the basis of Area Utilization.

From the figure 5.8, it can easily be concluded that Radix-2 has the maximum area utilization compared to Radix- 4 and Radix-8. Radix-4 has the minimum consumption of area and the area used by Radix-8 is less than that consumed by Radix-2 but more than that of Radix-4. This trend in results may be due to the increased complexity in the design of radix-8 multiplier.

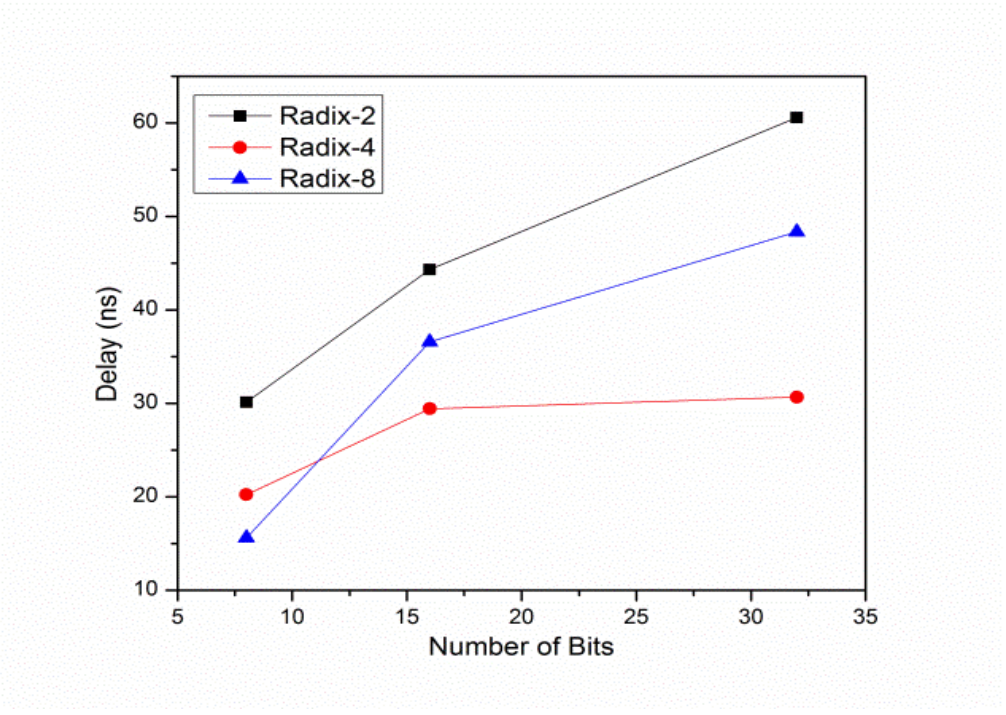


Figure 5.5: Comparison between Radix-2, Radix-4 and Radix-8 on the basis of Delay (ns).

Figure 5.9 shows that among radix-2, radix-4 and radix-8 multipliers, radix -8 has minimum delay in 8 bit multiplier. But for 16 and 32 bits , radix-4 is the fastest among the three.

5.2 Results of Booth encoded RNS multiplier

The simulation results of a Booth encoded RNS multiplier is shown in the figure 5.6. Here ‘a’ and ‘b’ are the inputs, whereas out3, out5, out7, out11 are the outputs obtained after residue multiplication w.r.t. to the moduli’s 3, 5, 7 and 11 respectively.

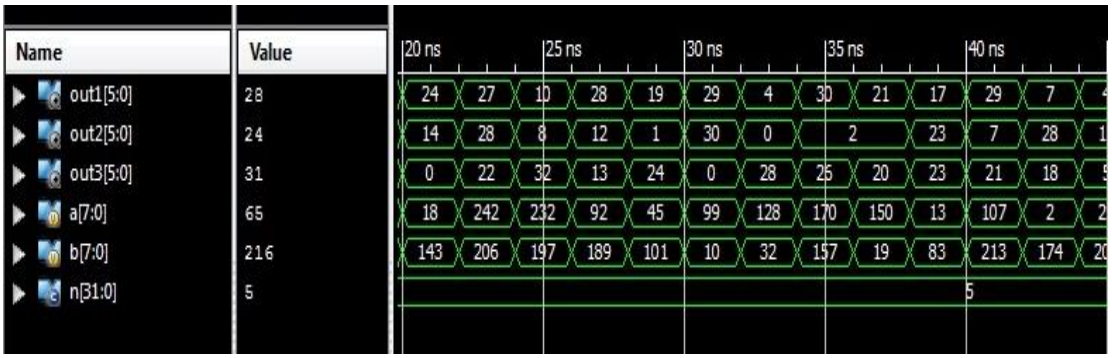


Figure 5.6: Simulation result of an RNS based Booth encoded multiplier

When pipelining was incorporated in Booth multiplier, it was observed that there was a significant increase in speed of the multiplier. The table 5.2 below shows the area and delay report of a simple Booth Multiplier when pipelining is applied to it. It has been observed that pipelining improves overall performance of system, both in terms of speed and area utilization as compared to a simple Booth Multiplier.

Table 5.2: Area and Delay report of a pipelined Booth Multiplier

BITS	AREA		DELAY(ns)
	USED/ AVAILABLE	UTILIZATION(%)	
8	82/4656	1	16.474
16	150/4656	3	20.621
32	350/4656	7	28.693

When Radix-4 Booth multiplier is implemented for RNS, the results were improved, as the RNS deals with small set of integers. Further when pipelining was incorporated in the architecture, the speed and area of the design enhanced further, as shown in the table 5.

Table 5.3: Area and Delay report of a pipelined Booth encoded RNS Multiplier

BITS	AREA		DELAY(ns)
	USED/ AVAILABLE	UTILIZATION(%)	
8	70/4656	1	8.526
16	112/4656	2	11.056
32	249/4656	5	21.693

A simple comparison of above results i.e. for simple Booth Multiplier and an RNS based multiplier in which pipelining has been incorporated is tabulated below in table 5.4.

Table 5.4: Comparison between area and delay report of a pipelined simple Radix-4 Booth multiplier with RNS based Booth multiplier

BITS	AREA UTILIZATION(%)		DELAY(ns)	
	BOOTH	RNS	BOOTH	RNS
8	1	1	16.474	8.526
16	3	2	20.621	11.056
32	7	5	28.602	21.693

The results observed in table 5.2, 5.3 and 5.4 are presented in the graphs shown below in Figure 5.7 and Figure 5.8.

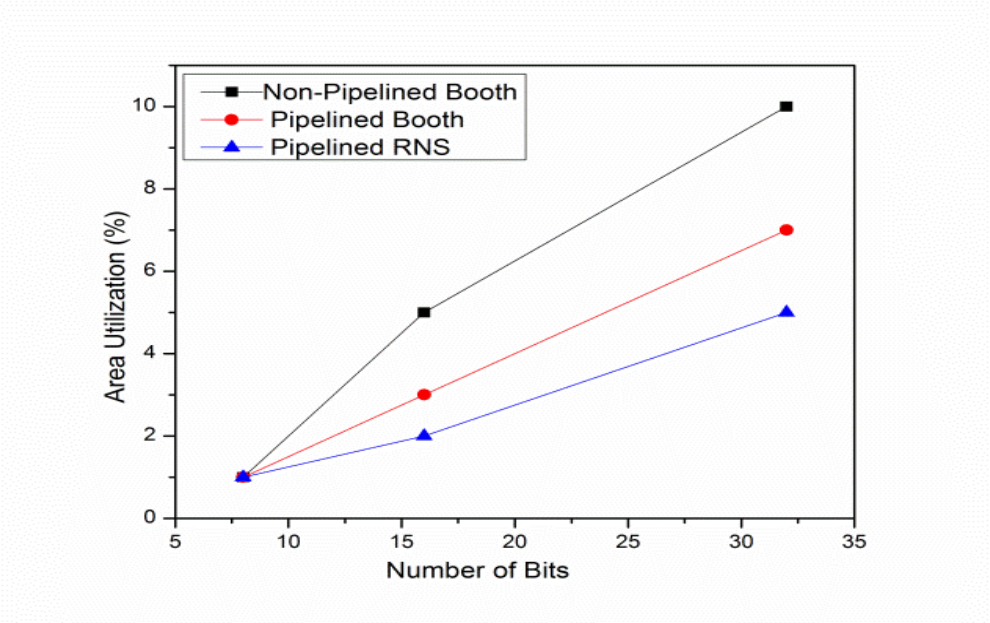


Figure 5.7: Area comparison in pipelined and non-pipelined simple Booth multiplier and RNS based Booth multiplier.

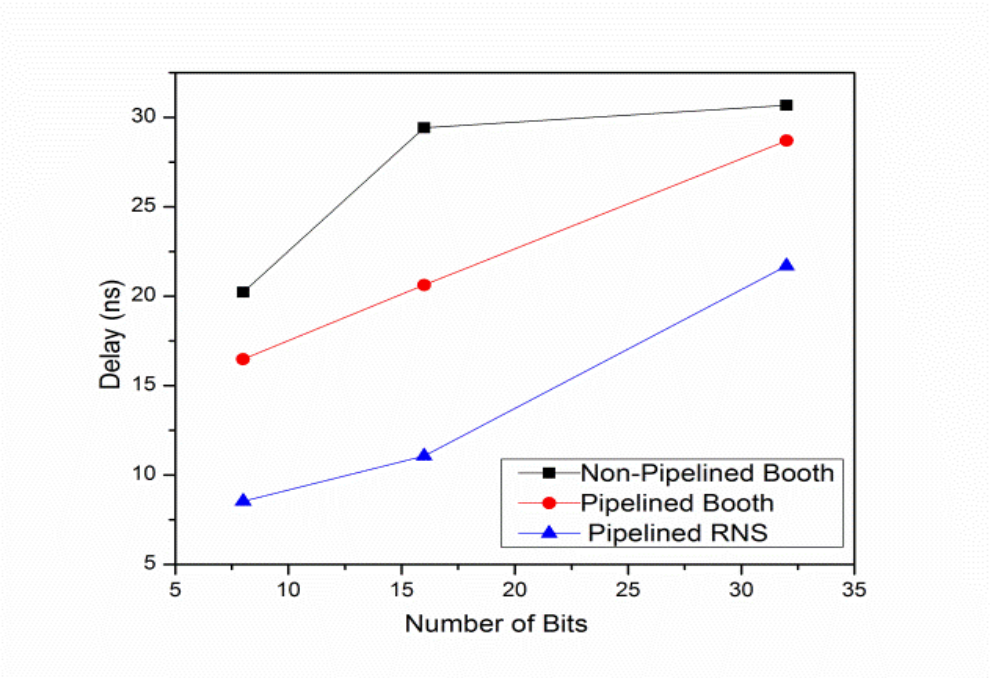


Figure 5.8: Delay comparison in pipelined and non-pipelined simple Booth multiplier and RNS based Booth multiplier

The graph shown in figure 5.8 clearly shows that a pipelined Radix-4 Booth encoded RNS multiplier utilizes the minimum area and is the fastest among simple Radix-4 Booth multiplier and a pipelined Radix-4 Booth multiplier.

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

Conclusion

Implementation and design of three radix-based multipliers(Booth encoded) was done using Verilog HDL (VHDL), synthesized and simulated on Xilinx 14.5, targeted on Spartan 3E FPGA device.The results so obtained clearly depicted that Radix-4 Booth encoded multiplier provides a significant area and speed advantage among the designs of the three radices named above. Also Radix-2 occupies the largest area compared to other two. Radix-4 is the best for applications requiring high speed and low area requirements.

On the basis of above results, Radix-4 was selected and an RNS based booth encoded multiplier (Radix-4) was designed and compared to a simple Booth multiplier for the same number of bits (here 8,16 and 32 bits). Since RNS deals with a small set of numbers, calculations and implementation of these numbers is easier to realize. So we chose RNS for our research work. Area and delay readings were observed for different values of n. A comparison showed significant improvement in terms of speed and area utilization in RNS based Booth multiplier in relation to a simple Booth multiplier of same radix. Pipelining has been incorporated to make the system performance even better.

Future Scope

Booth's algorithm has been illustrated as the best algorithm for the generation of Partial Product, but some new technique can be used to do the same and enhance the results further. Very little work is done for the reverse conversion of RNS, since reverse conversion is a difficult process as the extra adder circuit at the last stage of converter increases the complexity. So efforts can be made to design the same. Also, the main target of the thesis was on the Partial Product generation, emphasis can be laid to identify the techniques on accumulation of Partial Product (PP).

REFERENCES

[1] H.L.Garner, "The Residue Number System," *Proceedings of the Western Joint Computer Conference*, 1959.

[2] K.Y. Khoo, Z. Yu and A.N. Willson, " Improved-Booth Encoding for low-power multipliers," *IEEE International Symposium on Circuits and Systems*, vol. 1, pp.62-65, June 1999.

[3] R.Conway and J.Nelson, "Improved RNS FIR Filter Architectures," *IEEE Transactions on circuits and systems*," vol. 51, no. 1, pp.26-28, Jan 2004.

[4] R.C.Ismail and R.Hussin, "High Performance Complex Number Multiplier Using Booth-Wallace Algorithm," *IEEE International Conference on Semiconductor Electronics*," vol.10, no. 3, pp. 786-790, March 2006.

[5] A.Omondi and B.Premkumar, "Residue Number Systems: Theory and Implementation," London, U.K.: Imperial College Press, 2007.

[6] D.Adamidis and H.T.Vergos, "RNS multiplication/sum-of-squares units," *IET journal and magazine on Computer and Digital Techniques*," vol. 1, no. 1, pp. 38-48, Jan 2007.

[7] Y. He and C.H.Chang, "A New Redundant Binary Booth Encoding for Fast 2n-Bit Multiplier Design," *IEEE Transactions on circuits and systems*," vol. 56, no. 6, pp. 1192-1201, June 2009.

[8] Z.Li, H.Chen and X.Yang, "Research on the Disposal of Negative Partial Product for Booth Algorithm*," IEEE International Conference on Information Theory and Information security*," vol. 20, no. 3, pp. 1115-1117, May 2010.

[9] R.Muralidharan and C.H.Chang, "A Simple Radix-4 Booth Encoded Modulo 2n+1 Multiplier," *IEEE International Symposium on Circuits and Systems*, pp. 1163-1166, May 2011.

[10] M.Sharma and R.Verma, "Disposition (reduction) of (negative) partial product for Radix 4 Booth's Algorithm," *World Congress on Information and Communication Technologies*, pp.1169-1174, March 2011.

[11] S.Shrivastava, J.Singh and M.Tiwari, "Implementation of Radix-2 Booth Multiplier and Comparison with Radix-4 Encoder Booth Multiplier," *International Journal of Emerging Technology and Advanced Engineering*, pp. 14-16, Feb 2011.

[12] R.Muralidharan and C.H.Chang, "Area-Power Efficient Modulo 2n+1 and Modulo 2n-1 Multipliers for {2n-1, 2n, 2n+1} Based RNS", *IEEE Transactions on circuits and systems*, vol. 59, no. 10, pp. 2263-2274, Oct 2012.

[13] D. Chandel, G.Kumawat, P.Lahoty, V.V. Chandrodaya and S.Sharma, "Booth Multiplier: Ease Of Multiplication," *International Journal of Emerging Technology and Advanced Engineering*, vol. 3, no. 3, pp. 118-122, March 2013.

[14] B.Sreekanth and K.Padmavathi, "Modulo Multiplier by using Radix-8 Modified Booth Algorithm," *International Journal of Latest Trends in Engineering and Technology (IJLTET)*, vol. 2, no. 3, pp. 1156-1160, May 2013.

[15] R.Muralidharan and C.H.Chang, "Radix-4 and Radix-8 Booth Encoded Multi-Modulus Multipliers", *IEEE Transactions on circuits and systems*, vol. 60, no. 11, pp. 2940-2952, Nov 2013.

[16] C.P.R.Mejia, V.T.Olaya and J.V.Medina, "8912-Bit Montgomery Multipliers Using Radix-8 Booth Encoding and Coded-Digit", *IEEE Fourth Latin American Symposium On Circuits and Systems(LASCAS)*, pp. 1-4, March 2013.

[17] K.M. Karthik and C.H. Vun, "High performance hardware based RNS-to-binary Converter", *IEEE International Conference on Consumer Electronics (ICCE)*, pp. 147-148, Jan 2014.

[18] H.Jang, J.Han and F.Qiao, “Approximate Radix-8 Booth Multipliers for Low-Power and High- Performance Operation,” *IEEE Transactions on Computers*, pp.1-8, May 2015.

[19] V. G. Oklobdzija and David Villeger, “Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach,” *IEEE Transactions on Computers* ” vol. 45, no. 3, pp. 294-306, March 1996.

[20] S.Yen, S.Kim, S.Lim and S.Moon, “RSA Speedup with Chinese Remainder Theorem Immune against Hardware Fault Cryptanalysis,” *IEEE Transactions on Computers*, vol. 52, no. 4, pp. 461-472, 2003.

[21] F.Barsi and P.Maestrini, “Error correcting properties of redundant residue number systems,” *IEEE Transactions on Computers*, vol. 23, no.9, pp. 915-923.

[22] J.Ramirez, et al., “Fast RNS FPL-Based Communications Receiver Design and Implementation," *Proceedings of the 12th International Conference on Field Programmable Logic*, pp. 472-481, 2002.

[23] W.Wei et al., “RNS application for digital image processing,” *Proceedings of the 4th IEEE international workshop on system-on-chip for real time applications, Canada*, pp. 77-80, 2004.

[24] R.W.Watson and C.W.Hastings, “Self-checked computation using residue arithmetic,” *Proceedings of the IEEE*, vol. 54, pp. 1920-1931, 1966.

PUBLICATIONS

[1] Saguna Goel, Sakshi Bajaj and Amanpreet Kaur, "Design of Booth encoded multi-modulus $\{2^{n-1},2^{n},2^{n+1}\}$ RNS Multiplier," communicated in Journal of VLSI Design Tools and Technology, STM.

Saguna 801461025

ORIGINALITY REPORT

16%	10%	9%	9%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	en.wikipedia.org Internet Source	2%
2	Submitted to Visvesvaraya Technological University Student Paper	2%
3	Omondi, . "Mathematical fundamentals", Advances in Computer Science and Engineering Texts, 2007. Publication	1%
4	www.ijettjournal.org Internet Source	1%
5	ethesis.nitrkl.ac.in Internet Source	1%
6	R. Conway. "Improved RNS FIR Filter Architectures", IEEE Transactions on Circuits and Systems II Analog and Digital Signal Processing, 1/2004 Publication	1%
7	Submitted to Indian Institute of Technology, Kharagpure Student Paper	1%