FPGA IMPLEMENTATION OF 2^r VARIABLE RNS SCALER FOR EXTENDED FOUR MODULI SETS

A Dissertation Submitted In Partial Fulfillment of the Required for the Degree of

MASTER OF TECHNOLOGY

In

VLSI Design

Submitted By

NIKHIL SHARMA

Roll no. 601361016

Under the guidance of

Ms. Sakshi Bajaj

Assistant Professor, ECED

T.U. Patiala

Department of Electronics and Communication Engineering

THAPAR UNIVERSITY

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

PATIALA 147004 (PUNJAB)

July, 2015

DECLARATION

I hereby declare that the work which is being presented in dissertation titled **"FPGA IMPLEMENTATION OF 2′ VARIABLE RNS SCALER FOR EXTENDED FOUR MODULI SETS"** in partial fulfillment of the requirement for the award of degree of Master of Technology in VLSI Design submitted in Electronics and Communication Engineering Department of Thapar university, Patiala is an authentic record of my study carried out as under the guidance of **Ms. Sakshi Bajaj,** Assistant Professor, ECED during 2013-2015.

Date: 13th July, 2015

NIKHIL SHARMA
Roll no.601361016

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

Sakshi Bajaj

Assistant Professor, ECED,
Thapar University,
Patiala 147004, (Punjab)

Countersigned by:

(Dr. Sanjay Sharma)
Head
ECED, Thapar University,
Patiala, 147004

Dean of Academics Affairs
ECED, Thapar University,
Patiala, 147004

ACKNOWLEDGEMENT

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

I convey my sincere thanks to **Head of the Department, Dr. Sanjay Sharma** as well as **PG Coordinator, Dr. Amit Kohli, Associate Professor,** Electronics and Communication Engineering Department, entire faculty and staff of Electronics and Communication Engineering Department for their encouragement and cooperation.

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

(NIKHIL SHARMA)

ABSTRACT

Variable scaling by power-of-two factor is the backbone operation of floating point arithmetic. It is commonly used in fixed-point digital signal processing (DSP) system for overflow prevention. This operation can be easily performed in binary number system, but it is very difficult to perform in Residue Number System (RNS). In this thesis, 2^r variable RNS scalers are designed for important classes of moduli sets that have large dynamic range. These important classes of moduli sets include traditional three moduli set $\{2^n-1, 2^n, 2^n+1\}$ and any extended power-of-two moduli sets m_4 ($\{2^n-1, 2^n, 2^n+1, m_4\}$). In this approach, scaling is done by combining both Chinese Remainder Theorem (CRT) and mixed radix conversion (MRC). Scaling is completely done in RNS domain, no reverse conversion or forward conversion required in this approach. Simple, memory less VLSI architectures are proposed based on the obtained formulation. In hardware design, carry save adders with end around carry (CSA w EAC), carry save adder with complementary end around carry (CSA w CEAC), shifters different modulo adders and a modulo adder circuits are used. Shifters are used to provide variable scaling in RNS. Number of bits shifted in a string are controlled by the value of *r*. These kind of shifter can be implemented with the help of multiplexers.

These RNS scalers are synthesized and simulated using Xillinx ISE 14.5 targeting Spartan 6E FPGA device The relative assessment shows that these architectures provide variable power-of-two scaling but, the increment in area leads to increment in cost, also there is increase in delay as compared fixed scaling techniques. However, this scaler provide wide range of scaling options so that these limitations can be neglected.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBRIVEATIONS

RNS	Residue Number System
DSP	Digital Signal Processing
CRT	Chinese Remainder Theorem
MRC	Mixed Radix Conversion
VLSI	Very Large Scale Integration
CSA w EAC	Carry Save Adder with End Around Carry
CSA w CEAC	Carry Save Adder with Complementary End Around Carry
ISE	Integrated Software Environment
FPGA	Field Programmable Gate Array
LUT	Look Up Table
ROM	Read Only Memory
FIR	Finite Impulse Response
IIR	Infinite Impulse Response
HA	Half Adder
FA	Full Adder
CSA	Carry Save Adder
RCA	Ripple Carry Adder
LSB	Least Significant Bit
PFP	Pseudo Floating Point
CLS	Circular Left Shift
CCRS	Complementary Circular Right Shift
CRS	Circular Right Shift
CPA	Carry Propagate Adder
HDL	Hardware Description Language

ASIC	Application Specific Integrated Circuit
RAM	Random Access Memory
FSM	Finite State Machine
XST	Xilinx synthesis technology
RTL	Register Transfer Logic
VIO	Virtual Input Output

Chapter-1

INTRODUCTION

Residue Number System (RNS) were invented by Chinese scholar Sun Tzu in third century. Sun Tzu was the author of famous book *Art of War*. Sun Tzu posed a mathematical riddle, "Which number will yields the remainders 1, 4 and 6 when divide by 3, 5 and 7 respectively?" In modern terminology, 1, 4 and 6 are *residues* and 3, 5 and 7 are *moduli*. In 1247, another Chinese mathematician, Qin Jiushao, generalized this representation of number in an expression, which is now known as *Chinese Remainder Theorem.*

In the 1950s, RNS were rediscovered by computer scientists, who sought to put them to use in the implementation of fast arithmetic and fault tolerant computing. Three basic properties make them well suited for these. The first is the absence of carry-propagation in addition, subtraction and multiplication. This becomes the most significant factor for high speed operations. The second is, RNS representation is non-weighted representation. An error in any digit-position does not affect other digit-position. And finally the third is, there is no significant ordering of digits. The fault digit-position may be discarded with no effect other than reduction in dynamic range. There are some disadvantages of RNS representation. A complete arithmetic unit which is capable of performing square root, division and comparisons are very difficult to implement in RNS. However, RNS is extremely good for many applications, such as digital signal processing, communications, computer security (cryptography), image processing, speech processing, where high speed and low power consumptions is critical. In these applications, major arithmetic operations are addition and multiplication.

The choice of any arbitrary set of moduli leads to hardware consuming design. So, instead of choosing arbitrary moduli set, special power-of-two moduli sets such as $\{2^n\text{-}1, 2^n, 2^n\text{+}1\}$ or extended are preferred. The major advantages of these moduli sets are arithmetic operations becomes easier and also it can utilize maximum number of bits in particular hardware. The dynamic range of these moduli sets becomes almost

equal to binary number systems for particular value of *n*. All other arithmetic operations which are difficult in arbitrary RNS number become easier in these special moduli sets. For example, scaling was done in arbitrary RNS by LUTs scheme or ROM based designed. But with the emergence of power-of-two moduli sets, scaling operation can be performed with the help of simple combinational circuits.

Residue number system has gained a strong foothold in very large scale integration (VLSI) implementation of application specific digital signal processor. Its inherently modular, parallel and distributive arithmetic not only speed up the data paths dominated by additions, subtractions, and multiplications, but also facilitate to conserve power of arithmetic units without compromising their latency. Such sum-of-products kernels are generally found in DSP algorithms, such as finite impulse response (FIR), infinite impulse response (IIR), fast Fourier transform (FFT), discrete wavelet transform (DWT), etc. In recent researches, techniques such as multi-modulus and multi-functions architectures are suggested to minimize the hardware redundancy as well as to multi-threshold voltage and multi-supply voltages for low power design. RNS computations are faster than its counterpart binary number system because computations are performed in short-word length modulo channel. Also carry propagation does not take place from one channel to other channel. But the major problem in RNS is to design its converter to interface with the physical world. With the emergence of power- of- two modulus set, this problem has been resolved up to some extent.

With the advances in technology, many researchers have tried and are trying to design various arithmetic units for different moduli set which offer following design targets: high speed, low power, less area and different dynamic range.

1.1 Objective

The primary objective of this thesis is to design and implementation of RNS scaler for different moduli set. In this work, RNS scalers are proposed for two different moduli sets for different dynamic range. Scaling was done by one of its moduli or product of two or more moduli [4], [10], [17], [19]. In this report, a variable power-of-two RNS scaler is designed for $\{2^n\text{-}1, 2^n, 2^n\text{+}1\}$ moduli set [25]. This variable scaler design is further explored to extended four- modulus RNS. These designs are fully based on

combinational circuits. No memory unit is required to perform scaling in these scalers.

1.2 Thesis Organisation

Chapter 1: Give the brief history of RNS. Why it is preferred over conventional number system?

Chapter 2: Discusses Literature Review

Chapter 3: Introduction to RNS representation, basic arithmetic operation, forward and reverse conversion, basic properties of RNS.

Chapter 4: Design of variable RNS scaler for two different moduli set.

Chapter 5: This chapter is mainly concerned with FPGA and its design flow

Chapter 6: Shows the simulation and synthesis results for RNS scaler using proposed algorithm.

Chapter 7: Derives the Conclusion and tells about future scope.

Chapter-2

LITERATURE REVIEW

Many papers from the recent researches and developments in the direction of designing the RNS scaler more efficient were studied. A brief summary is as follows:

A.Garica and A. Lloris [2] have proposed an architecture for scaling in the RNS, which leads to faster and simpler implementations of this mathematical operation. This scheme is better than previously proposed algorithms when the set of moduli is not too large. Specifically, this scheme is very suitable for a set of three moduli, where one of them is the scaling constant *K*. Sets of three moduli, with one of them as *K*, can be used in RNS applications which need fast scaling, since this operation is performed in two look-up cycles with three look-up tables; this is a better solution than the one provided by former algorithms, and can be suitable for FPGA or VLSI implementations. On the other hand, this scheme leads to larger memory requirements when the number of moduli increases, which is not restrictive when the use of larger tables is allowed, but can be unsuitable for integrated circuit applications.

A.Garica and A. Lloris [4] have presented some improvements to the existing index calculus RNS multipliers. On the other hand, it proposed a new scheme for scaling based on arithmetic modules instead of look-up tables. They have also concentrated on the pipelining to reduce the critical path delay, including zero multiplication and error detection with a very low additional cost.

N. Burgess [10].has introduced a method for extracting the core of a residue number system (RNS) number within the RNS, this affording a new method for scaling RNS numbers. Suppose an RNS comprises a set of co-prime moduli, m_i, with $\Pi m_i = M$. It describes a method for approximately scaling such an RNS number by a subset of the moduli, $\prod m_j = M_j \approx \sqrt{M}$ with the characteristic that all computations are performed using the original moduli and one other non-maintained short word length modulus.

P. V. A. Mohan [14] has presented a RNS converter for new three moduli set $\{2^{n+1}-1, 2^n, 2^n+1\}$. In this technique, the converter for first modulus is derived using mixed radix conversion (MRC) and for second and third modulus the converters are designed using well known Chinese Remainder Theorem. The implementations of first and second modulus converter have been done by using fully combinational circuit whereas, for third modulus conversion is fully ROM based. The experimental result of the proposed architecture has been compared with the recent converters shown in [14].

K. Yinan and B. Phillips [17] have presented fast scaling technique. Scaling of residues are done by constant *k* which is co-prime with the relative moduli set. Experimental results was compared with look-up-table scaling scheme. Result shows reduction in hardware area complexity without affecting time complexity.

Ramya Muralidharan and Chip-Hong Chang [20] have proposed RNS multiplier. The hard multiple is generated using small word-length ripple carry adders (RCAs) operating in parallel. The carry-out bits from the adders are not propagated but treated as partial product bits to be accumulated in the CSA tree. The effect of the RCA word-length *k*, on the time complexities of each constituent component of the multiplier is analysed qualitatively and the multiplier delay is shown to be almost linearly dependent on the RCA word-length. Consequently, the delay of modulo 2^n -1 multiplier can be directly controlled by the word-length of the RCAs to equal the delay of the critical modulo 2^n -1 multiplier of the RNS. By means of modulo 2^n -1 arithmetic properties, we show that the compensation constant that negates the effect of the bias introduced in this process can be pre computed and implemented by direct hardwiring with no delay overhead for all feasible combinations of *k* and *n*. It is shown that the proposed multiplier lowers the area and power dissipation of the radix-4 Booth encoded modulo multiplier under the delay constraints derived from various high dynamic range RNS multipliers.

Jeremy Yung Shern Low and Chip-Hong Chang 2011[21] have proposed a novel area-efficient, high-speed and precise RNS scalar for the three-moduli set RNS $\{2^n$ -1, 2^n, 2^n +1$\}$. The new scaling algorithm is formulated based on the CRT and it uniquely exploits the number theoretic properties of this moduli set and the scaling factor of 2^n to overcome the complexity associated with the hardware implementation of inter modulo operation. The proposed RNS scalar has an area

complexity of $O(n \log_2 n)$ and a time complexity of $O(\log_2 n)$. The integer scaled output in normal binary number representation is also generated as a by-product of this new formulation. Hence, the expensive residue-to-binary converter can be saved if the result after scaling is also required by a normal binary number system.

L. Sousa and S. Antao [22] have proposed an algorithm for RNS reverse converter for new sets of four moduli $\{2^n + 1, 2^n - 1, 2^n, 2^{2n+1} + 1\}$ and $\{2^n + 1,\ 2^n - 1,\ 2^{2n},\ 2^{n+1} + 1\}$. The dynamic ranges of these moduli-sets are 5*n* and 6*n* respectively. This algorithm is completely based on MRC. The simulation results are performed by targeting Vertex-IV FPGA board. The delay of proposed algorithm has been improved up to 18% and 12% for $\{2^n + 1, 2^n - 1, 2^n, 2^{2n+1} + 1\}$ and $\{2^n + 1, 2^n - 1, 2^{2n}, 2^{n+1} + 1\}$ moduli-sets respectively with respect to conventional RNS converter. Experimental results also show reduction in product of area with square of delay by 25% and 21% with respect to related conventional moduli-sets for 5*n* and 6*n* bits.

Ramya Muralidharan and Chip-Hong Chang [23] have proposed an area –power efficient modulo 2^n-1 and modulo 2^n+1 multiplier employing radix-8 Booth encoding scheme. The hardware cost of hard multiple generator [18] was minimized by customizing the parallel-prefix modulo 2^n-1 and modulo 2^n+1 adders for the efficient computation of hard multiples. The area-delay-power metrics of the proposed modulo multipliers were evaluated and compared against radix-4 Booth encoded and non-encoded modulo multiplier based on the normalized area model and synthesis result from CMOS standard-cell implementation. The RNS multiplies based on these modulo lower both implementation of area and total power dissipation by up to 40% .

Jeremy Yung Shern Low and Chip-Hong Chang [25] have proposed an elegant algorithm to overcome the daunting problem of performing programmable power-of-two scaling operation directly in the popular moduli set $\{2^n\text{-}1, 2^n, 2^n\text{+}1\}$. The synthesis results are very promising. Exceptional computation speed improvement of 52.2%, 52.8%, 53.1%and 53.2 % for *n*=5, 6, 7and 8 respectively, with an average area saving of 14.1%, over the hybrid design have been demonstrated. Besides, remarkable saving of total power by 44.6%, 51.5%, 37.8% and leakage power by 22.4%, 20.8%, 24.4% and 14.6% for *n*=5.6,7 and 8 respectively, have been achieved. In this

programmable power-of-two scalar can efficiently scale an RNS number up to one third of the dynamic range.

T. Tay et al. [26] have presented algorithm for signed RNS for $\{2^n - 1, 2^n, 2^n + 1\}$ moduli set. A sign detection circuit is added for negative number scaling. In order to correct the wrong scaled negative integer in RNS form, substantial hardware overheads have been suffered to detect the range of the residues upon magnitude scaling. In brief, an exact, fast and area efficient 2^n signed integer RNS scaler for the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ is proposed for complex DSP architectures.

L. Sousa [27] has proposed a new category of RNS scaler for four moduli set to in. In this work, both CRT and MRC technique is used. It performed RNS scaling in two stages. In its first stage, it uses CRT to find out intermediate scaling result for special three moduli set [21]. In second stage, the intermediate results of [21] and fourth modulus are then processed by mixed radix conversion (MRC) technique to find out final scaling results. The experimental results are then compared with hybrid RNS scaler. In hybrid RNS scaler, firstly RNS reverse converter is used to find out equivalent binary number, after that RNS scaling is performed in binary number system and then finally RNS forward converter is used to find out final RNS scaled output. The proposed RNS scaler is flexible, cost effective and also suitable for energy constrained equipment, specially mobile systems.

Karthik K. M et al. [29] have proposed a RNS to binary reverse converter. Due to its small size and parallel arithmetic operations, Residue Number System (RNS) based data is well suitable for implementing various digital signal processing functions that are commonly used in many consumer electronic devices nowadays. This paper presents the design of the RNS to Binary reverse converter which is typically the bottleneck in an RNS based signal processing system. Specifically, it describes a pipelined parallel prefix based modular adders which is used to implement the reverse converter for the $\{2^k\text{-}1, 2^k, 2^k\text{+}1\}$ moduli set based RNS system.

Piotr Patronik and Stanisław J. Piestrak [31] have presented a new residue number system (RNS) reverse converter for four balanced moduli $\{2^n + 1, 2^n - 1, 2^n, 2^{n-1} + 1\}$ (n odd). It offers large dynamic ranges with reduction in the width of the channel mod $2^{n-1} + 1$. From the set of basic properties, three versions of a

reverse (residue-to-binary) converter with different performance characteristics can be designed for this new RNS. The experimental results obtained in this paper suggest that power consumptions, delay and area of this new converter is significantly improved compared to the state-of-the-art converters for the 4-moduli set $\{2^n + 1, 2^n - 1, 2^n, 2^{n+1} + 1\}$ (n odd). Finally, the other major benefit of this new moduli set is that, it should be extended by including other moduli to increase dynamic range, reverse converters for these new multi-moduli systems could be made by reusing the reverse converters proposed here on the premises of the MRC.

Chapter-3

RESIDUE NUMBER SYSTEM

A residue number system (RNS) is a representation of number in which large integer number can be represented using a set of smaller integers, so that computation may be performed more efficiently. The RNS uses positional bases (called moduli) that are relatively prime to each other, e.g. 2, 3, 5. To convert a conventional weighted number (X) to residue system, simply take the residue of X with respect to each of positional moduli. Example of RNS number shown below.

To convert the decimal number 208 to {15, 16, 17} RNS

$$R_5 = 208 mod 15 = 13$$

$$R_3 = 208 mod 16 = 0$$

$$R_2 = 208 mod 17 = 4$$

The decimal number 208 is represented by [13,0,4] in above mentioned RNS.

The main advantage of RNS is the absence of carries between moduli in addition and multiplication. Arithmetic is completely done within each residue position, i.e there is no inter-moduli carry propagation in RNS. 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 [12].

3.1 Arithmetic Operation in RNS

3.1.1 Addition and Subtraction

Let two number X and Y having residue x and y respectively with respect to same modulus m. That is, if

$$X \equiv x \ (mod\ m) \tag{1}$$

$$Y \equiv y \ (mod\ m) \tag{2}$$

Then,

$$X \pm Y \equiv x \pm y \ (mod\ m) \tag{3}$$

3.1.2 Multiplication

Let two decimal numbers, represented by X and Y having residue x and y respectively with respect to same modulus m. That is, if

$$X \equiv x \ (mod\ m)$$

$$Y \equiv y \ (mod\ m)$$

Then,

$$X \times Y \equiv x \times y \ (mod\ m) \tag{4}$$

The properties above mentioned for addition and multiplication have two direct extensions. If $\{x_1,x_2,\ldots\ldots,x_n\}$ and $\{y_1,y_2,\ldots\ldots,y_n\}$ are respectively, the residue sets (without any restrictions) of X and Y and are obtained relative to the moduli $m_1,m_2,\ldots\ldots,m_n$, then the residue set of $X \pm Y$ is

$$\{|x_1 \pm y_1|_{m_1}, |x_2 \pm y_2|_{m_2}, \ldots\ldots |x_n \pm y_n|_{m_n}\} \tag{5}$$

And that for $X \times Y$ is

$$\{|x_1 \times y_1|_{m_1}, |x_2 \times y_2|_{m_2}, \ldots\ldots |x_n \times y_n|_{m_n}\} \tag{6}$$

3.1.3 Additive Inverse

The additive inverse, $\bar{x}$, of a residue, x, is defines by the equation

$$x + \bar{x} = 0 \tag{7}$$

The additive inverse operation can be applied to individual residues, or to a system as a whole, but its main role is in subtraction. The additive inverse does exist for a number and unique for any residues. It is similar to 2's compliment in binary number system representation. It is obtained through a simple operation:

$$\bar{x} = |\ m - x\ |_m \tag{8}$$

3.1.4 Multiplicative Inverse

The multiplicative inverse is an analogue of reciprocal in conventional arithmetic and is defined as follows. If x is a non-zero integer then x^{-1} is the multiplicative inverse of x, with respect to the modulus m if

$$|x \times x^{-1}|_m = 1 \tag{9}$$

where x and m have no common factor.

3.1.5 Division

In residue number systems, there is difficulty in even the seemingly simple matter of defining precisely what division means. It might appear possible to define division in a residue number system by proceeding as it has done above for addition and multiplication that is, define the operation for residues and then extend that to tuples [7] of residues, but two complications immediately arise if this attempt occur. The first is that residue division and normal division are in concord only when the quotient resulting from the division is an integer. And the second follows from the fact that zero does not have a multiplicative inverse for zero.

Conventional division may be represented by the equation

$$\frac{x}{y} = q \tag{10}$$

which implies that

$$x = y \times q \tag{11}$$

In residue number systems, however, this last equation does not necessarily hold, since in these systems the fundamental relationship is congruence and not plain equality. For RNS equivalent of preceding equation for residues x and y, the congruence is

$$y \times q \equiv x \ (mod\ m) \tag{12}$$

Multiply both sides by the multiplicative inverse of y^{-1}, we get

$$q \equiv x \times y^{-1} \ (mod\ m) \tag{13}$$

Unlike the corresponding situation in conventional arithmetic, in RNS multiplication by a multiplicative inverse is not always equivalent to division.

3.2 Chinese Remainder Theorem (CRT)

Given a set of relatively prime moduli $(m_1, m_2, \ldots\ldots m_n)$ than any number $X < M$, the set of residues $\{X \ mod\ m_i \mid 1 \leq i \leq n\}$ is unique, where

$$M = \prod_{i=1}^{n} m_i \tag{14}$$

$$x_i = |X|_{m_i} \tag{15}$$

Therefore the integer representation of X, is given by

$$X = \left| \sum_{i=1}^{N} M_i \left| M_i^{-1} \right|_{m_i} x_i \right|_M \tag{16}$$

where $M_i = M/m_i$ and $\left|M_i^{-1}\right|_{m_i}$ is multiplicative inverse of $|M_i|_{m_i}$

3.3 Mixed Radix Conversion (MRC)

Based on MRC, X can be calculated from its residue by

$$X = g_1 + m_1 \times \left(g_2 + m_2 \times \left(g_3 + m_3 \times (\ldots\ldots)\right)\right) \tag{17}$$

where

$$g_1 = x_1$$

$$g_2 = \left|(x_2 - g_1) \times |m_1^{-1}|_{m_2}\right|_{m_2}$$

$$g_3 = \left|((x_3 - g_1) \times |m_1^{-1}|_{m_3} - g_2) \times |m_2^{-1}|_{m_3}\right|_{m_2}$$

3.4 Special Moduli-Sets

The moduli sets with 2^n are special moduli set $\{2^n\text{-}1, 2^n, 2^n\text{+}1\}$ is one of them. The special moduli are usually referred to as low cost moduli, since forward conversion and reverse conversion from their residue can be realized relatively easily and does not require complex operation such as multiplicative inverses, multiplication [1]-[2], [3]-[7].

3.4.1 Modulus 2^n

This modulo can be implemented directly by n-LSB bits of X with no additional computation.

3.4.2 Modulus 2^n-1

This modulo is quite easy to implement. By using RNS properties, this can be written

$$|2^m|_{2^n-1} = |2^x|_{2^n-1} \tag{18}$$

where $m = nq + x$ and x is remainder [7].

3.4.3. Modulus 2^n+1

Similarly

$$|2^m|_{2^n+1} = 2^x \quad \text{if } q \text{ is even}$$
$$= 2^n + 2 - 2^x \quad \text{otherwise} \tag{19}$$

Residues are obtained by nominally dividing X by m_i . The residue x_2 is easier to compute because it is just right shift operation. In order to determine the residue x_1 and x_3, X is broken into three n bit blocks B_1, B_2, B_3.

$$X = x_{3n-1}\, x_{3n-2} \ldots\ldots\, x_{2n}\, x_{2n-1} \ldots\ldots\, x_n\, x_{n-1} \ldots\ldots\ldots\ldots x_0 \tag{20}$$

$$B_1 = \sum_{j=2n}^{3n-1} x^j\, 2^{j-2n} \tag{21}$$

$$B_2 = \sum_{j=n}^{2n-1} x^j\, 2^{j-n} \tag{22}$$

$$B_3 = \sum_{j=0}^{n-1} x^j\, 2^j \tag{23}$$

Therefore by expanding our results and using RNS properties [13] will give:

$$x_1 = |B_1 - B_2 + B_3|_{2^n+1} \tag{24}$$

$$x_3 = |B_1 + B_2 + B_3|_{2^n+1} \tag{25}$$

The forward conversion and reverse conversions are shown in Fig. 1 and Fig. 2 respectively. The reverse conversion is based on the CRT in which carry save adder is used.

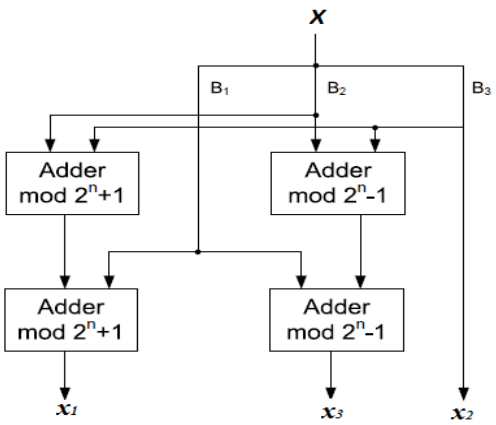

Fig. 1.1: $\{2^n-1, 2^n, 2^n+1\}$ Forward Conversion [13]

3.5 Scaling

Division in the RNS is difficult. Nevertheless if the divisor is one of the moduli or a product of several moduli, but not a multiple of a power of a modulus, then the division is much easier and is known as scaling. In conventional binary number system scaling is performed by shifting the dividend to the right. Although scaling in RNS is not as simple as shifting. Scaling is often used to prevent overflow, since it reduces the dynamic range of RNS and thus keeps them within the permissible bounds.

Scaling in RNS of a number X by a positive integer Y can be expressed as[2]:

$$\left\lfloor \frac{X}{Y} \right\rfloor_{mi} = \left| \frac{X - x}{Y} \right|_{mi} \tag{26}$$

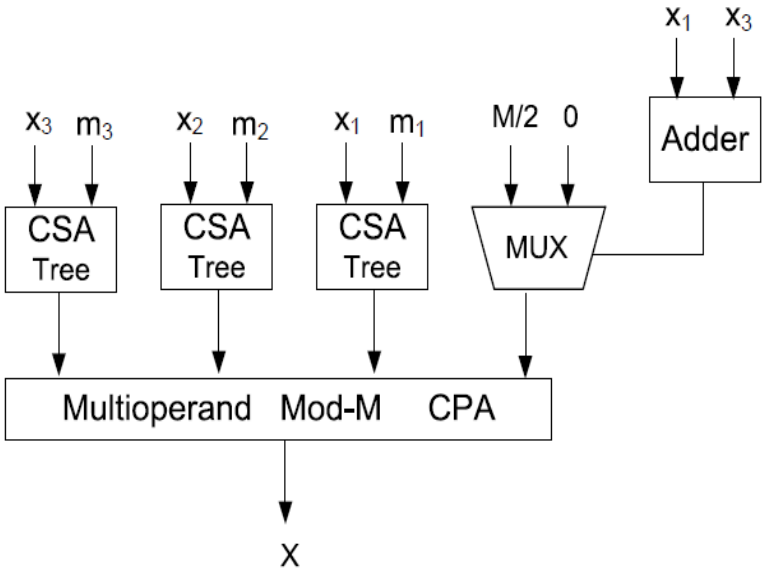

Fig. 1.2: Reverse Conversion [13]

3.6 Properties of RNS

Modular arithmetic has so many impressive properties over traditional arithmetic operation which makes it special. Some of them discussed below.

Property 1 [11]: Multiplying *n*-bit number X by i power of two in modulo 2^n-1 then X circularly shift left by i bits denoted by

$$\left|2^i X\right|_{2^n-1} = CLS_n(X, i) \tag{27}$$

Property 2[24]: Since $|-X|_{2^n-1}=|2^n - 1 - X|_{2^n-1} = \bar{X}$, where $\bar{X}$ denotes one's compliment of X, then

$$\left|-2^{i}X\right|_{2^{n}-1} = CLS_{n}(\bar{X}, i) \tag{28}$$

Property 3: If $X_{n-1:0}$ denote the *n*-LSB of *X* , then

$$|X|_{2^{n}} = X_{n-1:0} \tag{29}$$

Property 4 [16]: Let *x* be a single bit binary variable, then

$$\left|2^{n+i}x\right|_{2^{n}+1} = \left|2^{i}\bar{x} + 2^{n+i}\right|_{2^{n}+1} \tag{30}$$

Property 5: If *k* is an integer, then

$$|kx|_{km} = k|x|_{m} \tag{31}$$

Property 6: $$|2^{n}|_{2^{n}+1} = -1 \tag{32}$$

In the upcoming chapters, it has sown that, how these properties become useful for driving expressions and hardware design for RNS scaler.

Chapter-4

RNS SCALER DESIGN

In this chapter, techniques for variable scaling of RNS are discussed. Previously, RNS scalers were designed using LUT or ROM based design, in which delay and area required is very large [1]. But in recent studies, RNS scalers are designed using fully combinational circuit for power two based RNS moduli sets. These types of RNS scalers don't require any memory element. Section 4.1 describes the equations for variable power-of-two RNS scaler for special three moduli set $\{2^n\text{-}1, 2^n, 2^n\text{+}1\}$.In section 4.2, variable power-of-two RNS scaler is designed for 4-moduli set. In section 3, hardware description is given for these two RNS scalers.

4.1 Variable RNS Scaler for $\{2^n\text{-}1, 2^n, 2^n\text{+}1\}$

Let X be an integer and (x_1, x_2, x_3) is RNS representation of X with respect to moduli set $\{m_1, m_2, m_3\}$, where, $m_1 = 2^n - 1,\ m_2 = 2^n\ and\ m_3 = 2^n + 1$.

Before deriving scaling equation for this moduli set, it require some multiplicative inverse equation and these equations are

$$|M_1^{-1}|_{m_1} = 2^{n-1}$$

$$|M_2^{-1}|_{m_2} = 2^n - 1$$

$$|M_3^{-1}|_{m_3} = 2^{n-1} + 1$$

The proof of these multiplicative inverses is given in [21].

The fixed precision part of an integer in PFP can be expressed as

$$\left\lfloor \frac{X}{2^n} \right\rfloor = \frac{X - |X|_{2^n}}{2^n} \tag{33}$$

From (33), X can be written as follows:

$$X = \left\lfloor \frac{X}{2^n} \right\rfloor 2^n + |X|_{2^n} \tag{34}$$

The last term of equation (34) corresponds to variable precision part of *X* and equal to the residue x_2 of the moduli set RNS $\{2^n\text{-}1, 2^n, 2^n\text{+}1\}$. Therefore

$$X = \left\lfloor \frac{X}{2^n} \right\rfloor 2^n + x_2 \tag{35}$$

Without sacrificing the guaranteed minimum precision, the PFP representation of can be scaled by an arbitrary power-of-two factor 2^r to prevent overflow. This can be done by dividing both sides of (35) by 2^r followed by the least integer function, which gives

$$\left\lfloor \frac{X}{2^r} \right\rfloor = \left\lfloor \frac{\left\lfloor \frac{X}{2^n} \right\rfloor 2^n + x_2}{2^r} \right\rfloor \tag{36}$$

Note that, this expression is the scaled output of *X* and represented by *Y*. Therefore

$$Y = \left\lfloor \frac{X}{2^r} \right\rfloor \tag{37}$$

Let $\{y_1, y_2, y_3\}$ are residues of *Y*. Then,

$$y_i = \left| \left\lfloor \frac{X}{2^n} \right\rfloor 2^{n-r} + \left\lfloor \frac{x_2}{2^r} \right\rfloor \right|_{m_i} \tag{38}$$

4.1.1. Equation for y_1

y_1 is the representation of $|Y|_{m_1}$ in RNS w.r.t. modulo m_1. Therefore

$$y_1 = \left| \left| \left\lfloor \frac{X}{2^n} \right\rfloor \right|_{m_1} |2^{n-r}|_{m_1} + \left| \left\lfloor \frac{x_2}{2^r} \right\rfloor \right|_{m_1} \right|_{m_1} \tag{39}$$

By substituting $||\lfloor X/2^n \rfloor||_{m_1} = |x_1 - x_2|_{m_1}$ from (11) of [21] into (39), it becomes

$$y_1 = \left| \left| (2^{n-r}) x_1 \right|_{m_1} + \left| -(2^{n-r}) x_2 \right|_{m_2} + \left\| \left\lfloor \frac{x_2}{2^r} \right\rfloor \right|_{m_1} \right|_{m_1} \tag{40}$$

The last two terms of (40) can be as follows:

$$\left| (2^{n-r})(-x_2) + \left\lfloor \frac{x_2}{2^r} \right\rfloor \right|_{2^n-1} = \left| (2^{n-r})(-x_2) + \frac{x_2 - |x_2|_{2^r}}{2^r} \right|_{2^n-1} \tag{41}$$

It can be easily shown that the multiplicative inverse of 2^r modulo 2^n-1 is 2^{n-r}. Therefore

$$|(2^{n-r})(-x_2) + (2^{n-r})(x_2 - |x_2|_{2^r})|_{2^n-1} = |-|x_2|_{2^r}.2^{n-r}|_{2^n-1} \tag{42}$$

$$y_1 = |(2^{n-r})(x_1) - |x_2|_{2^r}.2^{n-r}|_{2^n-1} \tag{43}$$

which is further represented as:

$$y_1 = |P_1 + P_2|_{m_1} \tag{44}$$

where P_1,P_2 can be described as:

$$P_1 = \left|x_1.2^{n-r}\right|_{2^n-1} = CLS_n(x_1, n-r) = CRS_n(x_1 : r) \tag{45}$$

$$P_2 = \left|-(x_2.)_{r-1:0}.2^{n-r}\right|_{2^n-1} \tag{46}$$

$$P_2 = \overline{(x_2)_{r-1:0}} \| \underbrace{1....1}_{n-r} \tag{47}$$

Where || denotes the concatenation of two binary strings, *a* and *b* and CRS is circular right shift and CLS is circular left shift operation.

4.1.2 Equation for y_2

The scaled output y_2 can be calculated by using

$$y_2 = \left|\frac{\left\lfloor\frac{X}{2^n}\right\rfloor 2^n + x_2}{2^r}\right|_{m_2} \tag{48}$$

From first part of (48) using CRT, this can be written as

$$\left|\left\lfloor\frac{X}{2^n}\right\rfloor\right|_{m_2} = \left|\left|m_3|M_1^{-1}|_{m_1}x_1 + \frac{m_1m_3}{m_2}|M_2^{-1}|_{m_2}x_2 + m_2|M_3^{-1}|_{m_3}x_3\right|_{m_1m_3}\right|_{m_2} \tag{49}$$

$$\left|\left\lfloor\frac{X}{2^n}\right\rfloor\right|_{m_2} = ||(2^{n-1} + 2^{2n-1})x_1 - 2^n x_2 + (2^{n-1} + 2^{2n} - 1 - 2^{2n-1})x_3|_{2^{2n}-1}|_{m_2} \tag{50}$$

Each term present in (50) can be written as follows:

$$L_1 = |2^{2n-1}x_1|_{2^{2n}-1} = CLS_{2n}(x_1, 2n-1)$$

$$= \left(x_{1,0} \underbrace{00 \ldots .000}_{n} \quad x_{1,n-1} \ldots . x_{1,1} \right)_2 \tag{51}$$

$$L_2 = |2^{n-1} x_1|_{2^{2n}-1} = CLS_{2n}(x_1, n-1)$$

$$= \left(0 x_{1,n-1} \ldots . . x_{1,0} \underbrace{00 \ldots .0}_{n-1} \right)_2 \tag{52}$$

$$L_3 = |-2^n x_2|_{2^{2n}-1} = CLS_{2n}(\overline{x_2}, n)$$

$$= \left(\bar{x}_{2,n-1} \bar{x}_{2,n-2} \ldots . . \bar{x}_{2,0} \underbrace{11 \ldots 1}_{n} \right)_2 \tag{53}$$

$$L_4 = |2^{2n-1} x_3|_{2^{2n}-1} = CLS_{2n}(x_3, 2n-1)$$

$$= \left(x_{3,0} \underbrace{00 \ldots .000}_{n} \; x_{3,n} \ldots . x_{3,1} \right)_2 \tag{54}$$

$$L_5 = |2^{n-1} x_3|_{2^{2n}-1} = CLS_{2n}(x_3, n-1)$$

$$= \left(x_{3,n} x_{3,n-1} \ldots . . x_{3,0} \underbrace{00 \ldots .0}_{n-1} \right)_2 \tag{55}$$

$$L_6 = |-x_3|_{2^{2n}-1} = \left(\underbrace{11 \ldots .1}_{n-1} \; \bar{x}_{3,n} \ldots . \bar{x}_{3,0} \right)_2 \tag{56}$$

The sum of binary vector L_1 and L_2 can be merged and written in 2n- bit binary vector N_1. The remaining four terms can further reduced to three binary vector by reordering the bits of different addends at the same position by logic minimization given in [21]. The final 2n bit binary vector is given below.

$$N_1 = \left(x_{1,0}, x_{1,n-1}, x_{1,n-2,}, \ldots . . x_{1,0}, x_{1,n-1}, x_{1,n-2,}, \ldots . . x_{1,1} \right)_2 \tag{57}$$

$$N_2 = \left(\bar{x}_{2,n-1}, \bar{x}_{2,n-2} \ldots \ldots \ldots \bar{x}_{2,0}, \bar{x}_{3,n-1} \bar{x}_{3,n-2} \ldots \ldots . . \bar{x}_{3,0} \right)_2 \tag{58}$$

$$N_3 = \left(x_{3,0}, x_{3,n-1}, x_{3,n-2,}, \ldots . . x_{3,0}, x_{3,n-1}, x_{3,n-2,}, \ldots . . x_{3,1} \right)_2 \tag{59}$$

$$N_4 = \left(\bar{x}_{3,n} \underbrace{11 \ldots \ldots 1}_{n} \underbrace{\bar{x}_{3,n} \bar{x}_{3,n} \bar{x}_{3,n} \ldots \ldots \bar{x}_{3,n}}_{n-1} \right)_2 \tag{60}$$

Since $x_3 \leq 2^n, x_{3,n} = 1$ if $x_3 = 2^n$. Also when $x_3 = 2^n, x_{3,i} = 0 \; \forall i \leq n-1$. the n lower bits of N_2 become "1"s, and N_3 becomes an all-zero string. N_1, N_2, N_4 can be summed using 2n-bit CSA w EAC. On the other hand, if $x_3 < 2^n$ then, $x_{3,n} = 0$. In this

case, N_4 becomes all-ones string and its modulus with 2^{2n}-1 become equal to zero. Hence no additional 2*n*- bit vector is required to sum all the ones of N_4.

Finally to calculate the y_2, the *n*-bits of x_2 are concatenated at the LSB of the output of the 2*n*-bit CSA w EAC. It will form a *3n* bit vector.. Divisor by 2^r is simply a *r* bit shifting of this *3n* bit binary vector, which is similar like shifting of P_1. The last *n*-LSB of the output of 3*n*- bit shifter will give us y_2.

4.1.3 Equation for y_3

The *n* bit modulo y3 is expressed as follows:

$$y_3 = \left\| \left\lfloor \frac{X}{2^r} \right\rfloor \right\|_{m3} \tag{61}$$

$$y_3 = \left\lfloor \frac{\left\lfloor \frac{X}{2^n} \right\rfloor 2^n + x_2}{2^r} \right\rfloor_{m_3} \tag{62}$$

Substitute $||\lfloor X/2^n \rfloor||_{m_3} = |x_2 + 2^n x_3|_{m_3}$ from (11) of [21] in (62) and use *Property 6.* y_3 can be modified as:

$$y_3 = \left| \left| x_2 - x_3 \right|_{m_3} \cdot \left| 2^{n-r} \right|_{m_3} + \left\| \left\lfloor \frac{x_2}{2^r} \right\rfloor \right\|_{m_3} \right|_{m_3} \tag{63}$$

It can be easily found that the multiplicative inverse of 2^r modulo 2^n+1 is -2^{n-r} . Hence, first and last term of (63) can be combined and given as:

$$\left| (2^{n-r})x_2 + \left\lfloor \frac{x_2}{2^r} \right\rfloor \right|_{2^n+1} = |2^{n-r}|x_2|_{2^r}|_{2^n+1} \tag{64}$$

By using *property 4*, it can be further simplified to

$$\left| (2^{n-r})x_2 + \left\lfloor \frac{x_2}{2^r} \right\rfloor \right|_{2^n+1} = |2^{n-r}.(x_2)_{r-1:0}|_{2^n+1} \tag{65}$$

The expression in (65) is an *r* bit binary number shifted to the left by *n-r* bits.

The second term present in (63) can be written as :

$$\begin{aligned} |-(2^{n-r})x_3|_{2^n+1} &= |(2^{2n-r})x_3|_{2^n+1} \\ &= \left|(x_3)_{n-r} + (x_3)_{r-1:0}.2^{n+(n-r)}\right|_{2^n+1} \end{aligned} \tag{66}$$

By decomposing the second term present in (63) and applying *Property 4* to each binary bit variable give:

$$|-(2^{n-r})x_3|_{2^n+1} = \left|(x_3)_{n-r} + \overline{(x_3)}_{r-1:0}.2^{(n-r)} + \sum_{i=1}^{r} 2^{2n-1}\right|_{2^n+1} \tag{67}$$

The last term of (67) can be modified to

$$\left|\sum_{i=1}^{r} 2^{2n-1}\right|_{2^n+1} = |2^{2n} - 2^{2n-r}|_{2^n+1} = |1 + 2^{n-r}|_{2^n+1} \tag{68}$$

By substituting equation (65) and (67) into(63), y_3 can be written as

$$y_3 = \left|(x_3)_{n-r} + \overline{(x_3)}_{r-1:0}.2^{(n-r)} + 2^{n-r}.(x_2)_{r-1:0} + (1 + 2^{n-r})\right|_{2^n+1} \tag{69}$$

When $x_3 < 2^n, (x_3)_n =' 0' and\ (x_3)_i \in \{0,1\} \forall i < n$, and equation (69) reduced to.

$$y_3 = \left|(x_3)_{n-r} + \overline{(x_3)}_{r-1:0}.2^{(n-r)} + 2^{n-r}.(x_2)_{r-1:0} + 2^{n-r})\right|_{2^n+1} \tag{70}$$

The sum of first two terms of equation (70) give us $\left(\overline{(x_3)_{r-1}} \| (x_3)_{n-1:r}\right)$.

When $x_3 = 2^n, (x_3)_n =' 1' and\ \ (x_3)_i =' 0'\ \forall i < n$, then

$$y_3 = \underbrace{11\ldots.1}_{n} \tag{71}$$

Finally, y_3 is divided into three parts *i.e.* Q_1, Q_2, Q_3.The expression for Q_1, Q_2, Q_3 based on [26] are as follows:

$$Q_1 = \overline{(x_3)_n}\ \&\ \left(\overline{(x_3)_{r-1}} \| (x_3)_{n-1:r}\right)$$

$$= \overline{(x_3)_n}\ \&\ CCRS_n\left((x_3)_{n-1:0} : r\right) \tag{72}$$

$$Q_2 = (x_2)_{r-1:0} \| \underbrace{1.....1}_{n-r} \tag{73}$$

$$Q_3 = \overline{(x_3)_n} \tag{74}$$

where '*&*' denotes logical *AND* of the binary variables. $CCRS_n(X, r)$ denotes the complementary circular right shift of *r*-bit binary variable. By feeding Q_1, Q_2 and Q_3 into a CSA with CEAC, it can be easily shown that:

$$y_3 = \left|S + C_{n-2:0} \| \overline{C_{n-1}}\right|_{2^n+1} \tag{75}$$

where *S* and *C* denotes the *n*-bit sum and carry vectors of the CSA with CEAC.

4.2 Variable RNS Scaler for Extended Four-Moduli Sets

In section 4.1, scaling equations for moduli set $\{2^n\text{-}1, 2^n, 2^n\text{+}1\}$ has been derived. In this section, these equations has been explored for extended four moduli-set $\{2^n\text{-}1, 2^n, 2^n\text{+}1, m_4\}$ where m_4 is any power-of-two modulus. The MRC is applied in this level. The equation for y_1 and y_3 will remain same for this moduli set. So, only equations for y_2 and y_4 has been derived. The equation (50) can be re-written as follows:

$$\left\lfloor \frac{X_{3\leftrightarrow 1}}{2^n} \right\rfloor = |(2^{n-1} + 2^{2n-1})x_1 - 2^n x_2 + (2^{n-1} + 2^{2n} - 1 - 2^{2n-1})x_3|_{2^{2n}-1} \quad (76)$$

where $X_{3\leftrightarrow 1}$ might be consider as the intermediate result. To find out $X_{3\leftrightarrow 1}$ from (76), concatenate x_2 at the LSB of the (76). This can be represented as

$$X_{3\leftrightarrow 1} = \left\lfloor \frac{X_{3\leftrightarrow 1}}{2^n} \right\rfloor . 2^n + x_2 \quad (77)$$

From the residues $X_{3\leftrightarrow 1}$ and x_4, it reduced our four- moduli set to two moduli set, which is $\{(2^{2n} - 1) \times 2^n, m_4\}$. By applying MRC to this moduli set, *X* can be written as:

$$X = X_{3\leftrightarrow 1} + \left|(x_4 - X_{3\leftrightarrow 1}) \times |M_4^{-1}|_{m_4}\right|_{m4} \times M_4 \quad (78)$$

where $M_4 = \frac{M}{m_4} = (2^{2n} - 1) \times 2^n$. To obtain scaling equation, divide (78) by 2^r followed by least integer function as shown in (79).

$$\left\lfloor \frac{X}{2^r} \right\rfloor = \left\lfloor \frac{X_{3\leftrightarrow 1}}{2^r} \right\rfloor + \left|(x_4 - X_{3\leftrightarrow 1}) \times |M_4^{-1}|_{m_4}\right|_{m4} \times \frac{2^n}{2^r} \times (2^n - 1) \times (2^n + 1) \quad (79)$$

By using (79), the equations for y_2, y_4 can be computed as follow.

$$y_2 = \left|\left\lfloor \frac{X_{3\leftrightarrow 1}}{2^r} \right\rfloor\right|_{2^n} + \left|(x_4 - X_{3\leftrightarrow 1}) \times |M_4^{-1}|_{m_4}\right|_{m4} \times 2^{n-r} \times (-1)\Big|_{2^n} \quad (80)$$

$$y_4 = \left|\left\lfloor \frac{X_{3\leftrightarrow 1}}{2^r} \right\rfloor + \left|(x_4 - X_{3\leftrightarrow 1}) \times |M_4^{-1}|_{m_4}\right|_{m4} \times 2^{n-r} \times (2^{2n} - 1)\right|_{m_4} \quad (81)$$

4.2.1 Case study $\{2^n - 1, 2^n, 2^n + 1, 2^{n-1} + 1\}$

The four moduli set$\{2^n-1, 2^n, 2^n+1, 2^{n-1}+1\}$ has been proposed in [31] for relatively prime numbers for odd values of *n* and provide dynamic range $(3n + log_2 n)$. The multiplicative inverse $|M_4^{-1}|_{2^{n-1}+1}$ is provided in [31] for $n \geq 5$.

$$|M_4^{-1}|_{2^{n-1}+1} = m_4 - \left(\sum_{i=0}^{\frac{n-3}{2}-1} 2^{2i+1} + 1 \right) \tag{82}$$

The proof of correctness of multiplicative inverse is derived in [31]. In this case, $|2^{2n}-1|_{m_4}$ is always equal to 3 for any odd value of *n*.

As an example, consider *n*=7, $i.e., \{m_1, m_2, m_3, m_4\} = \{127,128,129,65\}$. The residue representation $\{x_1, x_2, x_3, x_4\}$ of an integer *X*=425629 is {52,29,58,9}. Scaling *X* by 2,4,8,32 produces 212814,106407,53203,13300 respectively. Table 1 shows the scaled value of $\{y_1, y_2, y_3, y_4\}$.

Table 4.1

Scaling of (52, 29, 58, 9) by 2, 4, 8 and 32 in {127, 128, 129, 65} RNS.		
r	*i*	y_i
1	1	89
	2	78
	3	93
	4	4
2	1	108
	2	39
	3	111
	4	2
3	1	117
	2	83
	3	55
	4	33
5	1	92
	2	116
	3	13
	4	40

4.3 Hardware Implementation

4.3.1 Hardware implementation of variable scaler for $\{2^n\text{-}1, 2^n, 2^n\text{+}1\}$ Moduli Set.

The complete block diagram for RNS scaler for $\{2^n\text{-}1, 2^n, 2^n\text{+}1\}$ is shown in fig. 4.1.

4.3.1.1 Generation of P_1

The scaled residue can be generated by P_1 and P_2. The P_1 can be generated CRS of x_1 by r bits. This can be implemented using multiplexers as shown in fig 4.2. x_1 is the input to the CRS and r decides maximum number of shifts.

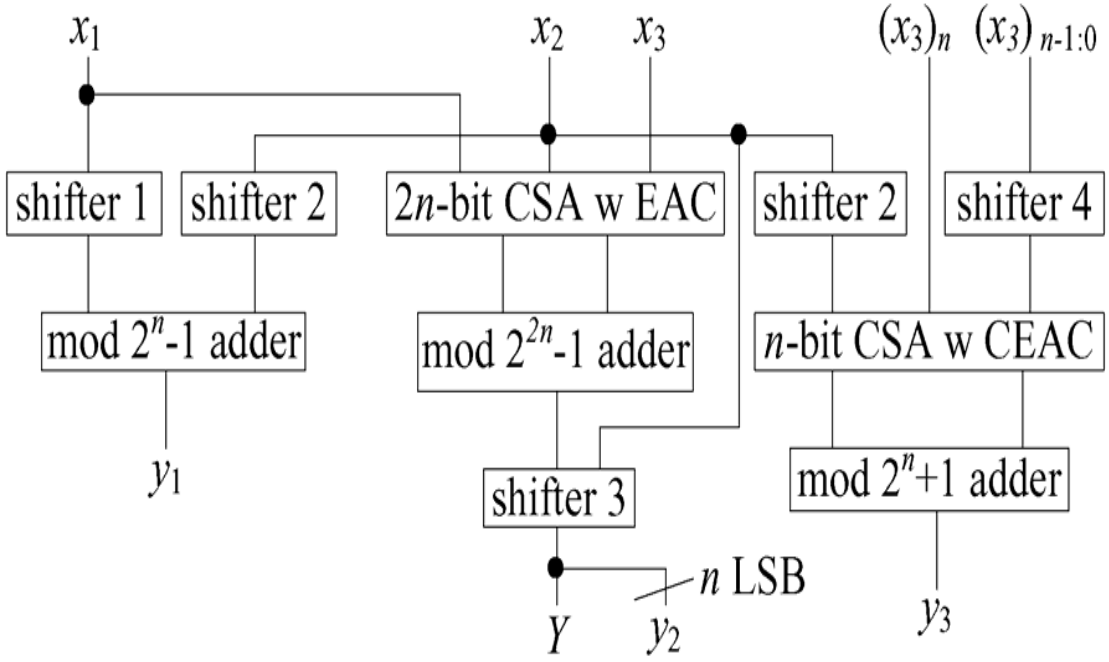

Fig. 4.1: Architecture of RNS Scaler for 3 moduli set

Shifter 1: $\lceil \log_2 n \rceil$-stage logarithmic cyclic right shifter of an n-bit word by r bits.

Shifter 2: $\lceil \log_2 (n+1) \rceil$-stage logarithmic left shifter of an n-bit word by r bits.

Shifter 3: $\lceil \log_2 (n+1) \rceil$-stage logarithmic logical right shifter of $3n$-bit word by r bits.

Shifter 4: $\lceil \log_2 (n+1) \rceil$-stage logarithmic complementary cyclic right shifter of an n-bit word by r bits.

4.3.1.2 Generation of P_2

P_2 can be generated by shifting the one's complement of x_2 to the left shift by $s=n-r$ bits and padded with "1". This can be implemented by a n -bit logarithmic left shifter (Shifter 2 of block diagram Fig. 4.1). The logarithmic shifter has $\lceil \log_2(n+1) \rceil$ stages. The implementation is based on multiplexer as shown in fig. 4.3.

Addition of P_1 and P_2 in *mod* 2^n-1 adder will provide y_1 as shown in fig. 4.1.

4.3.1.3 Generation of y_2

The scaled residue y_2 is obtained from the r-th to the $(n+r-1)$-th bit of the concatenation of $\lfloor X / 2^n \rfloor$ and x_2. A modulo 2^{2n} -1 adder with $2n$-bit CSA with EAC and a -bit shifter, as shown in Fig. 4.2., are used to implement y_2. As the right shift amount is programmable from 0 to n , Shifter 3 is a $3n$ -bit wide logical right shifter, where the r vacant positions are filled in with zeros.

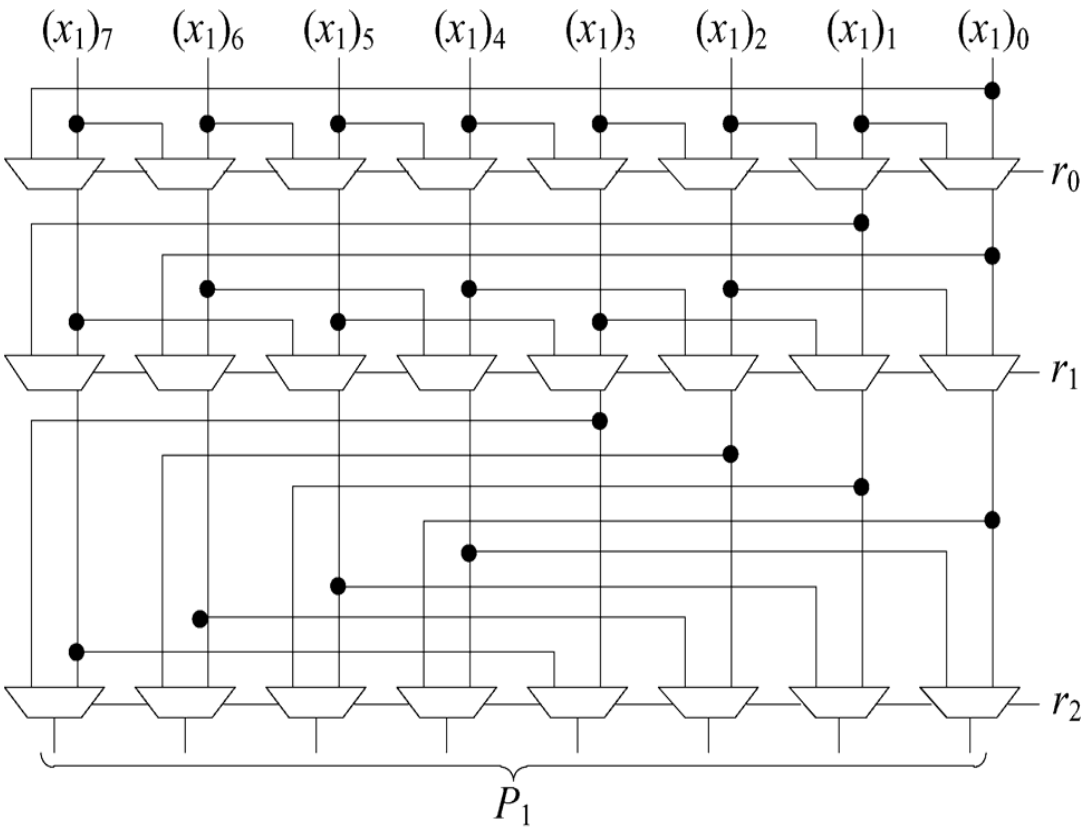

Fig. 4.2 8-bit 3 Stage logarithmic CRS for generation of P_1 [26]

4.3.1.4 Generation of y_3

The scaled residue y_3 is produced by a CSA with CEAC followed by a mod 2^n+1 adder, with the input vectors of Q_1, Q_2, Q_3. The complementary circular shift of x_3 by r-bits to the right can be implemented by an n-bit $\lceil \log_2(n+1) \rceil$-stage logarithmic complementary cyclic right shifter (Shifter 4 in Fig. 1.3), as shown in Fig. 3.3 for n=8 . The output is gated by $\overline{(x_3)_n}$ with n two-input *AND* gates to produce Q_1.

Q_2 is similar to x_2 of channel except that the input are not inverted. Hence it can be implemented by the same type of n -bit logarithmic left shifter shown in fig. 4.3 without the input inversion.

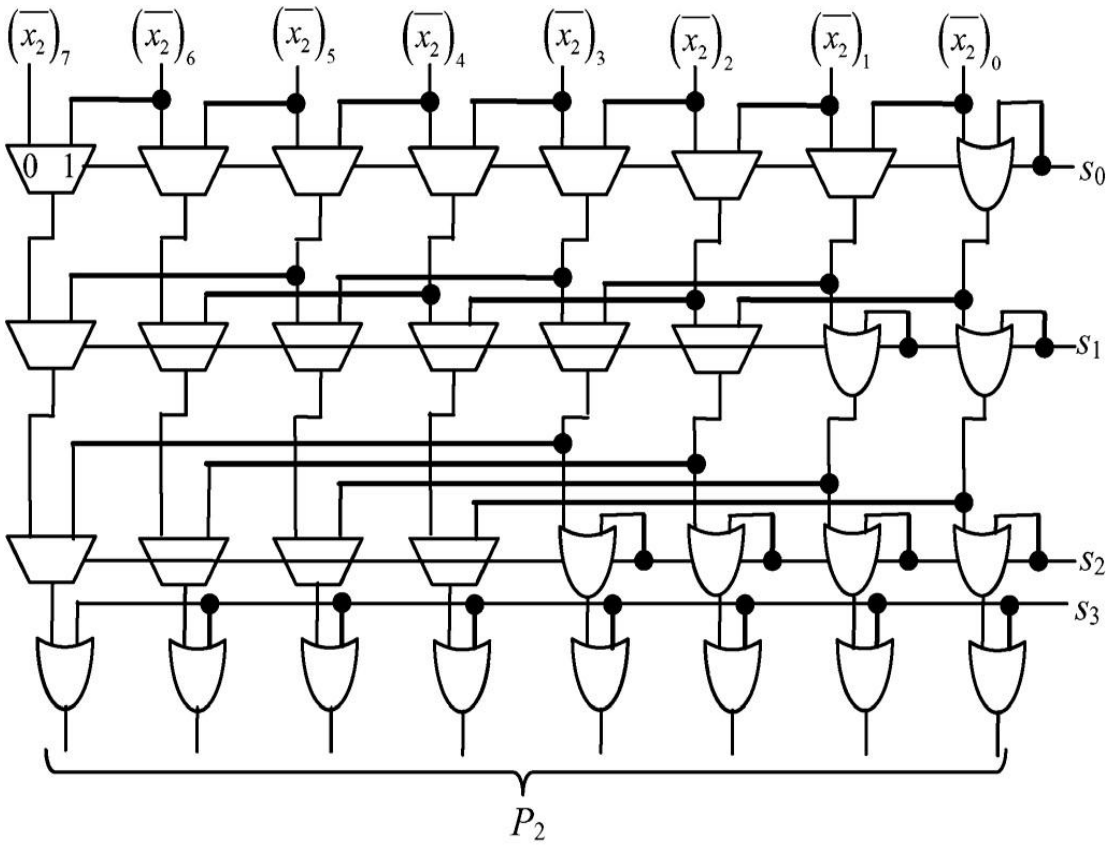

Fig.4.3. 8 bit 4-stage logarithmic left shifter for the generation of P_2 [26]

Fig. 4.4 shows the CSA with CEAC for the addition of Q_1, Q_2 and Q_3. It consists of one full adder (FA) and *(n-1)* half adder (HA) to produce the n-bit sum and carry outputs. These two outputs are then summed using modulo 2^n+1adder to produce the scaled residue output y_3.

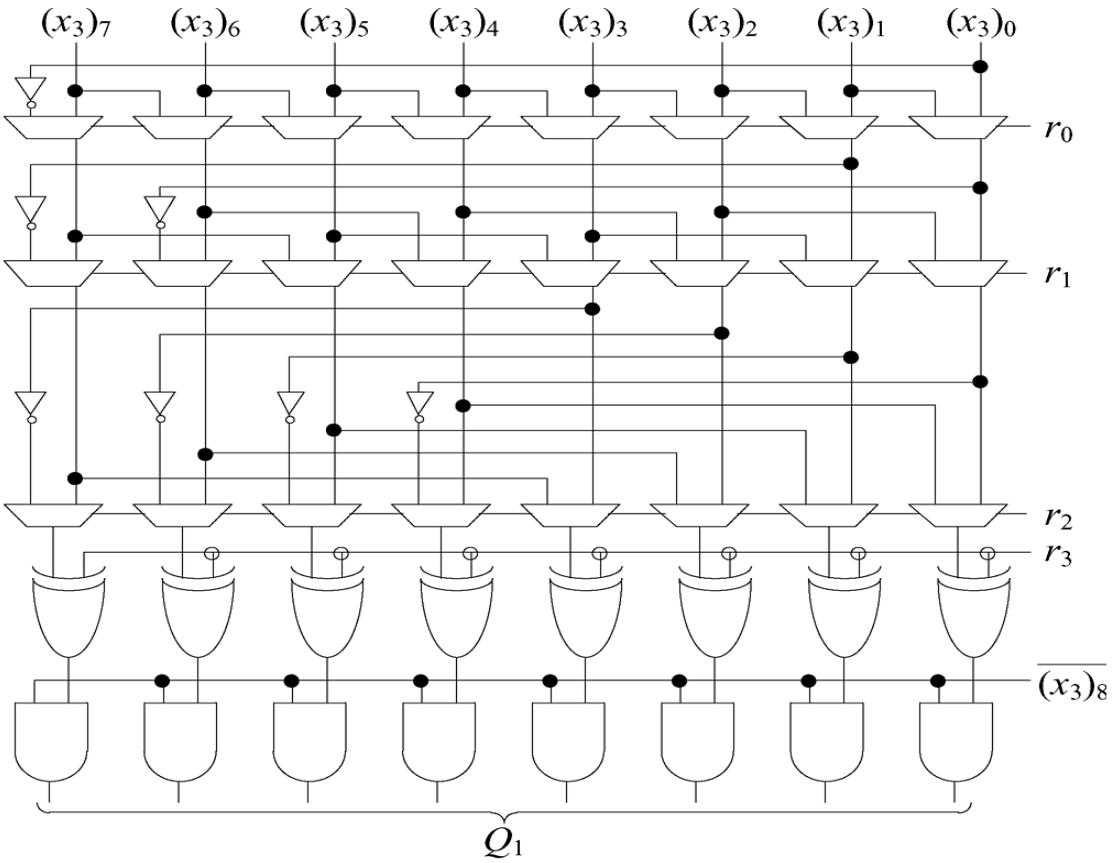

Fig.4.4 8 bit 4-stage logarithmic CCRS for generation of Q_1[1]

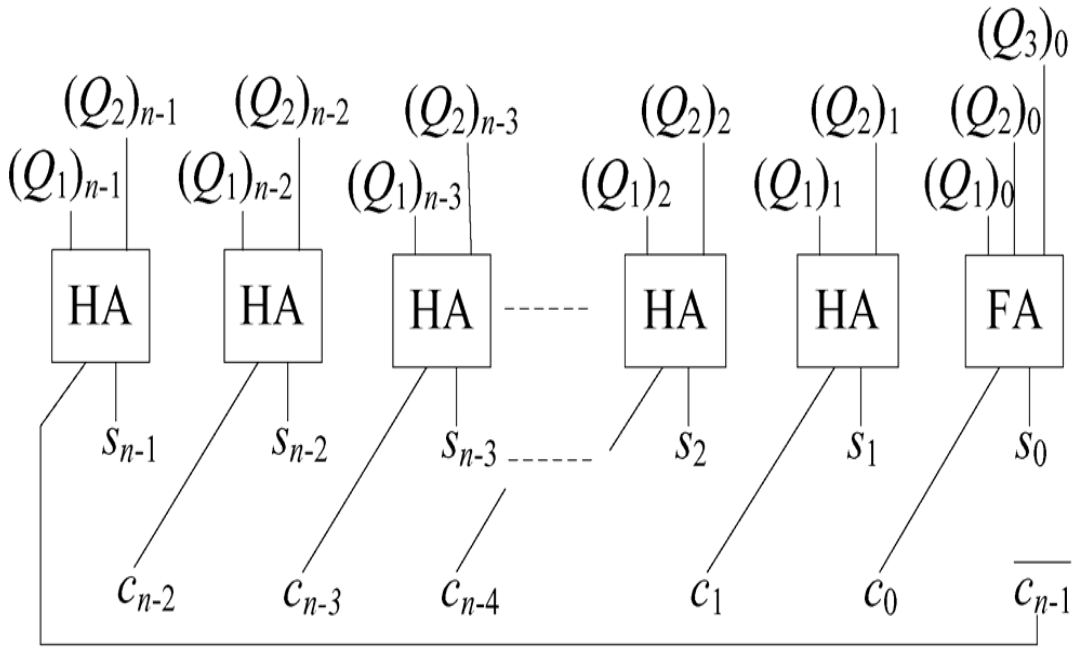

Fig.4.5 *n*-bit CSA with CEAC of modulus 2^n+1 channel

4.3.2 Hardware implementation of variable scaler for {2^n-1, 2^n, 2^n+1, m_4} Moduli Set.

Figure 4.6 shows the block diagram of variable RNS scaler for extended 4-moduli set. Here, only scaler blocks for y_2 and y_4 are shown. $y_1$and y_3 can be implemented in similar way as shown in section 4.3.1. (80) and (81) both require $T = \left|(x_4 - X_{3\leftrightarrow 1}) \times |M_4^{-1}|_{m_4}\right|_{m4}$, which is performed using highlighted component in fig. 4.6. Moreover on the left hand side of fig. 4.6, a shifter and a n bit CPA are used to produce the final result in channel 2^n (y_2). But on the right hand side of fig. 4.6 a multiplier and a CPA, both modulo m_4 are used to produce y_4. Although these general architectures can be tuned to perform optimization of specific moduli-sets , i.e., for particular value of m_4. This work is primarily focused on general architectures that can be used as a framework to develop comprehensive scalers.

The description of all logic blocks present in block diagram shown in fig. 4.6 is given below. Description is given for $m_4 = 2^{n-1}+1$.

- Shifter 1: $\lceil \log_2 n \rceil$-stage logarithmic logical right shifter of an 3n-bit word by r bits.
- Shifter 2: $\lceil \log_2 n \rceil$-stage logarithmic logical right shifter of an $\lceil n/2 \rceil$-bit word by r bits.
- Shifter 3: $\lceil \log_2 n \rceil$-stage logarithmic logical right shifter of an n-bit word by r bits.
- $-T$: Give the two's complement of T.
- $|-X_{3\leftrightarrow 1}|_{m_4}$: One's complement.
- Modulo m_4 multiplier is described in [23].
- Modulo m_4 carry propagate adder (CPA) is described in [30].
- Modulo m_2 carry propagate adder (CPA) is simple binary adder in which carry bit is neglected.
- $\left|\left\lfloor \frac{X_{3\leftrightarrow 1}}{2^r} \right\rfloor\right|_{m_4}$ can be generated by modulo m_4 converter.

Rest of block are described in previous section generation of y_2.

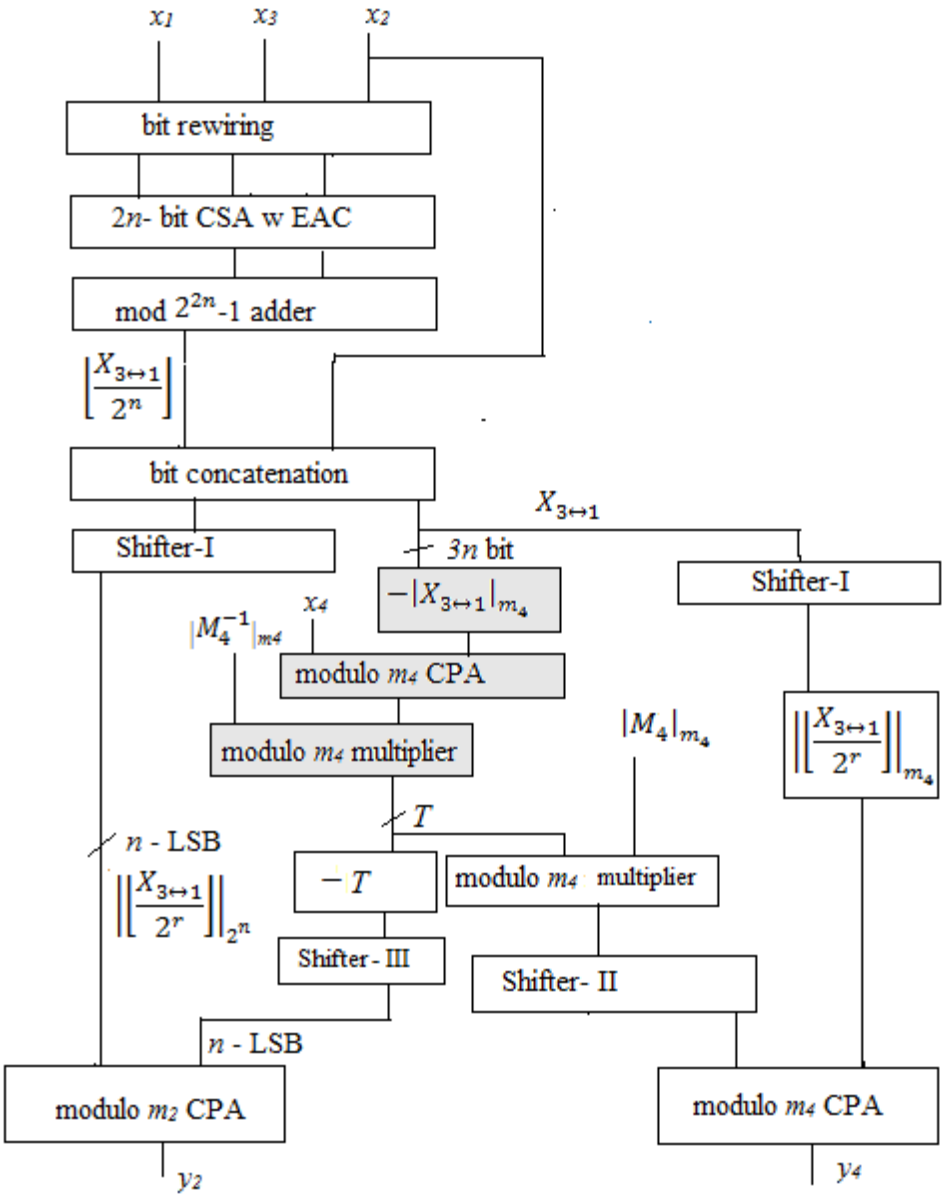

Fig.4.6. Generation of y_2 and y_4.

Chapter-5

FIELD PROGRAMMABLE GATE ARRAY

In this chapter, FPGA concepts and FPGA synthesis flow is discussed. An FPGA is a semiconductor device, in which large numbers of transistors are connected to perform logic functions. It can perform simple addition to complex digital logics such as digital filtering, error detection and correction.

5.1 Introduction to FPGA

A field programmable gate array (FPGA) is a device that can be configured by the designer after manufacturing. These devices are truly revolutionary devices that blend the benefits of both hardware and software. FPGAs are programmed using source code which is described in hardware description language (HDL) to specify how the chip will work. It can implement any logic function that Application Specific Integrated Circuit could perform. It contains logic blocks and a hierarchy of reconfigurable interconnects that allow the logic blocks to be wired together. The logic blocks present in FPGA can perform any complex combinational functions. In most FPGAs, the logic blocks also include memory elements that can be simple flip flops or complete blocks of memory.

FPGAs blend the benefits of both software and hardware. The implementation of circuits just like as hardware performing huge area, power and speed over software. Yet, these can be reprogrammed easily and cheaply to implement wide range of task. It performs tasks like computer. Such systems can perform task hundred times faster than the microprocessor based designs. The major advantage of FPGAs over ASICs is that, these can be reprogrammed many times. FPGAs are being incorporated as central processing elements in many applications such as consumer electronics, automotive, image/video processing military/aerospace, base stations, networking/communications, super computing and wireless applications.

5.2 FPGA Technology Trends

- General trend is bigger and faster.

- This is being achieved by increases in device density through even smaller fabrication process technology.
- New generations of FPGAs are geared towards implementing entire systems on a single device.
- Features such as RAM, dedicated arithmetic hardware, clock management and transceivers are available in addition to the main programmable logic.
- FPGAs are also available with the embedded processors (embedded in silicon or as cores within the programmable logic fabric).

5.3FPGA Implementation

The FPGA that is used for the implementation of the circuit is the Spartan 6E (Family), xc6slx45 (Device). The working environment/tool for the design is the Xilinx ISE 14.5i is used for FPGA Design flow of Verilog code

5.3.1 Overview of FPGA Design Flow

As increase in the complexity of design logic leads to increase in complexity of FPGA architecture. So today, most of FPGA vendors provide a fairly complete set of design tools that allow automatic synthesis and compilation of design code in hardware description languages, such as VHDL and Verilog HDL. The steps for FPGA design flow shown in fig 5.1. Design constraints, specification of design and specification of the target device are the input to design flow. Design constraints typically include the expected operating frequencies of different clocks, the delay bounds of the signal path delays from input pads to output pads (I/O delay), from the input pads to registers (setup time), and from registers to output pads (clock-to-output delay). In some cases, delays between some specific pairs of registers may be constrained.

Each FPGA vendor typically provides a large range of FPGA devices, which have different operating speed, cost and power trade-offs. Designer can start with small devices. But if, the synthesis results fails, the designer have to upgrade to higher capacity devices.

5.3.2 Design Entry

This is the first step of FPGA Design Flow. With the help of hardware description languages, the basic architecture of the system is designed. Sources files are created

for representing the given design. The source file can be EDIF file, embedded processor file, schematic file or HDL file.

5.3.3 Behavioral Simulation

After system design phase, the functionality of the design is checked by performing behavioral simulation. Simulation can be performed on HDL test benches, HDL source files, HDL test benches. Simulation can be performed either using ModelSim simulator or iSIM simulator. This step is performed after synthesizing the design. The simulation after place and route is called timing simulation.

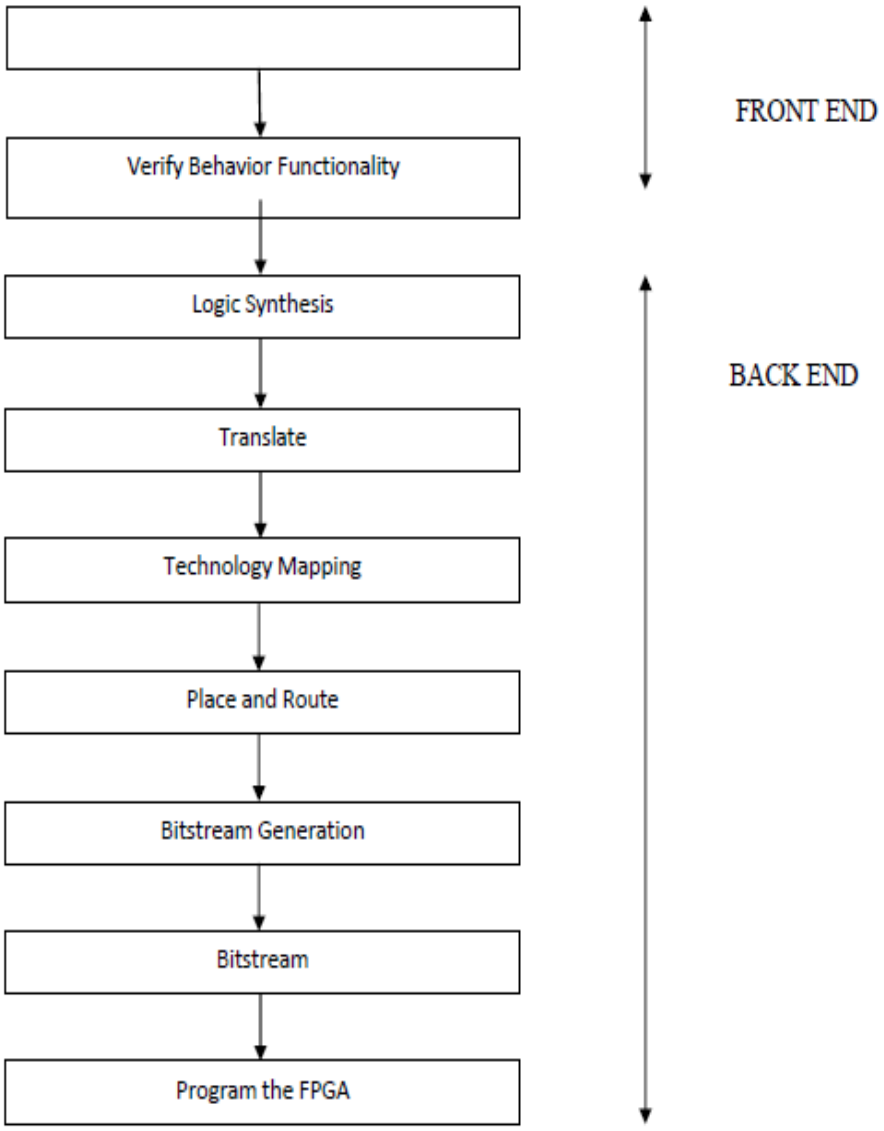

Fig 5.1 FPGA Design Flow

5.3.4. Design Synthesis

After successfully compilation of behavioural simulation, the design is then synthesized. Xilinx uses XST (Xilinx synthesis technology) synthesizer for synthesizing Verilog HDL and VHDL codes. It generates net-list file for given design. This file contains logic design data and constraints. The net-list file has an extension .NGC and serves as input for the translate process. This file is the result of three steps i.e. HDL parsing, HDL synthesis and low level optimization. In HDL parsing, code is checked for syntax errors. In HDL synthesis part code is analysed and inferred to basic macros like RAM, multiplexers, adders etc. This is done for efficient implementation of technology. In HDL synthesis part FSM recognition is also done. Synthesizer chooses one of the several FSM encoding algorithms according to the optimization goal for efficient implementation. In low level optimization the macros are transferred into technology specific components such as carry logic, shift registers, clock buffers etc. This step also contains timing optimization, technology mapping and register replication. So the NGC file is generated after HDL parsing, HDL synthesis and low level optimization. Fig. 5.2 shows the complete steps.

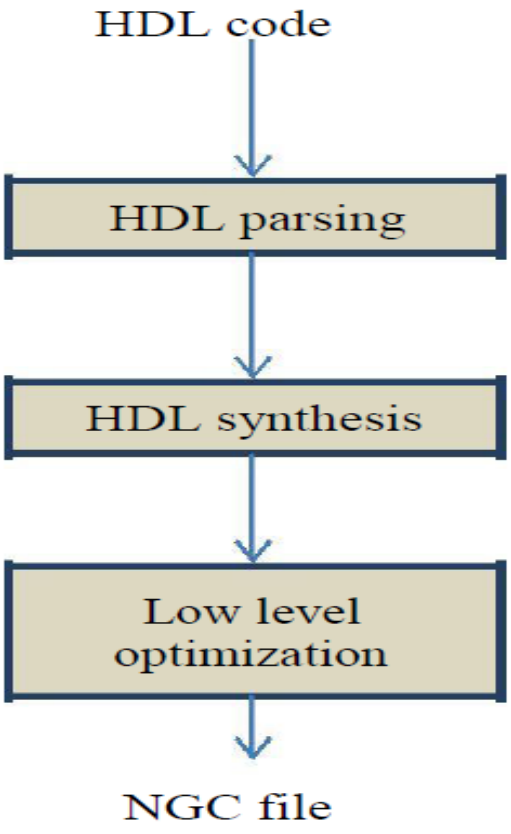

Figure 5.2 Steps in synthesis process

Beside this XST also generates RTL schematic, technology schematic and synthesis report.

RTL Schematic: Register transfer level (RTL) is a representation of pre optimized design. Basic building blocks, adders, AND gates etc. are used in this representation.

- Technology schematic: It is the representation of NGC file in terms of LUTs, I/O buffers.
- Synthesis Report: It gives the entire synthesis result like area and times estimations. It also give us device utilization summary, partition resource summery etc.

5.3.5 Implementing the design

In this process, the design is transferred onto desired device. Implementation is done in three steps. These are translate, map and route. The complete process flow is shown in fig.5.4.

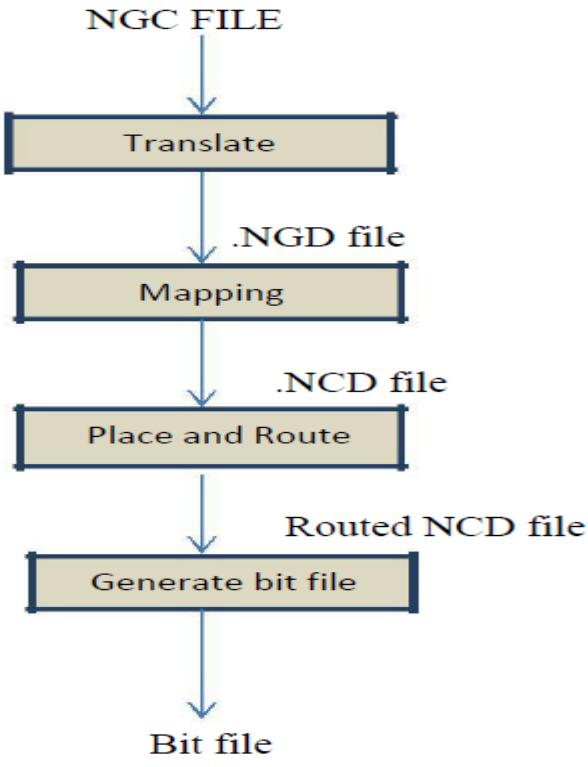

Figure 5.3 Different files generated in implementation process

- Translate: All the net-lists and design constraints are combined and saved in a file called native generic database file. Timing requirement of design are also specified in this process.

- Mapping: Circuit is divided into sub-blocks in this process. These sub-blocks are mad in such a way that they can fit into FPGA sub-blocks. A native circuit description file is generated in this process.
- Place and Route: This process takes input as NCD file. The sub-blocks of map process are converted into logic blocks.

5.3.6 Generating Programming File

In this process, bit file is generated from NCD file for particular hardware device. The generated bit file is used to program hardware. Sometimes this process is also known as bit stream generation.

5.3.7 Analysing design using ChipScope

The complexity of FPGA designs getting increased day by day. Also, debugging and verification takes almost 50% of design time. But the chip scope pro software perform faster debugging, almost shrink design by 25%. ChipScope pro uses FPGA resources like block RAM for trigger and data storage, slice logic or trigger comparison. It uses three types of flows i.e. core generator, core inserter and plan-ahead low. The core inserter flow is similar to plan-ahead flow and provided in plan-ahead software. In core generator flow the core is instantiated in source HDL, while in core inserter flow the core is inserted into generated file after synthesis. Different types of cores are used by ChipScope software like ICON core, ILA core, VIO core, IBA core. Some of them are explained below:

- VIO core: It generates virtual inputs and outputs. It is used to apply stimulus and read outputs transition on the node we want to select. It has options for synchronous and asynchronous inputs and outputs, where each can have width of 256 bits. There is also option of clock. It can be system clock or JTAG clock The outputs of the implemented design are connected to the input of the VIO core and inputs of the design are connected to the output of the VIO core therefore the inputs are virtual LEDs and outputs are virtual DIP Switches. This core is controlled by the ICON core. VIO core uses FPGA logic not RAM. It is only used in core generator flow.
- ICON core: It is used to control up to 15 capture cores. It acts as interface between JTAG interface and capture cores. The main function of this core is to

control the other cores. It can be used in both the core generator and core inserter flows.

- ILA core: It is a capture core. It can be used to create custom triggers when activated causes data to be stored during circuit operations. Signals can be stored depending on the condition specified by used. A design can contain up to 15 ILA cores.

Chapter-6

IMPLEMENTATION OF RNS SCALER

This chapter discuss the implementation results of RNS scalers. The implementation is done on Xilinx ISE 14.5 software.

The working environment of design is:

- Tool version :ISE 14.5
- Design Strategy : Balanced
- Family : Spartan 6
- Device : XC6SLX45
- Speed : -3
- Package : CSG324
- Simulator : iSim
- Total Slices : 27288

6.1. Design of RNS Scaler

Here, 2^r variable RNS scalers are designed for three and four moduli sets. The design has been coded in Verilog HDL and simulated and synthesized using Xilinx ISE design suite 14.5. Design algorithms present in [21] and [31] are also coded in same environment. The algorithms, [21] and [31] are also for three and four moduli sets respectively but, these are exact scaling algorithms. They have not functionality of variable scaling. The algorithms present in section 4.1 and 4.2 are designed for variable scaling in RNS and finally results are compared with each other.

6.2 Results and Discussions

6.2.1 Simulation Results

Fig. 6.1 shows the simulation results for algorithm present in [21]. Consider following inputs in {127,128,129} RNS representation.

x_1=104 is residue of 429872 *w.r.t* 127.

x_2=48 is residue of 429872 *w.r.t* 128.

x_3=44 is residue of 429872 *w.r.t* 129.

The output obtained after scaling with m_2 are (y_1, y_2, y_3) = (56, 30, 4), which represents 3358 in binary number system.

In fig.6.2 simulation results for variable RNS scaler present in section 4.1, is shown for the above given number. Here scaling is done by choosing r=3, which means number is scaled by 8. The outputs are (y_1, y_2, y_3) = (13, 102, 70), which represents 53734 in binary number system.

Figure 6.3 shows the simulation results for algorithm present in [31], which is for extended four moduli set. Considering the following inputs in {127, 128, 129, 65} RNS representation.

x_1 =104 is residue of 429872 *w.r.t* 127.

x_2 =48 is residue of 429872 *w.r.t* 128.

x_3 =44 is residue of 429872 *w.r.t* 129.

x_4 =27 is residue of 429872 *w.r.t* 65.

The output obtained after scaling with m_2 are (y_1, y_2, y_3, y_4) = (56, 30, 4, 43), which represents 3358 in binary number system.

In fig.6.4 simulation results for variable RNS scaler present in section 4.2, is shown for the above given number in four moduli sets. Here scaling is done by choosing r=3, which means number is scaled by 8. The outputs are (y_1, y_2, y_3, y_4) = (13, 102, 70, 44), which represents 53734 in binary number system.

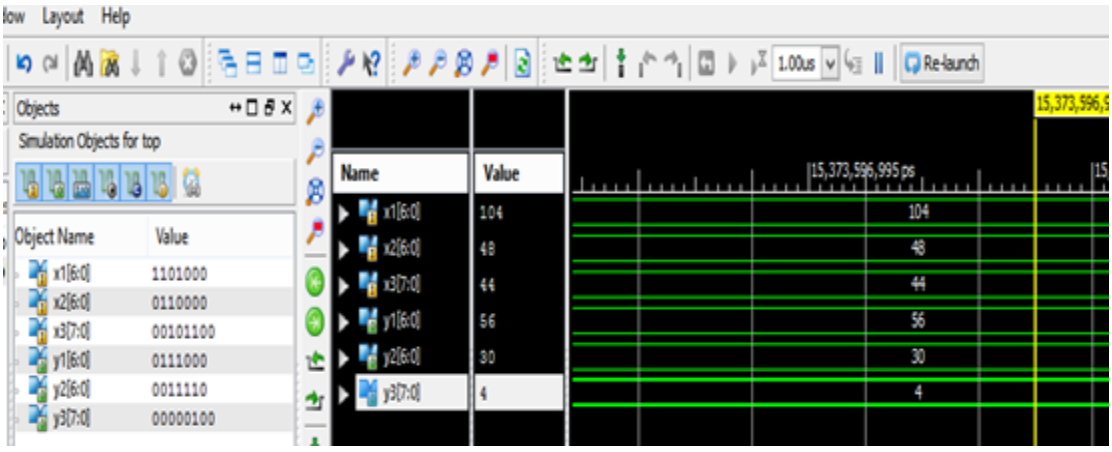

Fig. 6.1 Simulation result for exact RNS scaler for three moduli set.

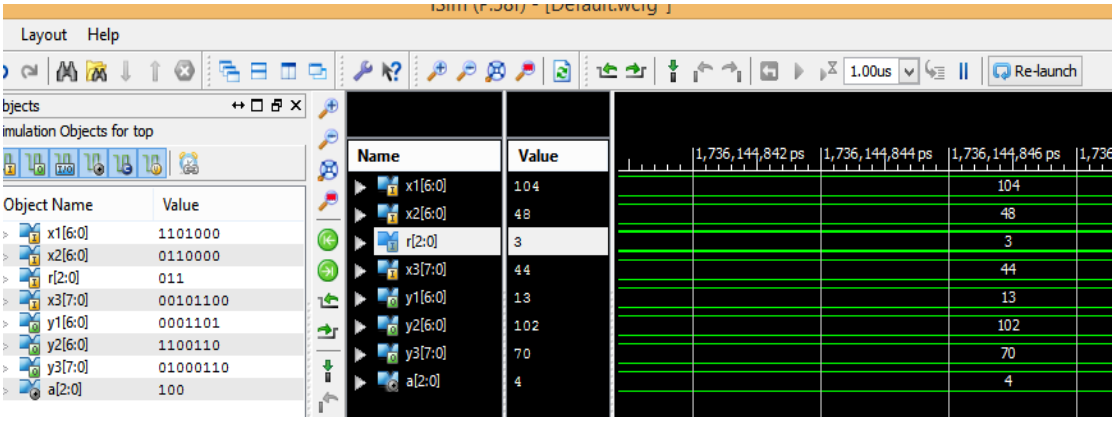

Fig. 6.2 Simulation result for variable RNS scaler for three moduli set.

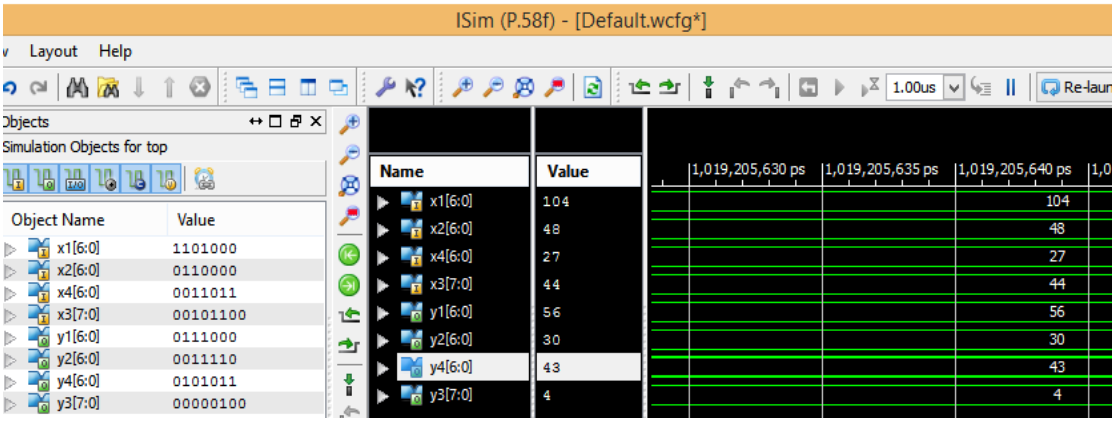

Fig. 6.3 Simulation result for exact RNS scaler for extended four moduli set.

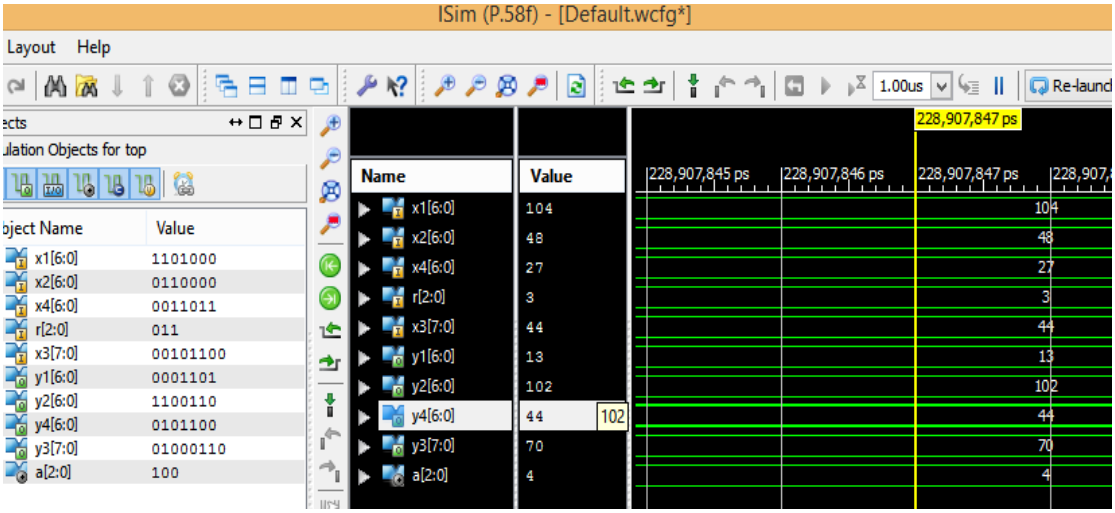

Fig. 6.4 Simulation result for variable RNS scaler for extended moduli set.

6.2.3 Synthesis Results

The synthesis results for various RNS scalers are shown in tabular form. In table 6.1, critical path delay is shown for exact and variable RNS scaler. Percentage increase in

delay when we moves from exact to variable scaler is also shown in last column. Similarly, in table 6.2 the results for area in terms of LUTs is compared for three moduli sets. Table 6.3 and 6.4 describes the results for delay and area respectively for extended four moduli sets.

Table 6.1. Delay of exact and variable 3-modili set RNS scalers.

Delay (*ns*)				
n	Moduli set	Exact	Variable	% increase
5	{31, 32, 33}	13.487	16.444	21.92
6	{63, 64, 65}	15.811	18.028	14.02
7	{127, 128, 129}	16.073	19.714	22.65
8	{255, 256, 257}	20.225	23.348	17.07

Table 6.2. Area of exact and variable 3-modili set RNS scalers.

Area (LUTs)				
n	Moduli set	Exact	Variable	% increase
5	{31, 32, 33}	48	94	95.83
6	{63, 64, 65}	62	116	87.10
7	{127, 128, 129}	78	147	88.46
8	{255, 256, 257}	94	175	86.17

Table 6.3. Delay of exact and variable 4-modili set RNS scalers.

Delay (*ns*)				
n	Moduli set	Exact	Variable	% increase
5	{31,32,33,17}	17.444	19.225	10.20
7	{127,128,129,65}	21.989	23.352	6.19
9	{511,512,513,257}	25.516	26.938	5.57
11	{2047,2048,2049,1025}	31.744	33.481	5.47

Table 6.4. Area of exact and variable 4-modili set RNS scalers.

Area (LUTs)				
n	Moduli set	Exact	Variable	% increase
5	{31,32,33,17}	582	1036	78.01
7	{127,128,129,65}	678	1125	65.92
9	{511,512,513,257}	1553	2553	64.39
11	{2047,2048,2049,1025}	2024	3548	75.29

6.2.2 Implementation Through chipScope pro analyser

The design is verified by chip scope pro tool of Xilinx. Chip scope pro analyzer is the in-built verification tool provided in the ISE pack of Xilinx that is used to verify the implemented multiplier. Since number of inputs for variable RNS scaler for four moduli set and *n*=7 is 30 and outputs are 27. So, direct implementation using on-board inputs outputs is very difficult on FPGA. Therefore chip scope pro analyzer is used. The code that is designed in Verilog HDL with necessary cores is instantiated in top module. For virtual input output operation and for controlling the generated VIO (virtual input/output) core, two IP cores are generated. VIO core is generated for providing virtual inputs and outputs. The outputs of the design are connected to the input of the VIO core and inputs of the design are connected to the output of the VIO core. This core is also provided with clock input to synchronize the operation. This same clock is also connected to the design block also. ICON core is generated for controlling the VIO core. The main function of the ICON core is to control the other cores. After generating the VIO and ICON core a top module is designed which instantiates both these cores and the design module. Then synthesis of top module is done. The RTL schematic for the top module is shown in figure 6.5. The figure shows that the connections are according to our desire. After that the top module is implemented on FPGA. The generated bit file is burned on the FPGA. The results can be verified by using chip scope analyzer window. The output will be generated on the port sync_in when inputs are given on async_out port. Figure 6.6 shows result on ChipScope pro for variable RNS scaler for extended four moduli set.

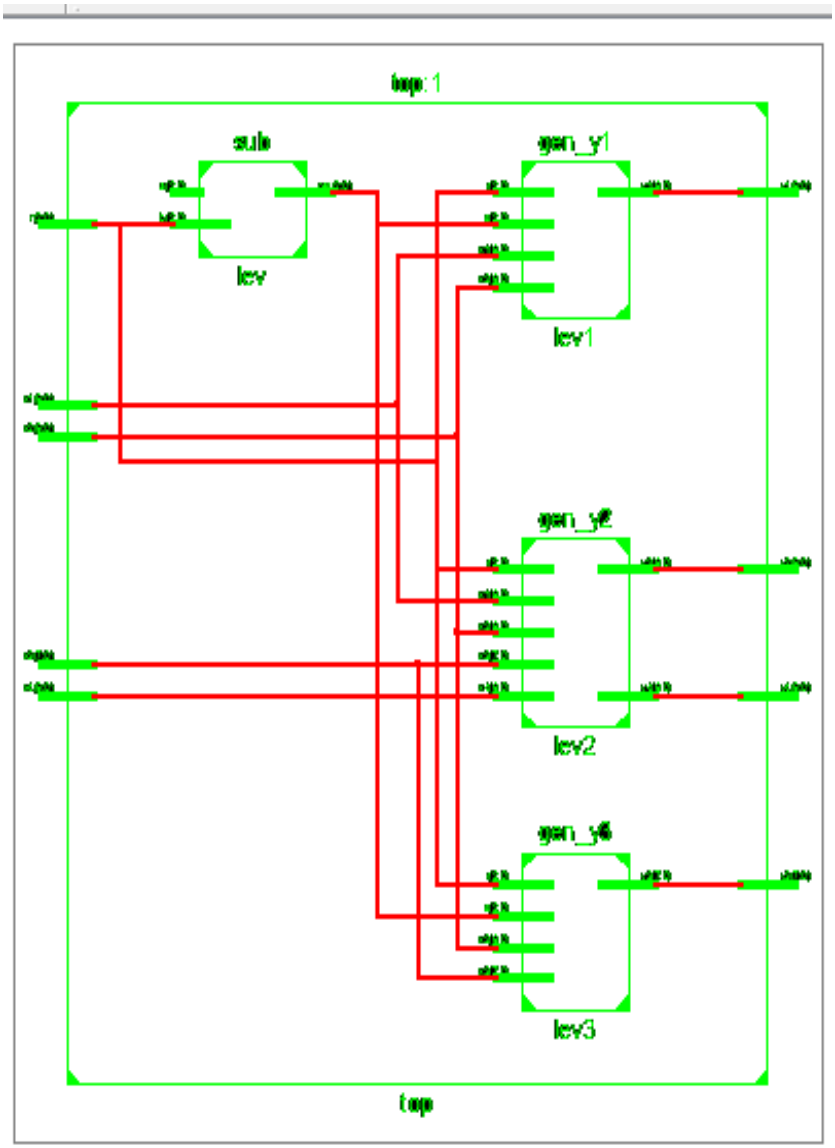

Fig. 6.5 RTL schematic for variable RNS scaler for extended moduli set.

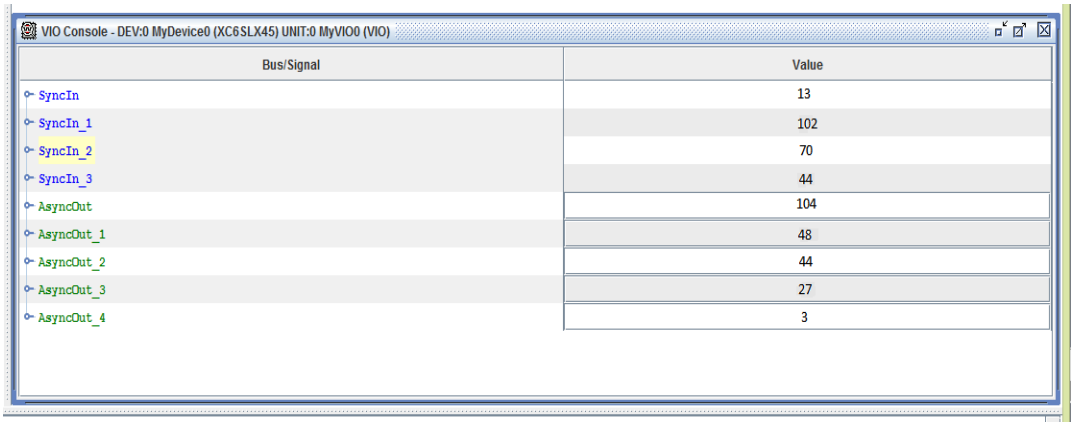

Fig. 6.6 Output results on ChipScope pro analyaer.

Chapter-7

CONCLUSION AND FUTURE SCOPE

7.1 Conclusion

In this thesis, 2^r variable RNS scalers are implemented for $\{2^n\text{-}1, 2^n, 2^n\text{+}1\}$ and $\{2^n + 1, 2^n - 1, 2^n, 2^{n-1} + 1\}$ (n odd) moduli sets. These moduli sets have higher dynamic ranges. The new adopted approach performs scaling directly in RNS domain by operating both hierarchically and individually at channel level and does not require reverse and forward conversions. Simple memoryless architectures are designed based on the obtained formulation. Here, scaling is performed in two levels. In first level, scaler is designed for three moduli sets in which scaling is performed completely by using Chinese Remainder Theorem. Then, this design is explored in second level for extended four moduli sets using mixed radix conversion technique. The proposed designs were evaluated and tested experimentally by synthesizing scalers using FPGA technology. Figure 7.1 and 7.2 show that the synthesis results of delay and area for these moduli sets respectively. From this graph, it has been concluded that with increase in the value of *n* critical path delay and area increases. For same value of *n*, the area increases drastically for extended four moduli set as compared to three moduli sets while increase in delay is comparable. Also, there is slightly increase in critical path delay when 2^r variable scaler were designed corresponding to exact scaler for same moduli set. The proposed architecture allows not only the design of static scalers but also support several augmented and extended moduli sets in FPGAs. A final conclusion is drawn from this work is that the proposed scalers are not only flexible but also represent a step forward in the design of high performance devices, particularly mobile systems.

7.2 Future Scope

The present work on the RNS scaler can be extended in various directions. Some suggestions are given below:

1. Scaling techniques can be extended to large number of moduli- set.

2. Delay and area can be improved by applying different multiplication algorithms for higher number of moduli sets.
3. Other arithmetic unit can be implement for modular arithmetic e.g. comparator.

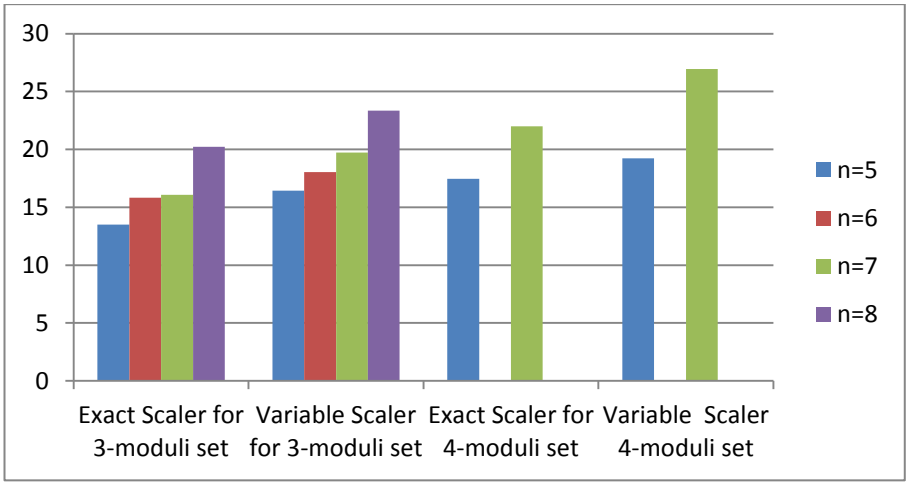

Fig.7.1 Delay (*ns*) comparison

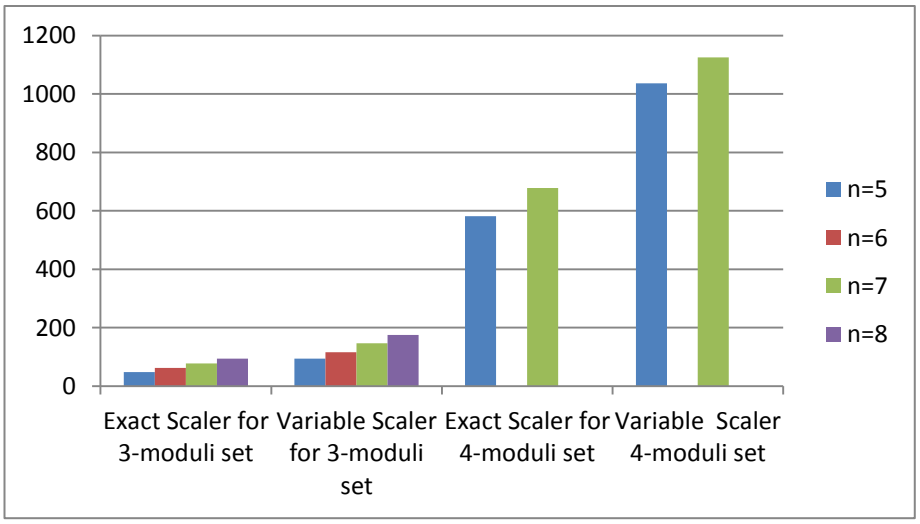

Fig. 7.2 Area (LUTs) Comparison

REFERENCES

[1] F. Barsi and M. Pinotti, "Fast base extension and precise scaling in RNS for look-up table implementations," *IEEE Transactions on Signal Processing,* vol. 43, no. 10, pp. 2427-2430, Oct. 1995.

[2] Garica and A. Lloris, "Pipelined RNS Multipliers: an application to scaling," *IEEE International Conference on Electronics Circuits and Systems,* Lisboa Portugal, vol. 3, pp. 55-58, Dec 1998.

[3] Z.Ulman and M. Czyzak, "Highly parallel, fast scaling of numbers in non rendundant residue arithmetic," *IEEE Transactions on Signal Processing,* vol. 46, no. 2 pp. 487- 496, Feb. 1998.

[4] Garcia and A. Lloris, "A look-up scheme for scaling in the RNS," *IEEE Trans. Comput.*, vol. 48, no. 7, pp. 748–751, Jul. 1999.

[5] K.P.Limand and A. B. Premkumar, "A modular approach to the computation of convolution sum using distributed arithmetic principles," *IEEE Trans. Circuits Syst. II, Analog Digit. signal Processing*, vol. 46, no. 1, pp. 92–96, Jan. 1999.

[6] Zhongde Wang, G. A. Jullien, and W. C. Miller, "An Improved Residue-to-Binary Converter," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 47, no. 9, pp. 1437-1440, Sep. 2000

[7] P.V. Ananda Mohan, *Residue Number Systems: Algorithms and Architectures*. New York: Kluwer Academic, 2002.

[8] Y. Wang, X. Song, M. Aboulhamid and H. Shen, " Adder based residue to binary converters for $\{2^n-1, 2^n, 2^n+1\}$," *IEEE Transactions on Signal Processing*, vol. 50, no. 7, pp. 1772-1779, Jul. 2002.

[9] H. Vergos, C. Efststhiou and D.G. Nikolos, "Diminished-one modulo 2^n+1adder design," *IEEE Transactions on computers,* vol.51, no. 12, pp. 1389-1399, Dec. 2002.

[10] N. Burgess, "Scaling an RNS number using the core function," in *Proc.16th IEEE Symp. Comput. Arith.*, Santiago de Compostela, Spain, pp. 262–269,Jun.2003.

[11] Cao, C. H. Chang, and T. Srikanthan, "An efficient reverse converter for the 4-moduli set $\{2^n-1, 2^n, 2^n+1\}$ based on the New Chinese Remainder Theorem," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 50, no. 10, pp. 1296–1303, Oct. 2003.

[12] R. Conway and J. Nelson, "Improved RNS FIR filter architectures," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 51, no. 1, pp. 26-28, Jan. 2004.

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

[14] P. V. A. Mohan, "RNS-to-binary converter for a new three-moduli set $\{2^n\text{-}1, 2^n, 2^n+1\}$," *IEEE Trans. Circuits Syst. II, Exp. Briefs*,vol. 54, no. 9, pp. 775–779, Sep. 2007.

[15] R. Chaves and L. Sousa, " Improving residue number system multiplication with more balanced moduli sets and enhanced modular arithmetic structure," *IET Proc. Comput. Digit. Tech.*, vol. 1, no. 5, pp. 472-480, Sep. 2007.

[16] R. Muralidharan, C. H. Chang, and C. C. Jong, "A low complexity modulo 2^n+1 squarer design," in *Proc. IEEE Asia-Pacific Conf. Circuits Syst.*, Macao, China, pp. 1296–1299, Dec. 2008.

[17] K.Yinan and B. Phillips, "Fast scaling in the residue number system," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 3, pp. 443–447, Mar. 2009.

[18] R.Muralidharan and C. H. Chang, "Hard multiple generator for higher radix modulo multiplication," in *Proc. 12th Int. Symp. Integr. Circuits*, Singapore, pp. 546–549, Dec. 2009.

[19] S.Ma, J. Hu, Y. Ye, L. Zhang, and X. Ling, "A 2^n scaling scheme for signed RNS integers and its VLSI implementation," *Sci. China, Inf. Sci.*, vol. 53, no. 1, pp. 203–212, Jan. 2010.

[20] Ramya Muralidharan & Chip-Hong Chang, "Radix-8 booth encoded modulo 2^n-multiplier with adaptive delay for high dynamic range residue number system," *IEEE Trans.Circuits Syst. I, Reg. Papers*, vol. 58, no.5, pp. 982-993, May 2011.

[21] Jeremy Yung Shern Low & Chip-Hong Chang, "Simple, fast and exact RNS scaler for the three-moduli set ," *IEEE Trans. Circuits Syst.I, Reg. Papers*, vol. 58, no. 11, pp. 2686–2697,Nov. 2011.

[22] L. Sousa and S. Antao, "MRC-based RNS converter for four-moduli sets $\{2^n+1, 2^n-1, 2^n, 2^{2n+1}+1\}$ and $\{2^n+1, 2^n-1, 2^{2n}, 2^{n+1}+1\}$," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 59, no.4, pp. 244-248, Mar. 2012.

[23] Ramya Muralidharan & Chip-Hong Chang, "Area-power efficient modulo 2^n-1 and modulo 2^n+1 multiplier for {2^n-1, 2^n, 2^n+1} based RNS," *IEEE Trans.Circuits Syst. I, Reg. Papers*, vol. 59, no.10, pp. 2263–2274, Oct. 2012.

[24] Deepak Garg & Sandeep Arya, "Design of Configurable Booth Multiplier Using Dynamic Range Detector", *International Journal of Electronics Engineering*, 2012.

[25] Jeremy Yung Shern Low & Chip-Hong Chang, "A VLSI efficient programmable power-of-two scaler for {2^n-1,2^n,2^n+1} RNS", *IEEE Trans. Circuits and Systems I, Reg. Papers*, vol. 59, no.12, pp. 2911–2919, Dec. 2012.

[26] T. Tay, C. H. Chang and J. Low, " Efficient VLSI implementation of 2^n scaling of signed integer in RNS {2^n-1,2^n,2^n+1}," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 10, pp. 1936-1940, Nov. 2013.

[27] L. Sousa, S. Antao and R. Chaves, "On the design of RNS reverse converters for the four-moduli set $\{2^n + 1, 2^n - 1, 2^n, 2^{n+1} + 1\}$," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no.10, pp. 1945-1949, Dec. 2013

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

[29] P.Matutino, R. Chaves and L. Sousa, "Arithmetic-based binary-to-RNS converter modulo $2^n \pm k$ for *jn*-bit dynamic range," *IEEE Transactions on Very Large Scale Integration Systems*,2014.

[30] P. Patronik and Stanislaw J. Piestrak, "Design of reverse converters for the new RNS moduli set $\{2^n + 1, 2^n - 1, 2^n, 2^{n-1} + 1\}$ (*n* odd)," *IEEE Transactions on Circuits and Systems, Reg, Papers*, vol. 61, no. 12, Dec. 2014.

[31] L. Sousa, "2^n RNS scalers for extended 4-moduli sets," *IEEE Transactions on Computers*, vol. 62, no. 12, pp. 1-14, Feb. 2015.