A New RNS 4-moduli Set for the Implementation of FIR Filters

by

Gayathri Chalivendra

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2011 by the
Graduate Supervisory Committee:

Sarma Vrudhula, Chair
Aviral Shrivastava
Bertan Bakkaloglu

ARIZONA STATE UNIVERSITY

May 2011

ABSTRACT

Residue number systems have gained significant importance in the field of high-speed digital signal processing due to their carry-free nature and speed-up provided by parallelism. The critical aspect in the application of RNS is the selection of the moduli set and the design of the conversion units. There have been several RNS moduli sets proposed for the implementation of digital filters. However, some are unbalanced and some do not provide the required dynamic range. This thesis addresses the drawbacks of existing RNS moduli sets and proposes a new moduli set for efficient implementation of FIR filters. An efficient VLSI implementation model has been derived for the design of a reverse converter from RNS to the conventional two's complement representation. This model facilitates the realization of a reverse converter for better performance with less hardware complexity when compared with the reverse converter designs of the existing balanced 4-moduli sets. Experimental results comparing multiply and accumulate units using RNS that are implemented using the proposed four-moduli set with the state-of-the-art balanced four-moduli sets, show large improvements in area (46%) and power (43%) reduction for various dynamic ranges. RNS FIR filters using the proposed moduli-set and existing balanced 4-moduli set are implemented in RTL and compared for chip area and power and observed 20% improvements. This thesis also presents threshold logic implementation of the reverse converter.

dedicated to my brother Sai and friend Samatha

ACKNOWLEDGEMENTS

I would like to express my gratitude and sincere thanks to my advisor and mentor Dr. Sarma Vrudhula, for his continuous support and guidance, during the course of the work. I am grateful to Dr. Aviral Shrivastava and Dr. Bertan Bakkaloglu for agreeing to be on my defense committee and for their time and efforts in reviewing my work.

I would like to acknowledge the valuable inputs provided by my friend and labmate Vinay Hanumaiah and convey sincere thanks to him. I also thank all the members of VEDA lab for their support and encouragement in finishing the thesis.

Finally, I take this opportunity to thank my family Srinivasulu, Sulochana, and Sai, and friends who have been my pillars of strength through out my career, and who helped me become who I am today.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Motivation

Digital signal processors (DSP) are the core of wide range of applications like audio, image and video processing and consumer electronics to name a few. Unlike general purpose microprocessors, DSPs involve repetitive numerical computations at high data rate. Most of the DSPs such as digital filters, correlators and FFT processors involve repetitive operations of addition, subtraction and multiplication on large integers. Such specialized needs of DSPs demand very high-speed VLSI implementation of arithmetic units that perform computations in real time as the data arrives. For instance the typical high data rate of a stereo equipment is 20KHz, which requires the computation speed of the DSP in the range of hundreds of millions per second. There has been significant research since the emergence of VLSI implementation of DSPs in 1970s on developing algorithms for high speed arithmetic operations [18]. These traditional approaches to improve speed have resulted in complex hardware and power hungry circuits to implement simple arithmetic operations.

The performance and complexity of an arithmetic circuit are highly dependent on word length. A smaller word length results in a faster system with less complex hardware. Residue number system (RNS) represents a large integer in slices of small integers. Arithmetic operations performed on large integers now can be performed on these small integers in parallel without carry propagation, thus improving the speed of the processor. This simple feature of RNS to reduce the word length of an operation makes it attractive for VLSI implementation of computational intensive DSP applications using low power architectures.

RNS speeds up simple arithmetic operations like addition, subtraction, and multiplication but it is complex to perform division, comparison, and sign-detection oper-

ations. Hence the advantages of RNS are apparent only to computationally intensive applications that involve only addition and multiplication. For example digital finite impulse response (FIR) filters involve only multiply and accumulate operations. This thesis proposes a new four-moduli residue number system for implementation of high-speed and low power FIR filters.

1.2 Introduction to the Thesis

Residue number system is represented by a set of relatively prime numbers called the moduli set. The challenging task in the implementation of RNS arithmetic units is the selection of moduli set. The moduli set selected should be able to cover the dynamic range demanded by the application as well as ensuring high-speed and low-cost implementation of the modular arithmetic units and the overhead units. For example, a 32 order FIR filter with 16 bit wide input data and co-efficients has dynamic range of $2*16+log_2 32 = 37$ bits and the moduli set selected to implement this filter should have a dynamic range of 2^{37} or higher. Early researchers proposed moduli sets of arbitrary integers which are pairwise prime. The realization of modular arithmetic operation for such moduli sets were based on look-up tables as ASIC based implementations are much complex. Example of such moduli set is $\{3,5,7,11,17,64\}$ [10]. A detailed study on the selection of moduli set based on the dynamic range of the application is carried out by Wang et.al in [21]. It is shown that the moduli of the form 2^n, 2^n-1, and 2^n+1 allow for efficient VLSI implementations of modulo arithmetic units. Additionally the complexity of the conversion units, especially the reverse converter unit from RNS to binary is simplified due to special properties of the moduli set. Increasing the number of moduli in the moduli set increases the parallelism of arithmetic operations but it in turn increases the complexity of the reverse converter design. Hence there is an optimal choice in the selection of number of moduli in the moduli set.

For digital filter applications, initially three-moduli sets [15, 6, 12, 20] were

common with $\{2^n, 2^n-1, 2^n+1\}$ moduli set being most popular. Although three moduli sets result in simple implementation of the reverse converter, the dynamic ranges provided by them are insufficient for higher order filters. For high dynamic range filters, four-moduli set is considered the suitable choice [2]. There are several four-moduli sets introduced in literature,

- $\{2^n, 2^n-1, 2^n+1, 2^{n+1}-1\}$, n is even [2, 8]
- $\{2^n, 2^n-1, 2^n+1, 2^{n-1}-1\}$, n is even [2]
- $\{2^n, 2^n-1, 2^n+1, 2^{n+1}+1\}$, n is odd [8]
- $\{2^n-1, 2^n, 2^n+1, 2^{2n}+1\}$ [1]
- $\{2^n-1, 2^n, 2^n+1, 2^{2n+1}-1\}$, $\{2^n-1, 2^n+1, 2^{2n}, 2^{2n}+1\}$ [9]

Of these moduli sets, [1, 9] provide high dynamic ranges of $5n$, $5n+1$ and $6n$ bits respectively but they suffer from the imbalance in speed in the RNS arithmetic channels. The slowest channels operate on $2n$, $2n+1$ and $2n$ bits respectively while the fastest channels operate on n bits. This wide difference may result in in-efficient distribution of computation load among the RNS channels and may not take much advantage of parallelism provide by RNS. The relatively balanced moduli sets are $\{2^n, 2^n-1, 2^n+1, 2^{n+1}-1\}$ and $\{2^n, 2^n-1, 2^n+1, 2^{n-1}-1\}$ for even n. For these moduli sets the slowest channel operate on $n+1$ bits and the fastest channels operate on n bits. However there is still some inherent difference in the speeds of the fastest (2^n) and slowest channels (2^n+1) due the variable complexity in the hardware architectures of the arithmetic channels. Also, there is a constraint on the nature of n to be even which limits the programmability of the moduli set for different dynamic ranges. This thesis addresses the above issues by proposing a new balanced four moduli set $\{2^k, 2^n-1, 2^n+1, 2^{n+1}-1\}$, where $k \in [n, 2n]$. The proposed moduli set is well bal-

anced and has programmable dynamic range. The main contributions of this thesis are:

1. Proposing a new balanced moduli set for implementing RNS based FIR filters. The proposed moduli set addresses the issues present in the existing 4-moduli RNS systems.

2. Design of efficient reverse converter from RNS to conventional number system for the residue number stem proposed by deriving an implementation friendly mathematical model.

1.3 Mathematical Background of RNS

Basic Definitions

This section detials some of the basic definitions used in discussing the mathematical background of RNS.

- Modulo

  Modulo of a number a with respect to number b is the remainder when a is divided by b. The modulo is also called as residues in RNS terminology. Modulo operation is represented in the thesis in either one of two forms: a mod b, or $|a|_b$.

- Congruence

  Two integers a and b are congruent modulo m ($a \cong b(\text{mod } m)$) if m divides exactly the difference of a and b or equivalently it may leave the same remainder when divided by m. For example $2 \cong 7(\text{mod } 5)$, $4 \cong 7(\text{mod } 3)$ etc.

- Multiplicative inverse

  The multiplicative inverse of a modulo m, represented as $|a^{-1}|_m$ is defined as follows.

$$(a * a^{-1}) \; mod \; m \cong 1 \tag{1.1}$$

There can be multiple multiplicative inverses of a modulo m. For example, some of the multiplicative inverses of 5 modulo 3 are 2, 5, 7 and it can be observed that these multiplicative inverses are congruent modulo m. Multiplicative inverse of a modulo m exists only if a and m are relatively prime. For example there is no multiplicative inverse for 4 modulo 6.

Representation of RNS

A RNS is defined by a set of relatively prime integers called moduli set. A large integer in weighted number system like 2's complement number system can be represented in RNS as the remainders (residues) of the integer when divided by each of the moduli in the moduli set. Consider an RNS defined by the moduli set $\{m_1, m_2, \cdots, m_n\}$ where $m_1, m_2, \cdots, m_n$ are relatively prime integers. An integer X in binary number system can be encoded using this RNS as n residues - $\{x_1, x_2, \cdots, x_n\}$, where

$$x_n = X \; mod \; m_n. \tag{1.2}$$

The range of binary numbers that can be represented by a given moduli set is called the dynamic range of the RNS. It is calculated as the product of all the moduli in the moduli set as follows,

$$M = \prod_{i=1}^{n} m_i. \tag{1.3}$$

If M is the dynamic range of a moduli set $\{m_1, m_2, \cdots, m_n\}$, then any number $X \leq M$ can be uniquely represented in RNS. It is the necessary condition that the moduli set should comprise of **relatively prime** integers. If this condition is not met, two or more numbers will have same RNS representation. The table 1.1 shows the RNS representations of random numbers that fall within the dynamic range of the moduli set $\{2,3,5\}$.

To represent negative numbers, the dynamic range is divided in to two equal parts. If M is the dynamic range of the moduli set $\{m_1, m_2, \cdots, m_n\}$, then any integer that falls with in $\{-(M-1)/2, (M-1)/2\}$ or $\{-M/2, M/2-1\}$ for odd and

Binary Number	RNS	Binary Number	RNS
0	{0,0,0}	15	{1,0,0}
1	{1,1,1}	16	{0,1,1}
2	{0,2,2}	17	{1,2,2}
3	{1,0,3}	18	{0,0,3}
4	{0,1,4}	19	{1,1,4}
5	{3,2,0}	20	{0,2,0}
6	{0,0,1}	21	{1,0,1}
7	{1,1,2}	22	{0,1,2}
8	{0,2,3}	23	{1,2,3}
9	{1,0,4}	24	{0,0,4}
10	{0,1,0}	25	{1,1,0}
11	{1,2,1}	26	{0,2,1}
12	{0,0,2}	27	{1,0,2}
13	{1,1,3}	28	{0,1,3}
14	{0,2,4}	29	{1,2,4}

Table 1.1: Examples of residue encoding

even M respectively can be represented uniquely in RNS. If the RNS representation of number X is $\{x_1, x_2, x_3\}$ then the RNS representation of the complement of X is $\{|m_1 - x_1|_{m_1}, |m_2 - x_2|_{m_2}, |m_3 - x_3|_{m_3}\}$. The table 1.2 shows the encoding of the negative numbers for the same RNS moduli set $\{2,3,5\}$.

Binary Number	RNS	Binary Number	RNS
0	{0,0,0}	-15	{1,0,0}
1	{1,1,1}	-14	{0,1,1}
2	{0,2,2}	-13	{1,2,2}
3	{1,0,3}	-12	{0,0,3}
4	{0,1,4}	-11	{1,1,4}
5	{3,2,0}	-10	{0,2,0}
6	{0,0,1}	-9	{1,0,1}
7	{1,1,2}	-8	{0,1,2}
8	{0,2,3}	-7	{1,2,3}
9	{1,0,4}	-6	{0,0,4}
10	{0,1,0}	-5	{1,1,0}
11	{1,2,1}	-4	{0,2,1}
12	{0,0,2}	-3	{1,0,2}
13	{1,1,3}	-2	{0,1,3}
14	{0,2,4}	-1	{1,2,4}

Table 1.2: Examples of residue encoding of negative numbers

1.4 Arithmetic Operations

All the arithmetic operations performed on two integers in binary number system are performed as modulo arithmetic operations on the residues in the residue number system. Consider two binary numbers X, Y and the corresponding RNS representations $\{x_1, x_2, \cdots, x_n\}$ and $\{y_1, y_2, \cdots, y_n\}$. If $Z = X\ op\ Y$, where op represents one of the arithmetic operations of addition, subtraction, or multiplication, then $Z = \{z_1, z_2, \cdots, z_n\}$ in RNS, where

$$z_i = (x_i\ op\ y_i)\ mod\ m_i, 1 \leq i \leq n$$

The calculation of z_i depends only on x_i and y_i and does not interact with the calculation of z_j for $j \neq i$ [18]. This property is termed as carry-free property of RNS. The carry-free property holds only for addition, subtraction and multiplication operations while division and scaling operations result in complicated operations that involve interactions between the residues. This is one of the main drawbacks of RNS. Hence RNS is more advantageous for computation intensive applications involving simple arithmetic operations like addition and multiplication. The application of RNS in general purpose processors in limited as division and comparison are common operations in general purpose processing.

The modulo operation is distributive over addition, subtraction and multiplication represented as,

$$|X\ op\ Y|_{m_1} = ||X|_{m1}\ op\ |Y|_{m_1}|_{m_1}.$$

Since RNS arithmetic is modular arithmetic, the hardware of units are more complex to build compared to conventional 2's complement binary arithmetic units.

1.5 Conversion Algorithms

The overhead associated with the implementation of an RNS processor are the conversion units that convert from a binary number system to RNS, and vice versa. This conversion is unavoidable as the peripheral interfaces of most digital systems are based on binary number system. A block diagram of a typical RNS processor is as shown in 1.1. The input X to the RNS system is available as binary input. It is first converted to residues. After processing the data, the result in the form of residues is converted to the conventional binary representation. The process of converting binary number to residues is called forward conversion and process of converting residues to binary numbers is called reverse conversion.

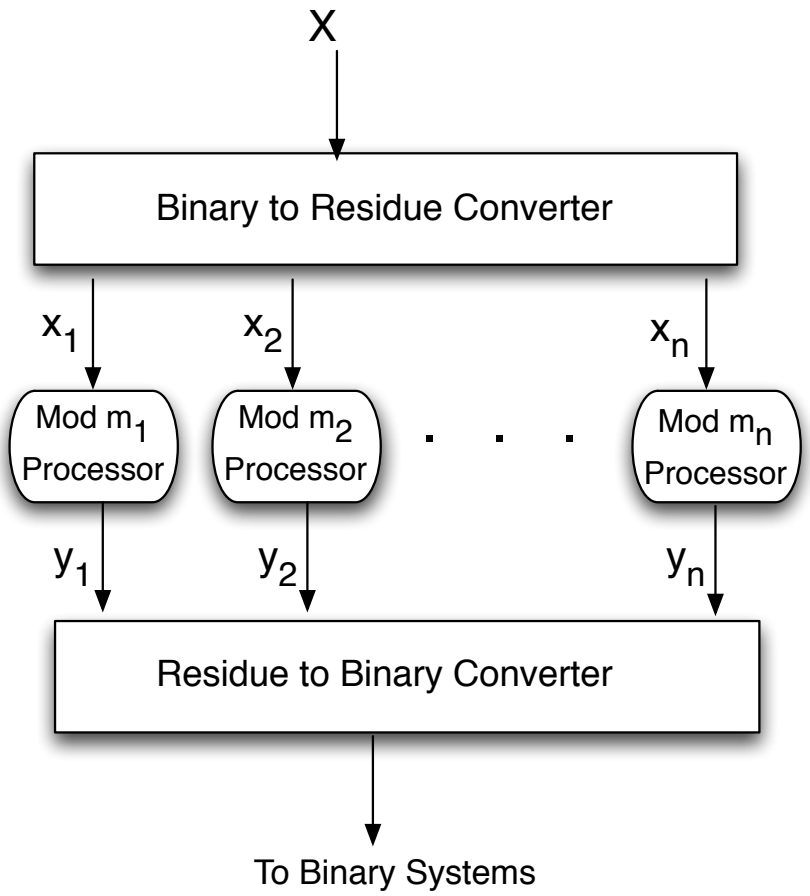

Figure 1.1: RNS Processor

Forward Conversion

Forward conversion involves the computation of remainders of input X with respect to each modulus in the RNS moduli set. There are well known algorithms for forward conversion [14, 17, 13, 11] in literature. The hardware complexity of the forward converter depends on the type of moduli set selected. For arbitrary moduli sets like $\{2,5,7,11,19\}$, the forward conversion involves conventional way of calculating the remainders using division algorithm. This is much complex to implement as combinational logic. Hence look-up tables are used to implement forward converters for arbitrary moduli sets. For special moduli sets like $\{2^n, 2^n-1, 2^n+1\}$ the architecture of forward converters is simple and can be implemented in hardware using modulo adders and or carry save adders due to the periodicity properties of the modulus of kind $2^n, 2^n-1$ and 2^n+1. The architecture of forward converter for special moduli is discussed in 3.

Numerical Example: Consider an RNS system with moduli set 4,3,5. The dynamic range of the system is 60 and numbers from -30 to 29 can be uniquely represented as residues. Some examples are listed in table1.3.

Binary Number	x_1	x_2	x_3
0	0	0	0
1	1	1	1
5	1	2	0
15	3	0	0
-21	3	0	4
-30	2	0	0

Table 1.3: Forward conversion examples

Reverse Conversion

Compared to forward conversion, reverse conversion is a much complex process and its complexity is completely determined by the chosen moduli set. Reverse conversion calculates the binary number X given the residues $\{x_1, x_2, .., x_n\}$ and the moduli

set $\{m_1, m_2, .., m_n\}$. Let $M = \prod_{i=1}^{n} m_i$ be the dynamic range. There are two popular algorithms in literature for reverse conversion.

- Reverse conversion (RC) based on the classical Chinese remainder theorem (CRT)

  Given a set of relatively prime moduli $\{m_1, m_2, \cdots, m_n\}$, the conventional representation X of its residues $\{x_1, x_2, \cdots, x_n\}$ is calculated using the following mathematical model.

$$X = |\sum_{i=1}^{n} x_i |M_i|_{m_i} M_i|_M, \tag{1.4}$$

  where, $M_i = M/m_i$.

- RC based on New Chinese remainder theorem

  Wang et.al [22] proposed a method for reverse conversion that is based on CRT that is more efficient in terms of hardware implementation. It is mathematically represented as,

$$\begin{aligned} X = |x_1 + (x_2 - x_1)k_1 m_1 + (x_3 - x_2)k_2 m_1 m_2 \\ + \cdots + (x_n - x_{n-1})k_{n-1} m_1 \cdots m_n|_M. \end{aligned} \tag{1.5}$$

  Notation $|A|_M$ indicates the remainder of A when divided with M. k_i are the multiplicative inverses such that,

$$\begin{aligned} k_1 m_1 &\cong 1 \ (mod \ (m_2 * m_3 \cdots m_n)), \\ k_2 m_1 m_2 &\cong 1 \ (mod \ (m_3 * m_4 \cdots m_n)), \\ &\vdots \\ k_{n-1} m_1 m_2 \cdots m_{n-1} &\cong \ 1 (mod \ m_n). \end{aligned}$$

  Example: Let the Binary representation of the residues $\{3,0,0\}$ with respect to the moduli set $\{4,3,5\}$ be X. Here $m_1 = 4$, $m_2 = 3$, $m_3 = 5$, and $M = 60$. The multiplicative inverses of m_1, m_2 are $k_1 = 4$, $k_2 = 3$ respectively since $k_1 m_1 = 16 \cong 1 \bmod m_2 m_3$, and $k_1 m_1 m_2 = 36 \cong 1 \bmod m_3$. Substituting these values in

(1.5),

$$X = |3 + (0-3)4*4 + (0-0)3*4*3|_{60},$$
$$X = |3-48|_{60} = |-45|_{60},$$
$$X = 60 - 45 = 15.$$

- Mixed radix conversion (MRC) algorithm

  According to MRC [18], the mathematical model for the reconstruction of X is,

$$X = a_n \prod_{i=1}^{n} m_i + \cdots + a_3 m_2 m_1 + a_2 m_1 + a_1, \tag{1.6}$$

  where $a_1 = x_1$, $a_2 = \left|(x_2 - a_1)\left|m_1^{-1}\right|_{m_2}\right|_{m_2}$ and so on. $|m_1^{-1}|_{m_2}$ is the multiplicative inverse of m_1 modulo m_2 such that $|m_1^{-1} m_1|_{m_2} = 1$.

  Example: Considering the same example as above. In this case,

$$a_1 = x_1 = 3,$$
$$a_2 = |4^{-1}(0-3)|_3$$
$$= |1(0-3)|_3 = 0,$$
$$a_3 = |3^{-1}([4^{-1}(0-3)] - 0)|_5$$
$$= |2([4(-3)] - 0)|_5 = 1.$$

  Substituting a_1, a_2, and a_3 in (1.6),

$$X = a_1 + a_2 m_1 + a_3 m_1 m_2,$$
$$X = 3 + 0 + 1*3*4 = 15.$$

Mixed radix conversion is a sequential process and is generally slow to implement reverse conversion compared to CRT based algorithms but it is simple to implement. The application of MRC algorithm is generally limited to two or three-moduli sets. The most popular algorithm used to implement reverse converter is Chinese remainder theorem.

1.6 Applications

Due to the carry-free nature, residue number encoding has gained importance in high-speed data processing applications where the critical path is associated with the propagation of the carry. Using RNS encoding, the word-length of the data operands is reduced and results in the minimization of critical path timing and in lower power consumption. RNS is fault tolerant and error detection and correction is easy as it facilitates the isolation of faulty residues. Due to these attractive properties of RNS, it is a promising alternative to conventional two's complement number system. Although RNS representation speeds up arithmetic operations like addition and multiplication, it is much more complex to perform other operations like division, shifting, comparison etc. This limits the application of RNS only to computationally intensive applications that require mainly addition and multiplication operations. Hence RNS has gained much popularity in the field of DSP and active research is going on in application of RNS in the following fields: Digital filtering- FIR and IIR filters, Digital convolution, Cryptography, Discrete Fourier, transform (DFT), Fast Fourier transform(FFT) processors, Digital image processing.

The use of RNS in general purpose processors where operations like division and comparison are common, is limited as it is more efficient to implement those operations in the conventional binary number system. As the RNS arithmetic operations are performed on inputs of smaller input width, lower power and higher speed can be expected.

Chapter 2

NEW RNS FOUR-MODULI SET FOR FIR FILTERS

2.1 Binary Vs RNS FIR Filter Architectures

The most popular use of RNS in the design of digital finite impulse response(FIR) filters. FIR filters are highly stable architectures and are less sensitive to quantization errors than filters of recursive architectures like Infinite impulse response (IIR) filters. A digital FIR filter response of N-taps is mathematically represented as (2.1) where x_n is the the input data and $a_1, a_2, \cdots, a_k$ are the filter co-efficients.

$$y_n = \sum_{k=0}^{N} a_k x_{n-k} \tag{2.1}$$

Generally, two's complement system (TCS) representation is widely chosen for the binary representation of the input and co-efficients of a digital filter. FIR filters can be implemented in hardware either in the Direct form, shown in the Fig 2.1 or in the Transpose form shown in the Fig 2.2. Direct form results in larger critical path delay of $t_D + t_{mul} + t_{adder(N)}$ compared to the critical path delay of the transposed form implementation which is $t_D + t_{mul} + t_{adder}$. Here, t_{mul} is the delay of the multiplier, $t_{adder(N)}$ is the delay of the adder tree adding N inputs and t_{adder} is a two-input adder delay and t_D is the delay of the register element. The Transpose form requires larger input buffers for the input x_n for it to be able to drive N multipliers. In general for ASIC implementations, the transpose form is preferred. For high speed implementations of transpose form FIR filters, the result of the multiplier is represented in carry-save format and the accumulator is implemented as carry save adder. The final stage of the such implementation of transpose form FIR filter is a conventional adder to add the carry save vectors of the last stage.

The dynamic range of a N-tap FIR filter with input width of M bits and co-efficient width of L bits is $M + L + log_2 N$. As the number of taps increases, the dynamic range of the filter increases, and the delay of the output adder increases due to

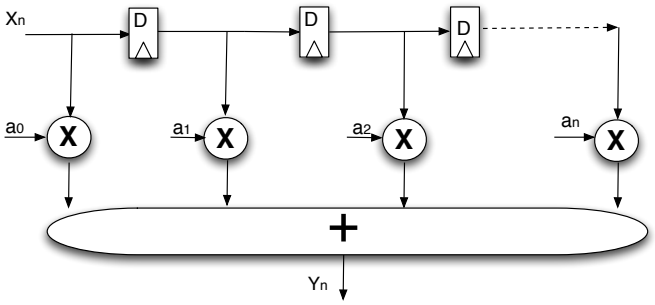

Figure 2.1: Direct form of FIR filters

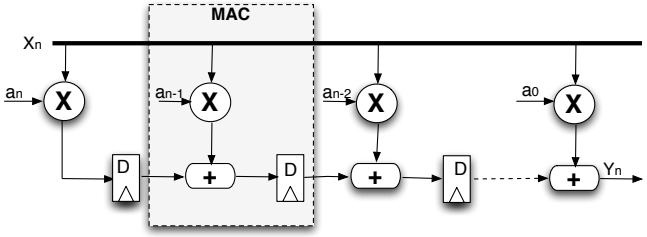

Figure 2.2: Transposed form of FIR filters

longer carry-propagation [19]. Using RNS, the dynamic range can be decomposed into smaller dynamic ranges and the MAC operations can be performed in parallel without carry propagation among the channels. Consider an RNS system of p-moduli set $\{m_1, m_2, \cdots, m_p\}$. The mathematical representation of FIR filter using the p-moduli set is,

$$\begin{aligned} y_{1,n} &= \left| \sum_{i=0}^{N} |a_{1,k} * x_1(n-k)|_{m_1} \right|_{m_1}, \\ y_{2,n} &= \left| \sum_{i=0}^{N} |a_{2,k} * x_2(n-k)|_{m_2} \right|_{m_2}, \\ &\vdots \\ y_{p,n} &= \left| \sum_{i=0}^{N} |a_{p,k} * x_p(n-k)|_{m_p} \right|_{m_p}. \end{aligned} \tag{2.2}$$

Here, $a_{1,k}, a_{2,k}, \cdots, a_{p,k}$ represent the residues of the filter co-efficients and $x_1, x_2, \cdots, x_p$ represents the residues of the input. In RNS FIR filter using p-moduli set, there are p filters operating in parallel without any inter-dependency. Due to parallelism and carry free-property nature of RNS, high-speed FIR filters are realizable compared to conven-

tional TCS representation. In addition to gain in performance, RNS filter architectures result in low power in the following ways [5].

- Reduction in the peak current: Compared to conventional implementation of FIR filters, RNS architectures uses smaller arithmetic units and less complex designs. Hence, the peak current in each arithmetic unit decreases.

- Reduction in the switching activity: The reason mentioned above is applied for smaller switching activities in RNS arithmetic units. As RNS systems operate on smaller input widths, the switching activities are also relatively smaller. The reduction in peak current as well as switching activity results in smaller dynamic power.

- Several other circuit level power reduction techniques like voltage scaling in non-critical paths using high threshold transistors can be applied very easily in RNS circuits. The non-critical channel can be completely implemented using high threshold transistors. In conventional binary systems, there are only specific paths where high threshold transistors can be used.

The FIR filter architecture using RNS is as shown in Fig 2.3. The only overhead in the implementation of RNS FIR filters is the conversion units from binary to RNS and the reverse converter to convert the individual filter responses to binary response. There are three basic steps in the implementation of RNS FIR filter using the moduli set $\{m_1, m_2, \cdots, m_p\}$.

1. Forward conversion: Let the input data sample at time n is $X(n)$ and filter co-efficients are a_k. The data input and the filter co-efficients are converted to

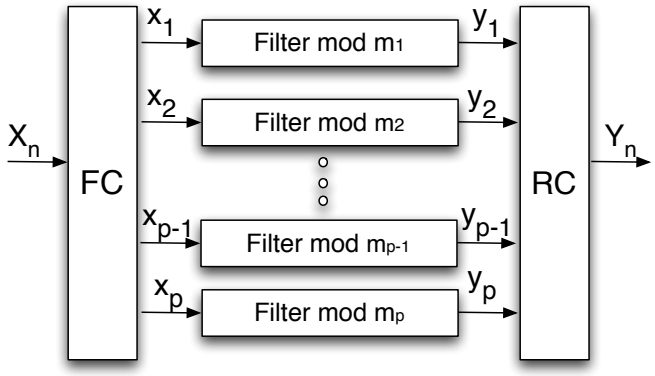

Figure 2.3: RNS FIR filter architecture

residues using modulo operations as shown in the following equations.

$$\begin{aligned} x_1(n) &= x(n)\ mod\ m_1,\ a_{1,k} = a_k\ mod\ m_1 \\ x_2(n) &= x(n)\ mod\ m_2,\ a_{2,k} = a_k\ mod m_2 \\ &\vdots \\ x_p(n) &= x(n)\ mod\ m_p, a_{p,k} = a_k\ mod\ m_p \end{aligned} \tag{2.3}$$

2. Modulo filters: The modulo filters are conventional filter with all the arithmetic operations being modulo arithmetic operations. The multipliers and the adders in conventional filters are replaced by modulo multipliers and adders respectively in RNS filters as shown in 2.4. For example multiplication in binary is converted

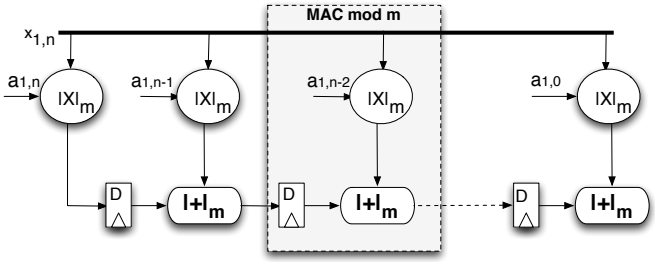

Figure 2.4: Modulo Filter

in to p-modulo multiplications in parallel as shown in (2.4).

$$\underbrace{x(n-k)*a_k}_{Binary} = \underbrace{\{(x_1(n-k)*a_{1,k})\ mod\ m_1, \cdots, (x_p(n-k)*a_{p,k})\ mod\ m_p\}}_{RNS} \tag{2.4}$$

3. Reverse conversion: Individual filter responses $y_{1,n}$,$y_{2,n}\cdots,y_{p,n}$ to the final response $Y_n = RC(y_1, y_2, \cdots, y_p)$ using popular conversion algorithms discussed in chapter 1.

2.2 A Study on Existing RNS Moduli Sets

Selection of moduli set is critical factor in determining the performance and power of an RNS system. The moduli set selected to implement FIR filters should cover the dynamic range of the filter. This in turn impacts the through put of the filter and the hardware efficiency of the forward converter, reverse converter, and modulo MAC units.

If n is the input width and assuming the filter coefficient width to be n, for an Nth order FIR filter, the output width without scaling is $2n+log2N$. Hence the dynamic range of the selected moduli set of the filter should be at least $2n+log2N$. For example, for a 40 tap RNS FIR filter with 16 bit input width and 16 bit co-efficient width, the dynamic range of the moduli set selected should be $32+log_240 = 36$ bits. There are two ways to achieve a higher dynamic range.

- Use a large number of moduli each with smaller magnitude: The moduli set consists of large number of relatively prime numbers. An example for this type of moduli set with dynamic range of 40 bits is {16,17,19,53,127,129,257}. Implementing modulo arithmetic units using the moduli set is not simple. For this reason, ROM table-lookup tables are used to implement modulo addition, subtraction and multiplications. Also increasing the number of moduli increases the reverse converter complexity. Hence for large dynamic range applications, moduli set of arbitrary prime moduli is not suitable for ASIC implementations.

- Use of small number of moduli with large magnitude: Examples for this type of moduli set are $\{2^n, 2^n-1, 2^n+1\}$, $\{2^n, 2^n-1, 2^n+1, 2^{n+1}+1\}$ etc. In these types

of moduli sets, the moduli are of the form 2^n, 2^n-1 and 2^n+1 and the modulo arithmetic blocks with respect to such moduli can be efficiently implemented as digital VLSI circuits due to the special properties of the moduli.

To implement a RNS FIR filter of 40 bits dynamic range, some of the choices are $\{2^{14}, 2^{14}-1, 2^{14}+1\}$, $\{2^{10}, 2^{10}-1, 2^{10}+1, 2^{11}+1\}$. The popular moduli sets of this form are the 3-moduli sets and the 4-moduli sets. There are few 5-moduli sets proposed in literature but the design of the reverse converter is complex and its overhead is substantially larger in terms of delay and power.

Three-moduli Sets

The most popular three-moduli set in the literature is $\{2^n, 2^n-1, 2^n+1\}$. Its main drawback is the larger difference in the critical path delays of the arithmetic channels. The binary channel 2^n is the fastest channel and the non-binary channel 2^n+1 is the slowest channel owing to the architecture difference in the modular arithmetic units. Any arithmetic operation modulo 2^n is performed as conventional arithmetic operation by discarding the higher order bits positioned after the bit position n. The arithmetic operations modulo 2^n+1 are much more complex, and involve addition of correction factors and carry save addition involves end-around carries. This difference in the speeds results in an inefficient distribution of computation load among different channels. To address this imbalance, [3, 4] proposed three -moduli set $\{2^k, 2^n-1, 2^n+1\}$, $k > n$ which has wider binary channel.

However, for smaller input widths and higher order filters, this moduli set does not provide any performance improvement over conventional binary filter. For example, consider a 8 bit wide filter with 64 taps. The dynamic range is $16 + log_2 64 = 22$. The best suitable moduli set with dynamic range of 22 bits is $\{2^8, 2^7-1, 2^7+1\}$. In this case, the modulo 2^k filter operates on 8-bit inputs, as does conventional filter. In such case we did not gain much advantage using RNS over conventional filter. For some

dynamic ranges, 3-moduli sets are advantageous. For example, if the input width is 16 and the filter taps are 8, the dynamic range is 35. The moduli set $\{2^{13}, 2^{11}-1, 2^{11}+1\}$ gives better speed compared to 2's complement implementation as the number of bits of MAC operation are reduced from 16 to 13. Hence for smaller input-widths, the parallelism provided by 3-moduli set is insufficient.

An experimental study on RNS FIR filters implemented using the balanced 3-moduli set $\{2^k, 2^n-1, 2^n+1\}$ was conducted to check the performance parameters. Binary FIR filters and RNS FIR filter with the moduli set $\{2^k, 2^n-1, 2^n+1\}$ are implemented in RTL and synthesized for minimum delay using commercial 65nm technology library. Figure 2.5, and figure 2.6 show the area and delay comparison of Binary FIR filters and RNS FIR filters with input and co-efficient widths of 24bits and figure 2.7, and figure 2.8 are for input and co-efficient widths of 28 bits. From the delay plots, it is observed that as the number of taps increases, the dynamic range increases and the advantage in speed by using RNS diminishes. It is also observed that the area advantage in RNS filters is small, and is less than 9% in most of the designs.

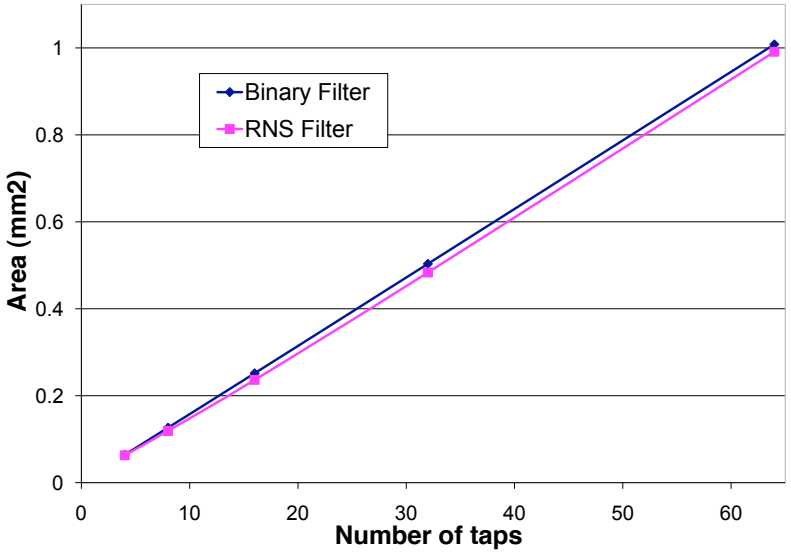

Figure 2.5: Comparison of area of Binary and RNS FIR filters with 24 bit input width

The experimental results show that the three-moduli set $\{2^k, 2^n-1, 2^n+1\}$ has

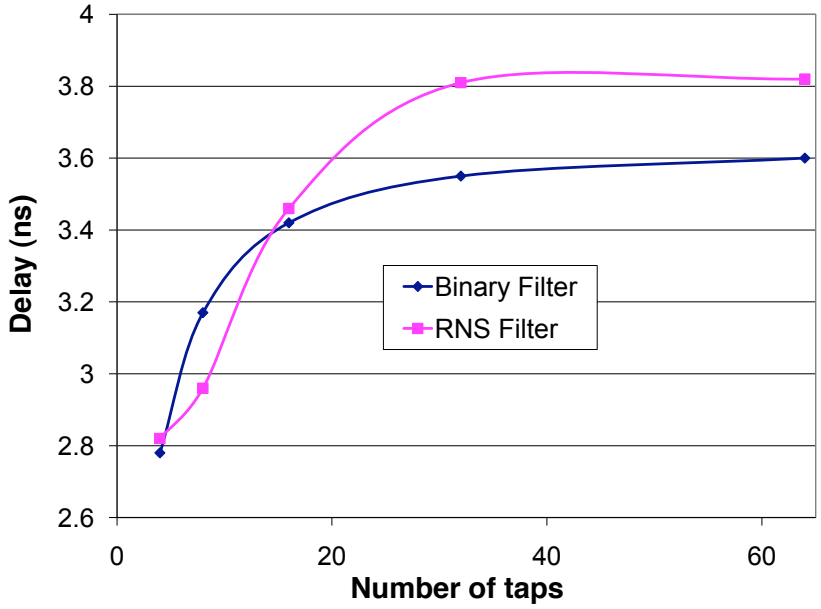

Figure 2.6: Comparison of delay of Binary and RNS FIR filters with 24 bit input width

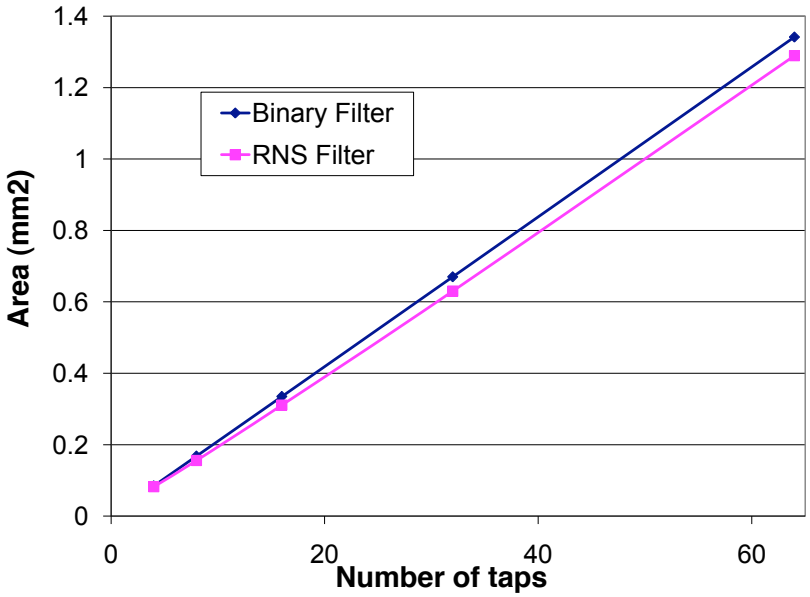

Figure 2.7: Comparison of area of Binary and RNS FIR filters with 28 bit input width

smaller dynamic range and is not beneficial to implement higher order FIR filter architectures.

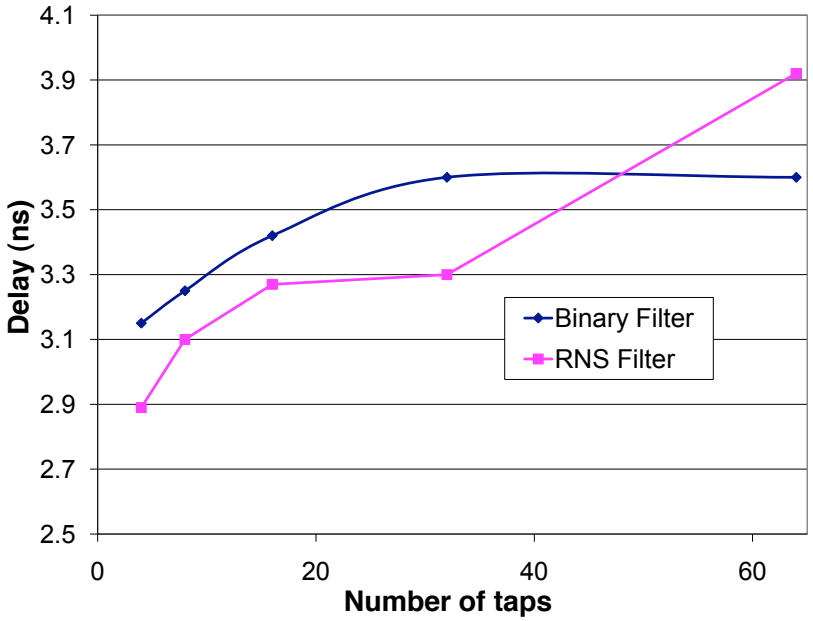

Figure 2.8: Comparison of delay of Binary and RNS FIR filters with 28 bit input width

Four-moduli Sets

Next, the implication of 4-moduli sets on the performance of RNS FIR filters is studied. Several 4-moduli sets and their optimal reverse converter design have been described in the literature: $\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1\}$[2], $\{2^n-1, 2^n, 2^n+1, 2^{n+1}+1\}$[6], $\{2^n-1, 2^n, 2^n+1, 2^{2n}+1\}$[1], $\{2^n-1, 2^n, 2^n+1, 2^{2n+1}-1\}$[7] and $\{2^n-1, 2^{2n}, 2^n+1, 2^{2n}+1\}$[7]. All these moduli-sets have imbalance in speeds of the arithmetic channels.

To quantify the amount of imbalance, an experiment was conducted to calculate the delay of the modulo MAC units - MAC mod 2^n, MAC mod 2^n-1, MAC mod 2^n+1. The simulation delay of the fastest and slowest channels of RNS MAC units using the popular 4-moduli sets are shown in Fig. 2.9. The differences in the speeds of the slowest and fastest channel are 11%, 16%, 26%, 23% and 22% respectively. Of these four-moduli sets, the relatively balanced moduli set is $\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1\}$ referred as *Cao-mod4* moduli set through out the thesis.

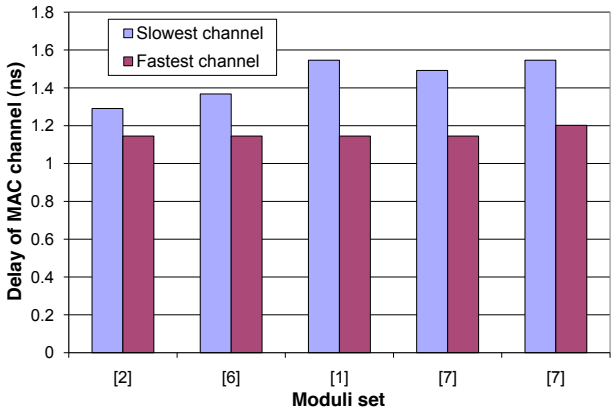

Figure 2.9: Comparison of delay of arithmetic channels of MACs

To address this issue of imbalance and for high dynamic range applications, this work proposes a new moduli set $\{2^k, 2^n-1, 2^n+1, 2^{n+1}-1\}$ where $k \in [n, 2n]$ is the selectable parameter. There is limit set on the parameter k, as arbitrary increase will again result in the imbalance in the modulo channels, with 2^k channel being the critical channel. The next section lists the advantages of the proposed 4-moduli set. This moduli set is referred to as *k-mod4* moduli set through out the thesis.

2.3 Advantages of the Proposed Moduli Set

Compared to the different four-moduli sets in literature, the proposed four-moduli set is much balanced, programmable and has less number of unused states.

1. Programmable dynamic range:

   The dynamic range of the proposed moduli set referred as *k-mod4* moduli set can be programmed by tuning k and fixing n. In case of other moduli sets for e.g.,$\{2^n, 2^n-1, 2^n+1, 2^{n+1}-1\}$ referred as *Cao-mod4*, to increase the dynamic range n has to be tuned. Changing the value of n would result in increase in hardware complexity of all the arithmetic channels. While in case of k-mod4 moduli set, only the arithmetic channel 2^k has to be modified to incorporate the change in dynamic range. The additional hardware cost to increase the dynamic

range in case of k-mod4 system is smaller to that Cao-mod4 RNS system.

Consider an example of $n = 4$, the moduli set $\{2^n, 2^n - 1, 2^n + 1, 2^{n+1} - 1\}$ provides a dynamic range of 17 bits. To implement an application with 18 bits dynamic range in RNS using Cao-mod4 moduli set, n has to be chosen as 6 and the moduli set is $\{2^6, 2^6 - 1, 2^6 + 1, 2^7 - 1\}$. As n is even, the next available value of n to tune for the higher dynamic range is 6. This will result in more hardware associated with increased power consumption and delay of all the modulo arithmetic channels. In case of k-mod4 moduli set k can be tuned to $k = 5$ and with $n = 4$, we can achieve the dynamic range of 18 bits using the moduli set $\{2^5, 2^4 - 1, 2^4 + 1, 2^5 - 1\}$.

2. Reduced number of unused states:

   The number of unused states in a moduli set is calculated as the difference between the dynamic range required by an application and the dynamic range offered by the moduli set. The fine programmability of dynamic range of the k-mod4 moduli set by tuning k would also result in less number of unused states for certain dynamic ranges compared to Cao-mod4 moduli set.

   For example, a 16 order FIR filter with 16 bit wide input data and co-efficients has a dynamic range of $(2 * 16 + log_2 16) = 36$ bits and the moduli set selected to implement this filter should have a dynamic range of 36 bits or higher. To implement this filter, $n = 10$ for Cao-Mod4 moduli set and $n = 8$, $k = 12$ for k-mod4 moduli set. In this case,

   the number of unused states for the Cao-mod4 moduli set

   = $(2^{10}(2^{20} - 1)(2^{11} - 1)) - 2^{36} = 2129227940864$ and

   the number of unused states for the k-mod4 moduli set

   = $(2^{11}(2^{16} - 1)(2^9 - 1)) - 2^{36} = 68448948224$.

3. Balanced moduli set:

The gap between the speed of the fastest binary channel and the slowest channel is reduced by overloading the number of bits, the channel 2^k operates on. But arbitrary increase of k would again result in imbalance in the arithmetic channels, hence the upper bound of k is limited to $2n$.

2.4 Design of Reverse Converter

For the proposed moduli set $\{2^k, 2^n-1, 2^n+1, 2^{n+1}-1\}$, a simple reverse conversion model is derived based on the standard approach of design of 4-moduli set reverse converters proposed in [2]. Let x_1, x_2, x_3, and x_4 represent the residues of a binary number X with respect to the moduli 2^k, 2^n+1, 2^n-1, and $2^{n+1}-1$ respectively. Given the residues and the moduli set, X can be reconstructed in two steps.

1. Partially reconstruct the binary number X_1 of the original binary X from the residues x_1, x_2, x_3 with respect to the three-moduli set $\{2^k, 2^n-1, 2^n+1\}$. X_1 is obtained using the 3-moduli reverse converter proposed in [4]. X_1 is represented as $2^{2n}Y_1 + x_1$ where Y_1 is the intermediate result of $2n$ bits wide.

2. Create a single modulus from the three moduli set $(2^k, 2^n-1, 2^n+1)$ by multiplying the moduli i.e., modulus $2^k(2^{2n}-1)$. Given X_1, x_4 and the two-moduli set $\{2^k(2^{2n}-1), 2^{n+1}-1\}$, X is reconstructed using MRC algorithm.

Reverse Converter Design for the Two-moduli Set $\{2^k(2^{2n}-1), 2^{n+1}-1\}$

Reconstruction of the binary result from the residues X_1 and x_4 w.r.t the moduli set $\{2^k(2^{2n-1}), 2^{n+1}-1\}$ is computed using MRC algorithm as follows,

$$X = a_1 + a_2 P_1, \tag{2.5}$$

where

$$a_1 = X_1 = x_1 + 2^k Y_1, \tag{2.6}$$

$$a_2 = \left| (x_4 - X_1) \left(\left| P_1^{-1} \right|_{P_2} \right) \right|_{P_2}, \tag{2.7}$$

$$P_1 = 2^k (2^{2n} - 1), \text{ and} \tag{2.8}$$

$$P_2 = 2^{n+1} - 1. \tag{2.9}$$

$\left|P_1^{-1}\right|_{P_2}$ is the multiplicative inverse of P_1 modulo P_2 i.e,

$$|P_1^{-1} P_1|_{P_2} = 1. \tag{2.10}$$

The multiplicative inverse of $|P_1|_{P_2}$ is given by the following lemma.

Lemma:

$$|P_1^{-1}|_{P_2} = \begin{cases} \left| \left(-\frac{1}{3}\right) 2^{n+3-k} \right|_{2^{n+1}-1}, & k < n+3 \\ \left| \left(-\frac{1}{3}\right) 2^{2n+4-k} \right|_{2^{n+1}-1}, & k \geq n+3. \end{cases} \tag{2.11}$$

Proof: First $|P_1|_{P2}$ is simplified as follows,

$$\begin{aligned} |P_1|_{P2} &= |2^k (2^{2n} - 1)|_{2^{n+1}-1} \\ &= |2^k (2^{n-1}(2^{n+1} - 1) + 2^{n-1} - 1)|_{2^{n+1}-1} \\ &= |2^k (2^{n-1} - 1)|_{2^{n+1}-1} = |2^{k-2}(2^{n+1} - 4)|_{2^{n+1}-1} \\ &= |2^{k-2}(-3)|_{2^{n+1}-1}. \end{aligned} \tag{2.12}$$

With this simplification, the lemma can be verified by substituting (2.11) and (2.12) in (2.10).

Case 1: When $k < n+3$,

$$\begin{aligned} |P_1^{-1} P_1|_{2^{n+1}-1} &= \left| \left(-\frac{1}{3}\right) 2^{n+3-k} 2^{k-2} (-3) \right|_{2^{n+1}-1} \\ &= |2^{n+1}|_{2^{n+1}-1} = 1. \end{aligned} \tag{2.13}$$

Case 2: When $k \geq n+3$,

$$\begin{aligned} |P_1^{-1}P_1|_{2^{n+1}-1} &= \left| \left(-\frac{1}{3}\right) 2^{2n+4-k} 2^{k-2} (-3) \right|_{2^{n+1}-1} \\ &= |2^{2(n+1)}|_{2^{n+1}-1} \\ &= \left(|2^{n+1}|_{2^{n+1}-1}\right) \left(|2^{n+1}|_{2^{n+1}-1}\right) = 1. \end{aligned} \tag{2.14}$$

In order to simplify the subsequent derivations,
$\left|P_1^{-1}\right|_{P_2} = \left|(-\frac{1}{3})2^{k'}\right|_{2^{n+1}-1}$, where

$$k' = \begin{cases} n+3-k, & k < n+3, \\ 2n+4-k, & k \geq n+3. \end{cases} \tag{2.15}$$

Knowing the multiplicative inverse, X is computed by substituting a_1, a_2 and P_1 from (2.6), (2.7) and (2.8) respectively in (2.5),

$$\begin{aligned} X &= X_1 + 2^k(2^{2n}-1) \left| (x_4 - X_1) \left(-\frac{1}{3}\right) 2^{k'} \right|_{2^{n+1}-1} \\ &= X_1 + 2^k(2^{2n}-1) \left| \left(\frac{1}{3}\right) (X_1 2^{k'} - x_4 2^{k'}) \right|_{2^{n+1}-1} \\ &= X_1 + 2^k(2^{2n}-1)Z = x_1 + Y_1 2^k + 2^k(2^{2n}-1)Z \\ &= x_1 + 2^k(Y_1 + 2^{2n}Z - Z). \end{aligned} \tag{2.16}$$

In the above equations,
$Z = \left| \left(\frac{1}{3}\right) (X_1 2^{k'} - x_4 2^{k'}) \right|_{2^{n+1}-1} = C(A+B)$ (say), where

$$\begin{aligned} C &= \left| \left(\frac{1}{3}\right) \right|_{2^{n+1}-1}, \\ A &= \left| X_1 2^{k'} \right|_{2^{n+1}-1}, \\ B &= \left| -x_4 2^{k'} \right|_{2^{n+1}-1}. \end{aligned}$$

The simplifications of A, B and C are given below.

$$\begin{aligned} A = \left| X_1 2^{k'} \right|_{2^{n+1}-1} &= \left| \left(x_1 + 2^k Y_1\right) 2^{k'} \right|_{2^{n+1}-1} \\ &= |A_1 + A_2|_{2^{n+1}-1}, \end{aligned} \tag{2.17}$$

where $A_1 = |\underbrace{x_1}_{k} 2^{k'}|_{2^{n+1}-1}$ and $A_2 = |\underbrace{Y_1}_{2n} 2^{k+k'}|_{2^{n+1}-1}$.

Since x_1 is a k bit vector and $n \leq k \leq 2n$, x_1 is split into two vectors x_{11} and x_{12}, each of $n+1$ bits. This is done to remove the modular operation w.r.t $2^{n+1}-1$.

$$x_{11} = \underbrace{0x_{1,k-1}, \cdots, x_{1,k-n}}_{n+1}. \tag{2.18}$$

$$x_{12} = \underbrace{00\cdots0}_{2n-k+1}\underbrace{x_{1,k-n-1}, \cdots, x_{1,0}}_{k-n}. \tag{2.19}$$

With $x = x_{11}x_{12}$, A_1 in (2.17) is computed as

$$\begin{aligned} A_1 &= \left|\left(x_{11}2^{k-n} + x_{12}\right)2^{k'}\right|_{2^{n+1}-1} \\ &= \left|x_{11}2^{k-n+k'} + x_{12}2^{k'}\right|_{2^{n+1}-1} \\ &= |A_{11} + A_{12}|_{2^{n+1}-1}. \end{aligned} \tag{2.20}$$

where

$$A_{11} = \underbrace{x_{11,n-k'_{11}}, \cdots, x_{11,0}}_{n+1-k'_{11}}\underbrace{x_{11,n}, \cdots, x_{11,n-k'_{11}+1}}_{k'_{11}}, \tag{2.21}$$

$$A_{12} = \underbrace{x_{12,n-k'}, \cdots, x_0}_{n+1-k'}\underbrace{x_{12,n}, \cdots, x_{12,n-k'+1}}_{k'}, \text{ and} \tag{2.22}$$

$$k'_{11} = |k-n+k'|_{n+1}. \tag{2.23}$$

In a similar fashion to splitting of x_1, Y_1 is also split into two vectors of $n+1$ bits.

$$Y_{11} = 00\underbrace{Y_{1,2n-1}, \cdots, Y_{1,n+1}}_{n-1}. \tag{2.24}$$

$$Y_{12} = \underbrace{Y_{1,n}, \cdots, Y_{1,0}}_{n+1}. \tag{2.25}$$

A_2 is computed with Y_{11} and Y_{12} as follows,

$$\begin{aligned} A_2 &= \left|(Y_{11}2^{n+1}+Y_{12})2^{k+k'}\right|_{2^{n+1}-1} \\ &= \left|Y_{11}2^{n+1+k+k'}+Y_{12}2^{k+k'}\right|_{2^{n+1}-1} \\ &= |A_{21}+A_{22}|_{2^{n+1}-1}\,, \end{aligned} \tag{2.26}$$

where

$$A_{21} = \underbrace{Y_{11,n-k'_{21}},\cdots,Y_{11,0}}_{n+1-k'_{21}}\underbrace{Y_{11,n},\cdots,Y_{11,n-k'_{21}+1}}_{k'_{21}}, \tag{2.27}$$

$$A_{22} = \underbrace{Y_{12,n-k'_{22}},\cdots,Y_{11,0}}_{n+1-k'_{22}}\underbrace{Y_{12,n},\cdots,Y_{12,n-k'_{22}+1}}_{k'_{22}}, \tag{2.28}$$

$$k'_{21} = |n+1+k+k'|_{n+1}, \quad k'_{22} = |k+k'|_{n+1}. \tag{2.29}$$

Simplifying B and C,

$$B = \left|-x_4 2^{k'}\right|_{2^{n+1}-1} = \underbrace{\bar{x}_{4,n-k'},\cdots,\bar{x}_0}_{n+1-k'}\underbrace{\bar{x}_{4,n},\cdots,\bar{x}_{4,n-k'+1}}_{k'}. \tag{2.30}$$

$$C = \left|\left(\frac{1}{3}\right)\right|_{2^{n+1}-1} = \sum_{i=0}^{n/2} 2^{2i}\ [2]. \tag{2.31}$$

Chapter 3

RNS FIR Filter Implementation

RNS implementation of FIR filter using the proposed moduli set comprise of the following components forward converters, modulo MAC blocks and a reverse converter as shown in Figure 3.1. This chapter details the implementation of each component in building the complete RNS FIR filter.

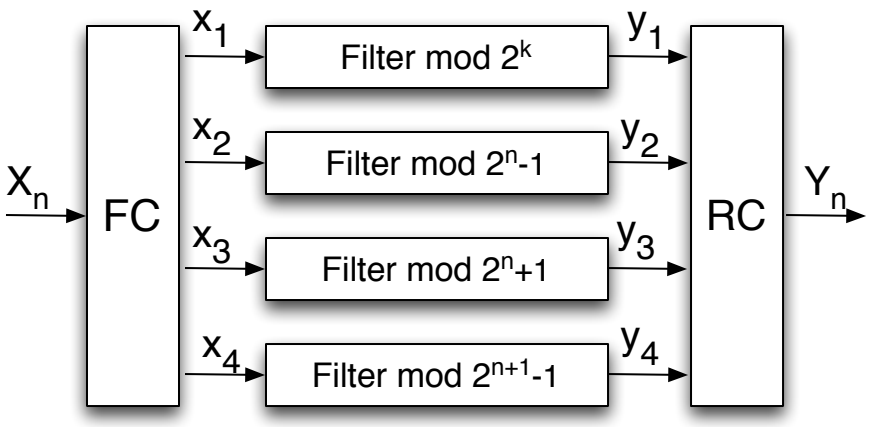

Figure 3.1: RNS implementation of a FIR filter

Forward Converter

The forward converter for this moduli set has 4 units that calculate the residue of the input with respect to moduli 2^k, 2^n-1, 2^n+1, and $2^{n+1}-1$.

- Modulo 2^k: If X is the input of p bits wide, then X *mod* 2^k is simply calculated by discarding the most significant bits to the kth bit position. This does not require additional hardware except routing. For example if X is represented as $X_{p-1}X_{p-2}\cdots X_0$, then X *mod* $2^k = X_{k-1}\cdots X_0$.

- Modulo 2^n-1(Modulo $2^{n+1}-1$): There are popular algorithms to calculate residues of the modulo of kind 2^n-1. For the present implementation of modulo

$2^n - 1$ and modulo $2^{n+1} - 1$ operations, architecture proposed in [11] is used. First step in the process of modulo calculation is to represent input X in slices of n $(n+1)$ bits. These operands generated are added using a multi-operand modulo adder (MOMA). A MOMA comprises of carry save adders (CSA) with end-around-carry to reduce multiple operands to two vectors (carry vector and save vector) of n bits wide. These two vectors are added using a modulo $2^n - 1$ adder. An example of a carry save addition of three inputs with EAC is shown in Figure 3.2. An example of CSA tree that reduces 5 input operands of n bits wide to two carry and save vectors is as shown in Figure 3.3. If S, C are the output

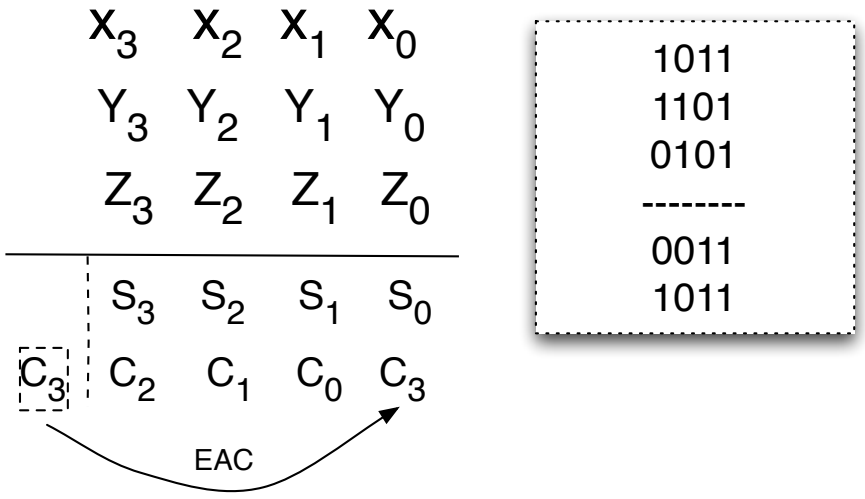

Figure 3.2: Example of a CSA with end-around-carry

vectors of CSA tree then,

$$(S+C)\ mod\ (2^n-1) = \begin{cases} S+C, & S+C < (2^n-1), \\ S+C-(2^n-1), & S+C \geq (2^n-1). \end{cases} \tag{3.1}$$

The modulo $2^n - 1$ adder can be implemented using a ripple carry adder with its carry out bit being fed back as its carry in (cin) bit. The critical path of such adder of n bits wide will be $2n$ full adder delays. Instead, it can be implemented as two ripple carry adders operating in parallel with cin=0 and cin=1 as carry in bits and a mux as shown in Figure 3.4. The critical path delay in this case is n full

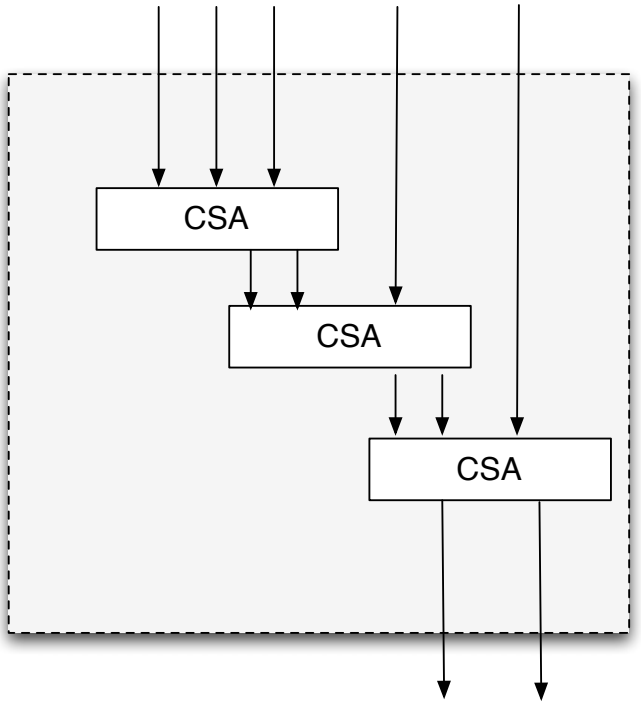

Figure 3.3: Example of 5 input CSA

adder delay and a 2:1 mux delay. This high speed implementation of the modulo adder is used in the current design.

- Modulo 2^n+1 : The architecture of modulo 2^n+1 residue generator is same as that of modulo 2^n-1 residue generator. The general architecture of it has three units - operand generation unit, CSA tree with EAC and a modulo 2^n+1 adder. The input bits are arranged using the periodicity property of the moduli 2^n+1 as operands of n bits. The operands generated and the correction factor are added using carry save adders modulo 2^n+1. The modulo 2^n+1 carry save addition is as explained in Figure 3.5. The final result is calculated using modulo 2^n+1 adder. If S, C are the output vectors of CSA tree, then the result of the modulo adder is as follows:

$$(S+C)\ mod\ (2^n+1) = \begin{cases} S+C, & S+C \leq (2^n+1), \\ S+C-(2^n+1), & S+C > (2^n+1). \end{cases} \tag{3.2}$$

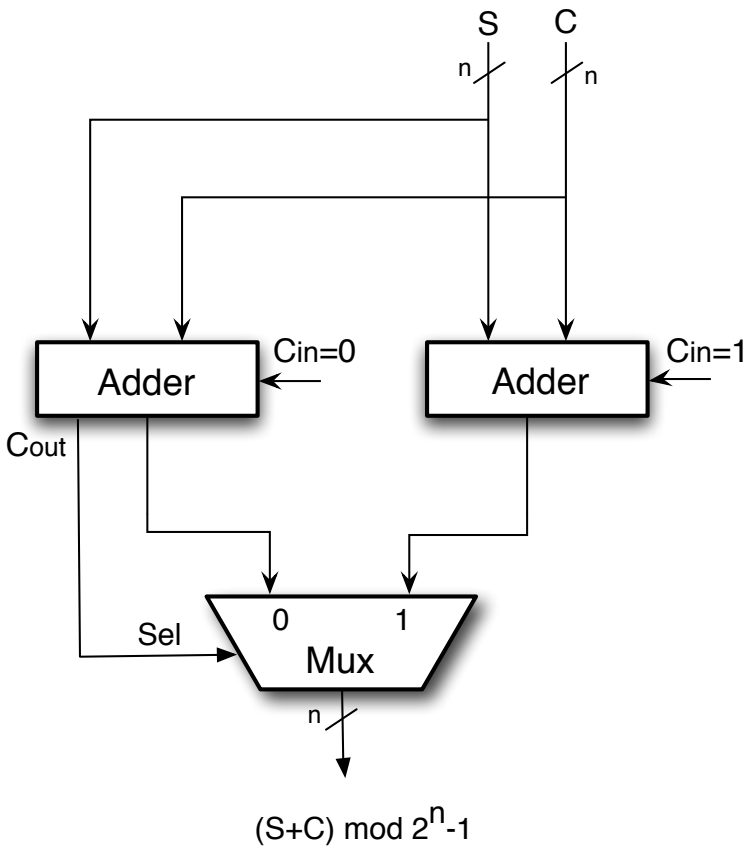

Figure 3.4: Modulo $2^n - 1$ adder

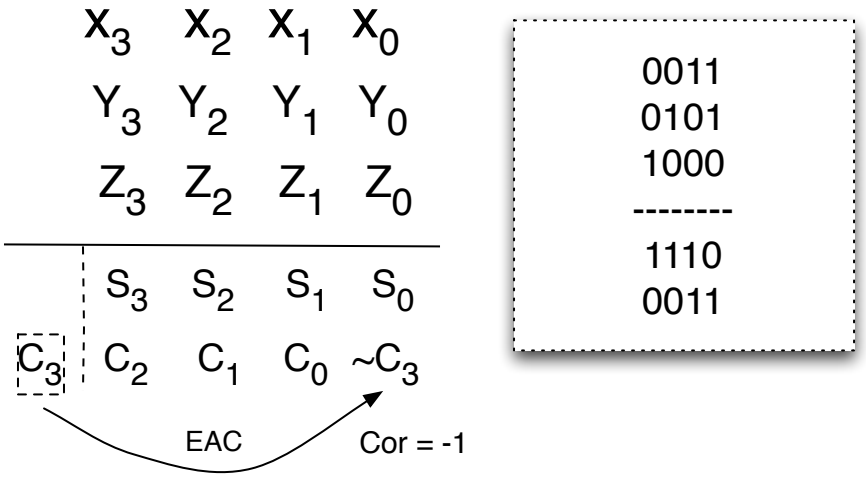

Figure 3.5: Example of a CSA mod $2^n + 1$ addition

A modulo $2^n + 1$ adder again requires two adders operating in parallel and a 2:1 mux to select the correct output. But the hardware requirements of modulo $2^n + 1$ adder in the implementation of FIR filters can be reduced by representing

output as intermediate sum. Instead of representing the output as the final modulo result, an intermediate result is generated by adding the sum and carry vectors from the CSA tree using conventional addition. As there are $n+1$ bits available to represent the result and S,C are of only n bits wide, $S+C \; mod \; (2^n+1)$ can always be represented as $S+C$. For example, consider $n=4$. The outputs of the CSA tree are of 4 bits wide and the final modulo is of 5 bits wide. Let $S=9$, $C=12$. Then according to 3.2,

$$S+C \; mod \; (2^n+1) = S+C-(2^n+1),$$

$$9+12 \; mod \; (2^4+1) = 9+12-17 = 4.$$

By using conventional addition, the result is $S+C=21$ which can be safely represented in 5 bits. This intermediate result still carry information about the final modulo result as $21 mod (2^4+1) = 4$. As there are further modulo operations in the modulo filters, this intermediate representation would not impact the filter output. The hardware architecture of the residue generator of modulo 2^n+1 is as shown in Figure 3.6

Modulo FIR Filters

Modulo FIR filters are realized in transpose form to reduce the critical path delay of the RNS filter. The output of each tap is represented in carry save (CS) form. CS representation avoid the computation of modular addition in each tap and computation of final moduli in each channel is carried out in the last stage. The only drawback in the CS representation of the accumulated result of each tap is increase in the number of registers required in each stage. Compared to conventional representation of accumulated result as final sum, CS representation requires double the number of registers to propagate both carry and save vectors to the next stage. The general architecture of a modulo filter is as shown in Figure 3.7.

The three different kinds of modulo filters are explained below:

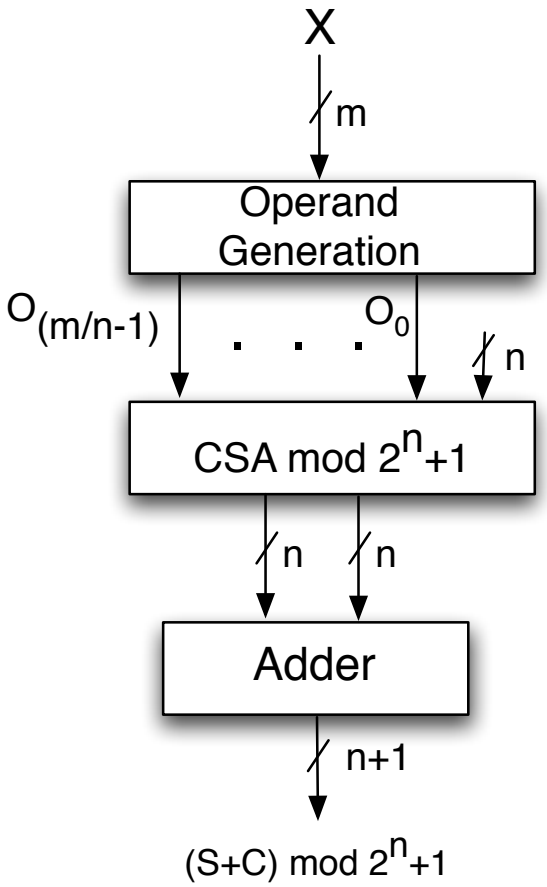

Figure 3.6: Modulo $2^n + 1$ adder

- Filter mod 2^k: If X is the input residue and Y is co-efficient of a filter tap, each of 4 bits wide then the partial product tree of the filter mod 2^k for $k = 4$ is as shown in Figure 3.8. The partial products are added using carry save adders with the carry out bit discarded as shown in Figure 3.9. The carry save representation of the multiplication result $(X * Y mod 2^k)$ is added with the carry save vectors from the previous stage using a 4:2 carry save adder modulo 2^k. A 4:2 CSA mod 2^k is as shown in Figure 3.10.

- Filter mod $2^n - 1$: Similarly if X and Y are the input residue and co-efficients of the filter mod $2^4 - 1$, the partial products of $(X * Y mod(2^4 - 1))$ is as shown in Figure 3.11. The ordering of partial products is based on the periodicity property

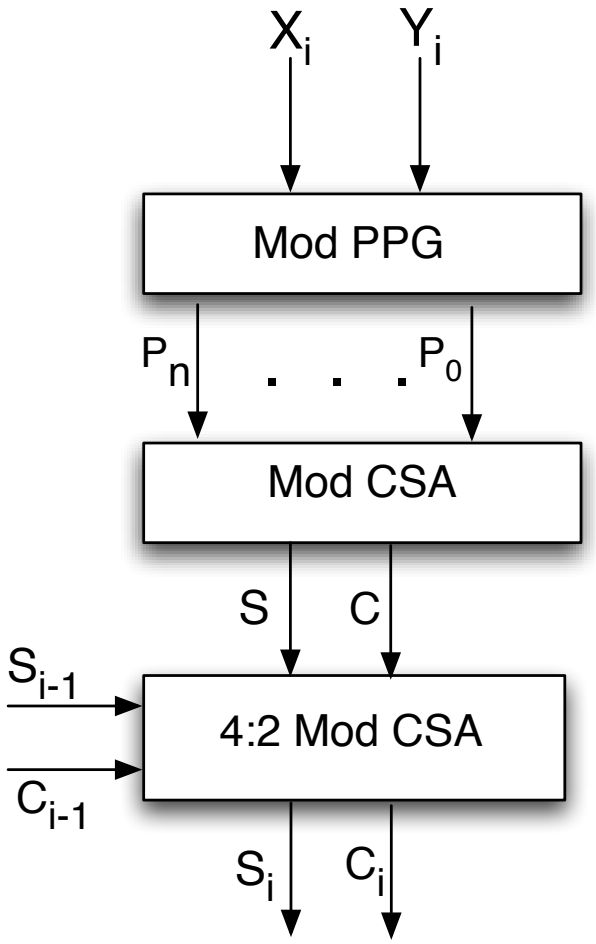

Figure 3.7: RNS modulo filter components

$$\begin{array}{cccc} x_0Y_3 & x_0Y_2 & x_0Y_1 & x_0Y_0 \\ x_1Y_2 & x_1Y_1 & x_1Y_0 & 0 \\ x_2Y_1 & x_2Y_0 & 0 & 0 \\ x_3Y_0 & 0 & 0 & 0 \end{array}$$

Figure 3.8: Partial product generation mod 2^4

of the moduli $2^n - 1$ as explained in the equation 3.3.

$$|2^i|_{2^n-1} = 2^{|i|_n} \tag{3.3}$$

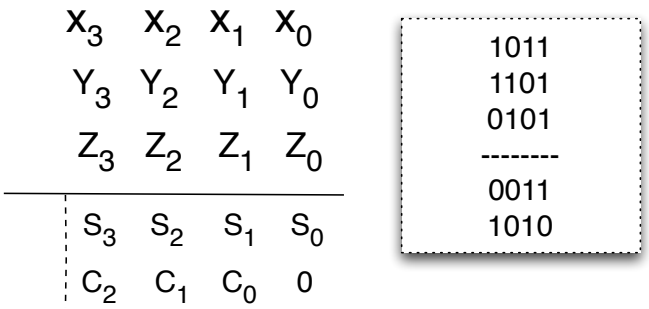

Figure 3.9: Carry save addition mod 2^4

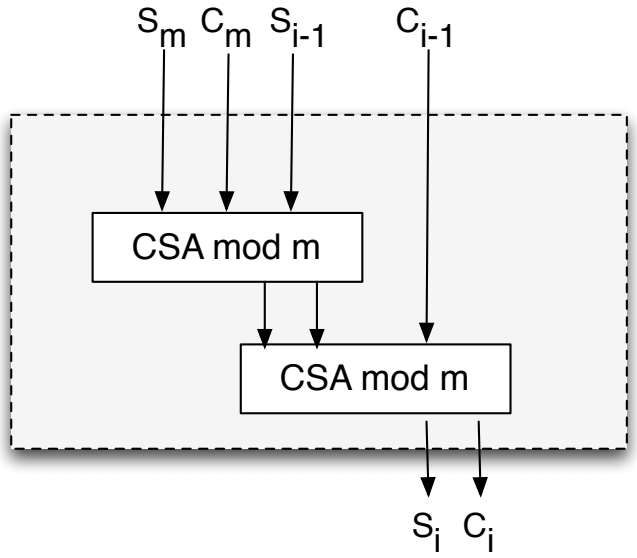

Figure 3.10: 4:2 Carry save accumulator

$$x_0Y_3 \quad x_0Y_2 \quad x_0Y_1 \quad x_0Y_0$$
$$x_1Y_2 \quad x_1Y_1 \quad x_1Y_0 \quad x_1Y_3$$
$$x_2Y_1 \quad x_2Y_0 \quad x_2Y_3 \quad x_2Y_2$$
$$x_3Y_0 \quad x_3Y_3 \quad x_3Y_2 \quad x_3Y_1$$

Figure 3.11: Partial product generation mod $2^4 - 1$

The partial products generated are added using carry save adders modulo $2^n - 1$. The carry save adder with end around carry is as shown in 3.2. The final carry and

sum vector result of the multiplication is added with the carry and save vectors from the previous stage using a 4:2 carry save adder modulo $2^n - 1$.

- Filter mod $2^n + 1$: The architecture of multiplication modulo $2^n + 1$ is as followed in [23]. The partial products if the input residues of $n + 1$ bits wide are arranged as vectors of n bits wide. The partial product generation for inputs of 5 bits wide is as shown in Figure 3.12. The carry save adder with end around carry for modulo $2^4 + 1$ is as shown in Figure 3.5. The arrangements of the partial

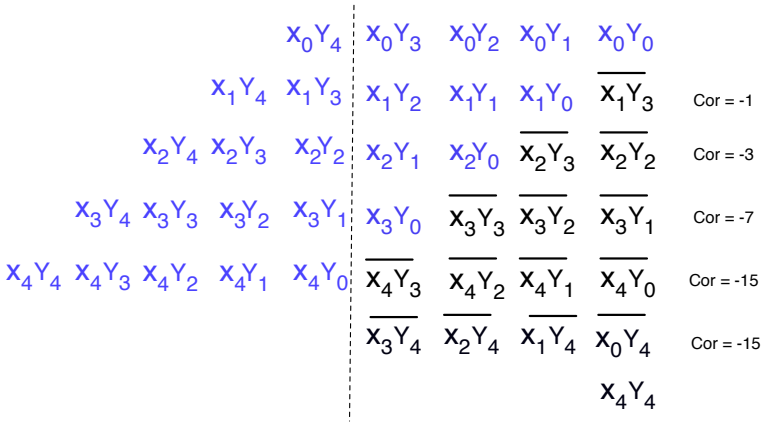

Figure 3.12: Partial product generation mod $2^4 + 1$

products is based on the periodicity property of moduli $2^n + 1$ as explained in the equation 3.4.

$$
\begin{aligned}
|2^i|_{2^n+1} &= 2^{|i|_{2n}}, 2j \leq i < (2j+1) \\
|2^i|_{2^n+1} &= -2^{|i|_n}, (2j-1) < i \leq 2j \\
j &> 1
\end{aligned}
\tag{3.4}
$$

The partial product vectors are added using carry save adders modulo $2^n + 1$. The carry save results of the multiplication are added to the carry save vectors of $n+1$ bits wide from the previous stage using a 4:2 CSA mod $2^n + 1$.

In this section the details of the Step 2 of the reverse converter design mentioned in chapter 2 is presented. The details of the derivation of this reverse converter can be found in the reverse converter design section in chapter 2. Fig. 3.13 shows the different components of the two moduli reverse converter. The two-moduli reverse converter

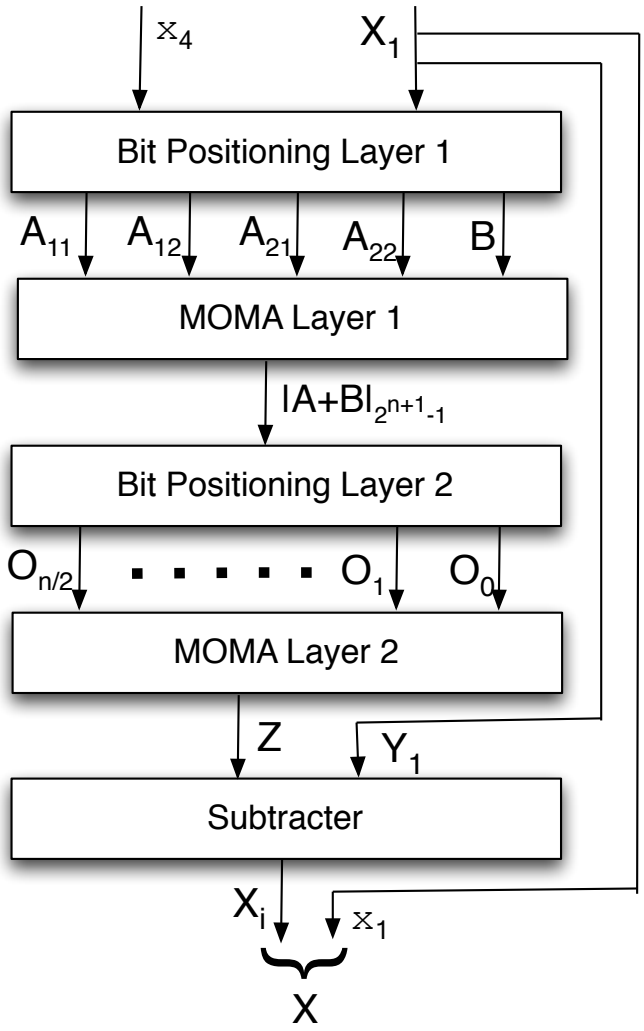

Figure 3.13: Hardware realization of two-reverse converter

consists of following components.

- Bit positioning layer 1: The inputs to the two-moduli reverse converter are X_1,

the partial reconstructed binary number from the three-moduli reverse converter of the moduli set $\{2^k, 2^n-1, 2^n+1\}$ and x_4, the residue of X with respect to the modulus $(2^{n+1}-1)$. X_1 is computed as [3]

$$X = 2^{2n}Y_1 + x_1, \tag{3.5}$$

where Y_1 is $2n$ bit wide intermediate result of the three-moduli reverse converter. The output of the bit positioning layer 1 are A_{11}, A_{12}, A_{21}, A_{22} and B, each is $n+1$ bits wide. The bit ordering of these outputs defined in derivation of reverse converter model are as follows.

$$A_{11} = \underbrace{x_{11,n-k'_{11}}, \cdots, x_{11,0}}_{n+1-k'_{11}} \underbrace{x_{11,n}, \cdots, x_{11,n-k'_{11}+1}}_{k'_{11}}, \tag{3.6}$$

$$A_{12} = \underbrace{x_{12,n-k'}, \cdots, x_0}_{n+1-k'} \underbrace{x_{12,n}, \cdots, x_{12,n-k'+1}}_{k'}, \text{ and} \tag{3.7}$$

$$k'_{11} = |k-n+k'|_{n+1}, \tag{3.8}$$

$$A_{21} = \underbrace{Y_{11,n-k'_{21}}, \cdots, Y_{11,0}}_{n+1-k'_{21}} \underbrace{Y_{11,n}, \cdots, Y_{11,n-k'_{21}+1}}_{k'_{21}}, \tag{3.9}$$

$$A_{22} = \underbrace{Y_{12,n-k'_{22}}, \cdots, Y_{11,0}}_{n+1-k'_{22}} \underbrace{Y_{12,n}, \cdots, Y_{12,n-k'_{22}+1}}_{k'_{22}}, \tag{3.10}$$

$$k'_{21} = |n+1+k+k'|_{n+1}, \tag{3.11}$$

$$k'_{22} = |k+k'|_{n+1}, \text{and} \tag{3.12}$$

$$B = \left|-x_4 2^{k'}\right|_{2^{n+1}-1} = \underbrace{\bar{x}_{4,n-k'}, \cdots, \bar{x}_0}_{n+1-k'} \underbrace{\bar{x}_{4,n}, \cdots, \bar{x}_{4,n-k'+1}}_{k'}. \tag{3.13}$$

Computation of B requires $n+1$ inverters and no additional hardware is required for the ordering of the bits.

- Multi-operand modulo addition (MOMA) layer 1: A MOMA [11] is basically a modulo adder, but can be doubled as a compressor as the output bit width is fixed by the modulo operation irrespective of the number of operands. The MOMA

used here takes five input vectors A_{11}, A_{12}, A_{21}, A_{22} and B of $n+1$ bits and added using carry save adders (CSA) with end-around carry (EAC). The *carry* and *save* bits of the output of the adder are added with a modulo $(2^{n+1}-1)$ adder to produce the output denoted by $|A+B|_{2^{n+1}-1}$, which is $n+1$ bit wide. This MOMA layer requires $4(n+1)$ full adders (FA), including the $(n+1)$ FAs required by the $2^{n+1}-1$ modulo adder.

- Bit positioning layer 2: This layer is same as the bit positioning layer 1, except that this layer generates $(n/2+1)$ operands, O_0, O_1, ..., $O_{n/2}$, each of $n+1$ bits obtained from ordering of the bits of $|A+B|_{2^{n+1}-1}$, from the output of the MOMA layer 1. The details of the ordering the bits can be found in the derivation of reverse converter model in chapter 2. Unlike the bit positioning layer 1, this layer does not require any gates.

- MOMA layer 2: This layer is similar to MOMA layer 1, except that the number of inputs are $(n/2+1)$, viz. O_0, O_1, ..., $O_{n/2}$. The total number of FAs required to implement this MOMA is $(n/2)(n+1)$, including $(n+1)$ FAs of the modulo $(2^{n+1}-1)$ adder. The output of this layer is $n+1$ bit and is denoted by Z.

- Subtracter: Here the output Z of MOMA layer 2 is left shifted by $2n$ bits and added to Y_1. The resultant $3n+1$ bit vector is as shown below

$$\underbrace{Z_n,\cdots,Z_1,Z_0}_{n+1},\overbrace{Y_{1,2n-1},\cdots,Y_{1,1},Y_{1,0}}^{2n}$$

  Z is subtracted from this result to generate the intermediate result X_i, which is left shifted by k bits and added to the residue x_1 (k bit wide) to get the final result X.

$$X = x_1 + Y_1 2^k + 2^k(2^{2n}-1)Z$$

  The $3n+1$ bit subtracter is implemented as a 2's complement adder and requires $(3n+1)$ FAs.

Chapter 4

Experimental Results

4.1 Performance of MAC units

In this section, the advantages of the proposed 4-moduli set k-mod4 moduli set over the Cao-mod4 moduli set is illustrated by implementing modulo MAC units. Area, delay and power consumption of an RNS filter is usually dominated by the MACs as the conversion overhead of the forward and the reverse converter remains constant, while the number of MACs increases linearly with the order of the filter. Hence the measurements of the area and the power of the MAC units alone is considered to compare the advantages of the k-mod4 moduli set with the Cao-mod4 moduli set.

An RNS MAC unit consists of modulo MACs for each moduli in the moduli set. Of all the modulo MACs, the modulo MAC for the modulus $2^n + 1$ has the longest delay [3]. There have been several implementations to minimize the delay of the MAC associated with the $2^n + 1$ channel. Of these [23] was found to be efficient and it is used in the RNS filter implementations. MACs for moduli 2^n and 2^k are implemented as conventional binary MACs with the MSB bits greater than 2^n and 2^k discarded respectively. MACs for moduli $(2^n - 1)$ and $(2^{n+1} - 1)$ are implemented as conventional binary MACs with end around carry added to the LSB.

In the experiments, the area and the power of the modulo MACs for the dynamic ranges 9–105 are compared. Selection of n and k for the moduli sets $\{2^{n1}, 2^{n1} - 1, 2^{n1} + 1, 2^{n1+1} - 1\}$ (Cao-mod4) and $\{2^k, 2^{n2} - 1, 2^{n2} + 1, 2^{n2+1} - 1\}$ (k-mod4) are shown in Table **??**. Due to the absence of the programmable k, Cao-mod4 uses higher n, while the k-mod4 moduli set can be programmed to cover the intermediate dynamic range with smaller n as can be seen from the table. However, k cannot be increased arbitrarily as it is upper bounded by the critical path of the $2^n + 1$ channel (*critical path condition*), which cannot exceed the delay of the binary channel 2^k.

Dynamic Range	$n1$	$n2$	k
9	2	2	2
10	4	2	3
11	4	2	4
12	4	4	4
13	4	4	4
14	4	4	4
15	4	4	4
16	4	4	4
17	4	4	4
18	6	4	5
19	6	4	6
20	6	4	7
21	6	4	8
22	6	6	6
23	6	6	6
24	6	6	6
25	6	6	6
26	8	6	7
27	8	6	8
28	8	6	9
29	8	6	10
30	8	6	11
31	8	6	12
32	8	8	8

Dynamic range	$n1$	$n2$	k
55	14	12	18
56	14	12	19
57	14	14	14
58	16	14	14
59	16	14	15
60	16	14	16
61	16	14	17
62	16	14	18
63	16	14	19
64	16	14	20
65	16	16	16
66	18	16	17
67	18	16	18
68	18	16	19
69	18	16	20
70	18	16	21
71	18	16	22
72	18	16	23
73	18	18	18
74	20	18	18
75	20	18	19
76	20	18	20
77	20	18	21
78	20	18	22

Table 4.1: Dynamic ranges used in the experiments

Dynamic Range	$n1$	$n2$	k
33	8	8	8
34	10	8	9
35	10	8	10
36	10	8	11
37	10	8	12
38	10	8	13
39	10	8	14
40	10	8	15
41	10	10	10
42	12	10	11
43	12	10	12
44	12	10	13
45	12	10	14
46	12	10	15
47	12	10	16
48	12	10	17
49	12	12	12
50	14	12	13
51	14	12	14
52	14	12	15
53	14	12	16
54	14	12	17

Dynamic range	$n1$	$n2$	k
79	20	18	23
80	20	18	24
81	20	20	20
82	22	20	21
83	22	20	22
84	22	20	23
85	22	20	24
86	22	20	25
87	22	20	26
88	22	20	27
89	22	22	22
90	24	22	23
91	24	22	24
92	24	22	25
93	24	22	26
94	24	22	27
95	24	22	28
96	24	22	29
97	24	24	24
98	26	24	25
99	26	24	26
100	26	24	27

Table 4.2: Dynamic ranges used in the experiments

For the experiments, the modulo MACs are implemented in RTL and synthesized using a 65nm standard cell library. The area and the power comparisons of the MAC units using the k-mod4 and the Cao-mod4 moduli sets synthesized at the maximum frequency of 200 MHz and 500 MHz are presented in Figs. 4.1 and 4.2, and in Figs. 4.3 and 4.4 respectively. Improvement in area and power reduction is observed for all dynamic ranges. From the plots, area reduction as much as 46% and power reduction as much as 43% are observed.

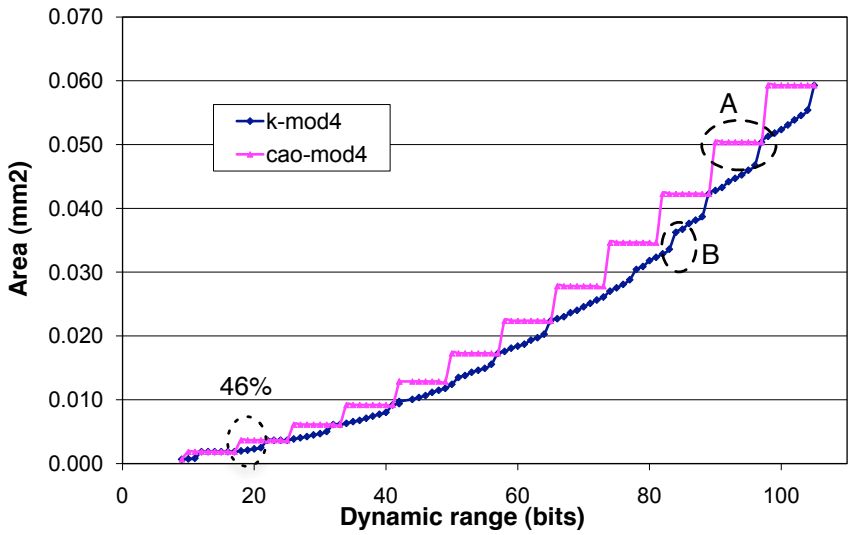

Figure 4.1: Comparisons of area of modular MACs for k-mod4 and Cao-mod4 synthesized at 200MHz

It can be seen that the synthesized area of MACs using Cao-mod4 moduli set for the dynamic range spanned between two consecutive ns i.e, n and $n+2$ remains constant (Annotated portion A in Fig. 4.1). This is due to the absence of a programmable moduli set, which can customize the hardware corresponding to the required dynamic range. For example the dynamic range of the Cao-mod4 moduli set, when $n = 4$ is $4n+1 = 17$bits. But for the next immediate dynamic range of 18 bits, $n = 4$ is not sufficient and the next available value is $n = 6$. However in case of k-mod4 moduli set, to achieve dynamic range of 18 bits, k can be programmed to $k = 5$, with $n = 4$.

Another observation from the area and the power plots for the k-mod4 moduli

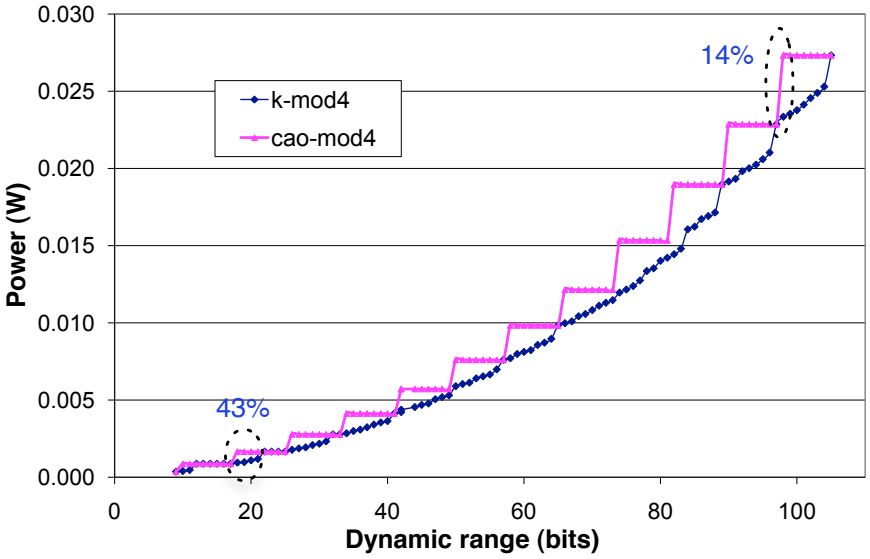

Figure 4.2: Comparisons of power of modular MACs for k-mod4 and Cao-mod4 synthesized at 200MHz

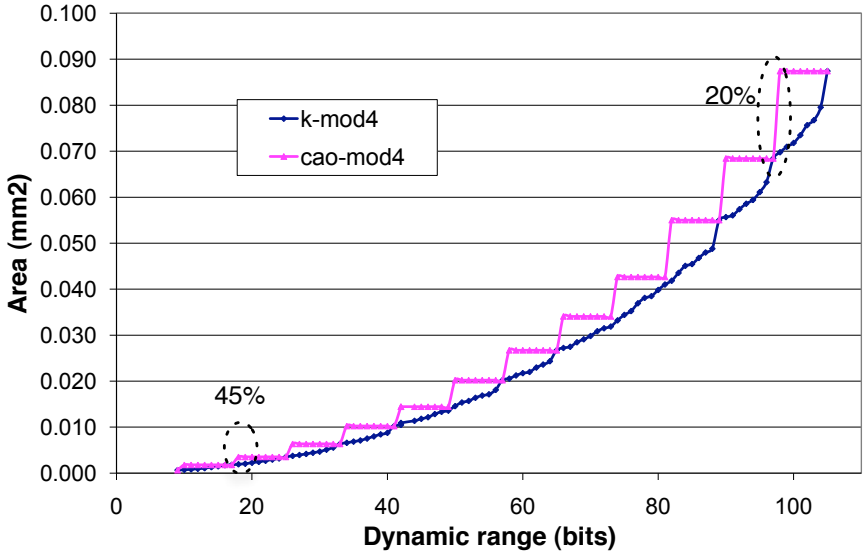

Figure 4.3: Comparisons of area of modular MACs for k-mod4 and Cao-mod4 synthesized at 500MHz

set is that there are sudden jumps in the area and the power at certain dynamic ranges (Annotated portion B in Fig. 4.1). This is due to the *critical path condition* , that disallows k from increasing beyond a certain value, and forces to choose the next higher n.

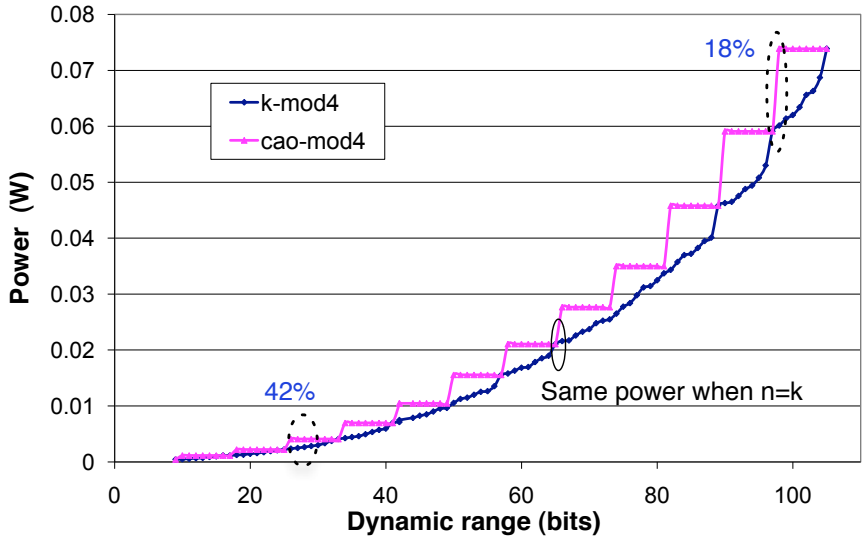

Figure 4.4: Comparisons of power of modular MACs for k-mod4 and Cao-mod4 synthesized at 500MHz

Compared to Cao-mod4 moduli set, k-mod4 moduli set gives maximum improvements for dynamic ranges of the form $4n+2$. While the dynamic range of Cao-mod moduli set is $4n+1$, to achieve the next dynamic range, the next available value of n has to be chosen. The area and power improvements of k-mod4 moduli set for such dynamic ranges is listed in tables Table 4.3, Table 4.4 and Table 4.5, Table 4.6 for the MACs synthesized at 200MHz and 500MHz respectively.

Dynamic range	Cao-mod4	k-mod4	%Improvement
10	1844.00	720.00	60.95
18	3643.00	1966.00	46.03
26	6104.00	3840.00	37.09
34	9179.00	6290.00	31.47
42	12857.00	9376.00	27.07
50	17269.00	12410.00	28.14
58	22375.00	17569.00	21.48
66	27794.00	22695.00	18.35
74	34627.00	27014.00	21.99
82	42277.00	32867.00	22.26
90	50364.00	42795.00	15.03
98	59285.00	51283.00	13.50

Table 4.3: Maximum Area (um2) Improvements at 200MHz

Dynamic range	Cao-mod4	k-mod4	%Improvement
10	0.871	0.397	54.42
18	1.656	0.943	43.08
26	2.775	1.772	36.16
34	4.128	2.840	31.21
42	5.716	4.217	26.23
50	7.606	5.892	22.53
58	9.834	7.702	21.68
66	12.147	9.973	17.90
74	15.340	11.957	22.05
82	18.960	14.443	23.82
90	22.859	19.154	16.21
98	27.325	23.347	14.56

Table 4.4: Maximum Area (um2) Improvements at 500MHz

Dynamic range	Cao-mod4	k-mod4	%Improvement
10	1793.000	721.000	59.79
18	3526.000	1917.000	45.63
26	6354.000	3742.000	41.11
34	10241.000	6590.000	35.65
42	14448.000	10485.000	27.43
50	20212.000	14552.000	28.00
58	26714.000	20544.000	23.10
66	34077.000	27207.000	20.16
74	42659.000	33201.000	22.17
82	55011.000	41832.000	23.96
90	68419.000	55652.000	18.66
98	87410.000	69753.000	20.20

Table 4.5: Maximum power (mW) Improvements at 200MHz

Dynamic range	Cao-mod4	k-mod4	%Improvement
10	1.143	0.48835	57.27
18	2.2066	1.2334	44.10
26	4.0892	2.3703	42.04
34	6.9681	4.2489	39.02
42	10.468	7.128	31.91
50	15.564	10.543	32.26
58	21.068	15.8	25.00
66	27.666	21.592	21.95
74	35.006	26.51	24.27
82	45.809	34.334	25.05
90	59.098	46.26	21.72
98	73.871	60.131	18.60

Table 4.6: Maximum power (mW) Improvements at 500MHz

4.2 Performance of Reverse Converter

In this section, the hardware complexity and the delay of the proposed k-mod4 reverse converter is compared with the existing 4-moduli reverse converters. Cao-mod4 moduli set $\{2^n, 2^n-1, 2^n+1, 2^{n+1}-1\}$ [2] has been chosen for comparison with the proposed reverse converter as it is the most balanced among the existing 4-moduli sets.

In the proposed k-mod4 moduli set $\{2^k, 2^n-1, 2^n+1\}$, if $k=n$, then the hardware complexity and the delay of k-mod4 reverse converter is identical to the Cao-mod4 reverse converter. Hence it is assumed that $k > n$ for comparison in this section.

Recall that reverse converter design of a 4-moduli set consists of two stages as mentioned in design of reverse converter section. In the first stage, the residues are processed through a 3-moduli set. For $k > n$, k-mod4 reverse converter will contain an additional CSA layer of $2n$ FAs over Cao-mod4 in the first stage. Also, k-mod4 incurs an additional FA delay over Cao-mod4 in stage 1. Similarly, in the second stage, the reverse converter for the two-moduli set $\{2^k(2^{2n}-1), 2^{n+1}-1\}$ requires an additional CSA layer of $n+1$ FAs and incurs extra FA delay over the reverse converter of the two-moduli set $\{2^n(2^{2n}-1), 2^{n+1}-1\}$.

Table 4.7 shows the detailed comparison of area, delay of the reverse converters for the proposed k-mod4 and the Cao-mod4 reverse converters. Note that t_{INV}, t_{MUX} and t_{FA} denote the gate delays of inverter, MUX and full adder respectively. l is the number of stages in the $n/2+1$ CSA tree.

The proposed reverse converter and the four-stage reverse converter for the Cao-mod4 moduli set [2] are synthesized using a 65nm standard cell library. The designs are optimized for minimum delay. The minimum delay of the reverse converters for different dynamic ranges are compared in Fig. 4.5 and their corresponding areas are compared in Fig. 4.6. The synthesis results show that for a given dynamic range, our

Gates	k-mod4 RC	Cao-mod4 RC
INV	$2n+k+2$	$3n+2$
HA	0	1
FA	$n^2/2+27n/2+2$	$n^2/2+21n/2+4$
MUX	0	2
Delay	$t_{INV}+$ $(11n+10+l)t_{FA}$	$t_{INV}+t_{MUX}$ $+(11n+8+l)t_{FA}$
Dyn. range	$3n+k+1$	$4n+1$

Table 4.7: Area and delay comparison of 4-moduli sets

k-mod4 reverse converter has 23% less delay and 54% less area. Note that the reverse converter implementations of both the moduli sets are same when $k = n$.

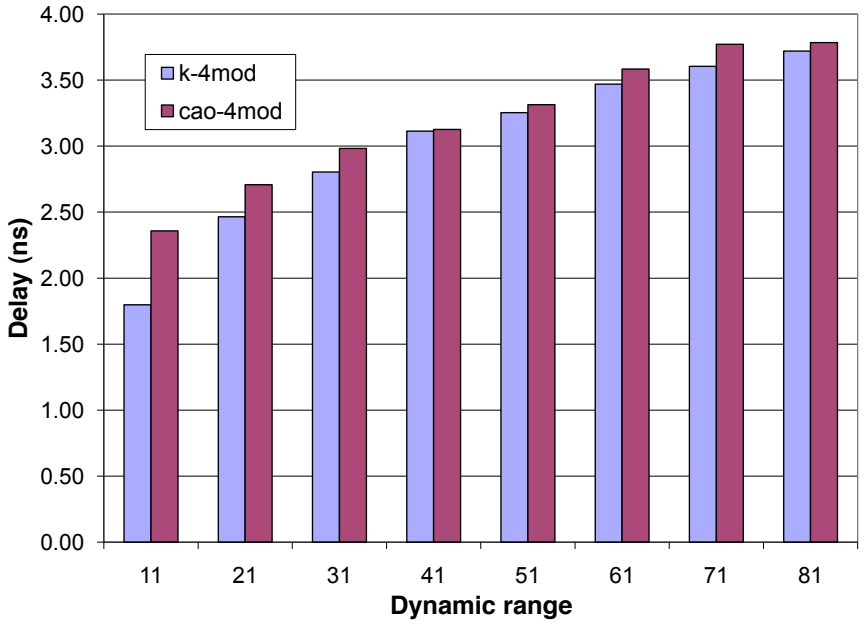

Figure 4.5: Delay comparison of reverse converter

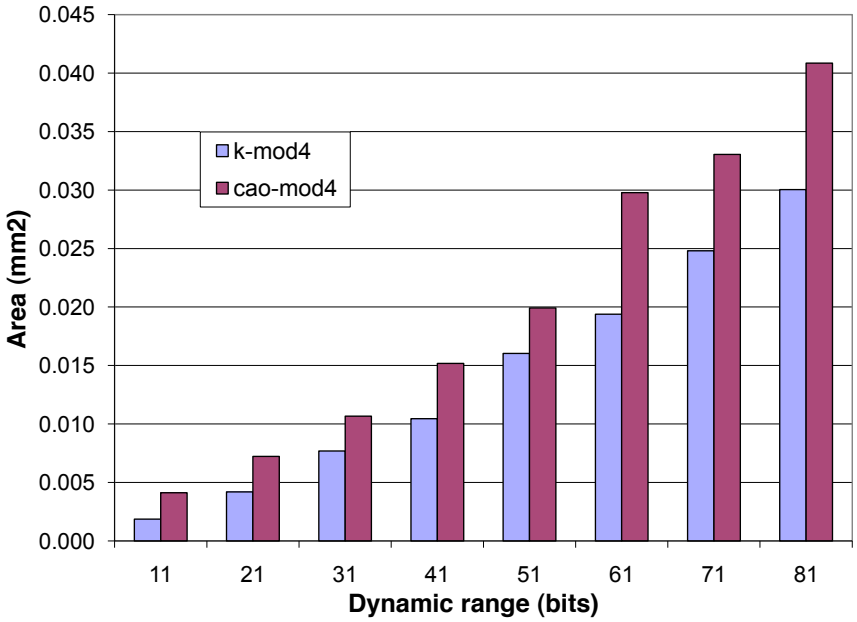

Figure 4.6: Area comparison of reverse converter

4.3 Performance of Filter

In this section, RNS filters implemented using Cao-mod4 moduli set and the proposed k-mod4 moduli set are compared for chip area and power. The specifications of the different types of filter- Butterworth, Elliptical, Least Square, Park Mc-Clennan filters as used in reference [7] are shown in Table 4.8. In the table, Fs represents the stop band frequency and Fp represents the pass band frequency.

Filter Type	Order	Fs	Fp
Butterworth	20	0.25	0.3
Elliptical	6	0.25	0.3
Least Square	41	0.25	0.3
Park Mc-Clennan	28	0.25	0.3
Butterworth	71	0.27	0.2875
Elliptical	8	0.27	0.2875
Least Square	172	0.27	0.2875
Park Mc-Clennan	119	0.27	0.2875
Elliptical	13	0.27	0.29
Least Square	326	0.27	0.29
Park Mc-Clennan	189	0.27	0.29

Table 4.8: Different Filter Specifications [7]

For this experiment, filter of input width 24 bits and with the following specifications is chosen:

- Filter type : Elliptical
- Stop band frequency : 0.3
- Pass band frequency : 0.25
- Pass band ripple : 3dB
- Stop band ripple : -50dB
- Order : 6

The dynamic range of the filter with 24 bit input width and 6 taps is $(2*24+log_2 6)$=51 bits. To implement this RNS filter of 51 bit dynamic range, using Cao-mod4 moduli set, $n = 14$ has to be chosen according to the Table 4.2. Using the k-mod4 moduli set, the parameters of n,k chosen to satisfy the required dynamic range are $n = 12$ and $k = 14$ (from Table 4.2). Therefore the moduli sets chosen for Cao-mod4 and k-mod4 are $\{2^{14}, 2^{14}-1, 2^{14}+1, 2^{15}-1\}$ and $\{2^{14}, 2^{12}-1, 2^{12}+1, 2^{13}-1\}$ respectively.

Two filters using Cao-mod4 moduli set and k-mod4 moduli set are implemented in RTL based on the RNS filter architecture as represented in Figure 3.1. The filters implemented in verilog are simulated for functionality checking and synthesized under no delay constraints. This analysis gives the total negative slack of the circuit which denotes the minimum clock period required to synthesize the circuits. The following table 4.9 shows the maximum frequency of operation and the cell area corresponding to zero delay synthesis of the two filters.

Parameter	Cao-mod4	k-mod4	%Improvement
Gate count	13812	11279	18.3
Cell area(um2)	78829	64188	18.5
Net area(um2)	61967	50025	19.2
Total area(um2)	140796	114213	18.8
Delay (ns)	4.13	4.07	1.4
ADP (mm2*ns)	0.573	0.471	17.6

Table 4.9: Comparison of delay and area of k-mod4 and cao-mod4 filters

From the synthesis results, it can be seen that there is significant improvement in cell area but the performance is comparable for both the designs. The synthesis tool tries to improve the performance of the design boosting the circuit area to reduce the total negative slack. Hence ADP is compared for the two designs which gives accurate estimate of the performance for designs synthesized using zero delay.

After knowing the maximum frequency of operation, the two filters are synthesized and placed and routed for the same clock frequency of 200Mhz. At this frequency,

both the designs meet timing at post-synthesis as well as post-place and route levels. Table 4.10 compares the post-place and route performance parameters of the two designs.

Parameter	Cao-mod4	k-mod4	%Improvement
Gate count	20875	16485	21.0
Power(mW)	17.3	14.0	19.0
Cell area(um2)	92870	74167	20.1
Chip Area(mm2)	0.20	0.16	20.6

Table 4.10: Area and Power improvements of k-mod4 moduli set

The reverse converter used in the filter designs are implemented as single stage and two stage pipeline. The frequency of operation of the filters with single stage RC is 344MHz and with two stage RC is 250MHz. Table 4.11 and Table 4.12 compare the post-place and route performance parameters of the filters using single stage RC and two stage RC.

Parameter	Cao-mod4	k-mod4	%Improvement
Gate count	21493	16857	21.5
Power(mW)	22.1	17.0	19.7
Cell area(um2)	92364	73551	20.4
Chip Area(mm2)	0.155	0.124	19.4

Table 4.11: Comparison of filters with single stage RC

Parameter	Cao-mod4	k-mod4	%Improvement
Gate count	60079	46204	23.0
Power(mW)	31.8	23.5	26.1
Cell area(um2)	93724	72079	23.1
Chip Area(mm2)	0.153	0.123	19.6

Table 4.12: Comparison of filters with two stage RC

The following figures Figure 4.7 and Figure 4.8 show the chip layouts of the filters implemented using the Cao-mod4 moduli set and k-mod4 moduli set respectively using two stage RC. The annotated values are the width and height of the chip. The two filters are place and routed for the same initial densities and frequencies.

The reduction in chip area and post-pnr power with the proposed moduli set is 20%. Power estimation was done using PrimeTime Px providing VCD (vector change dump) file as input to estimate the realistic switching activities of the primary inputs and internal nets.

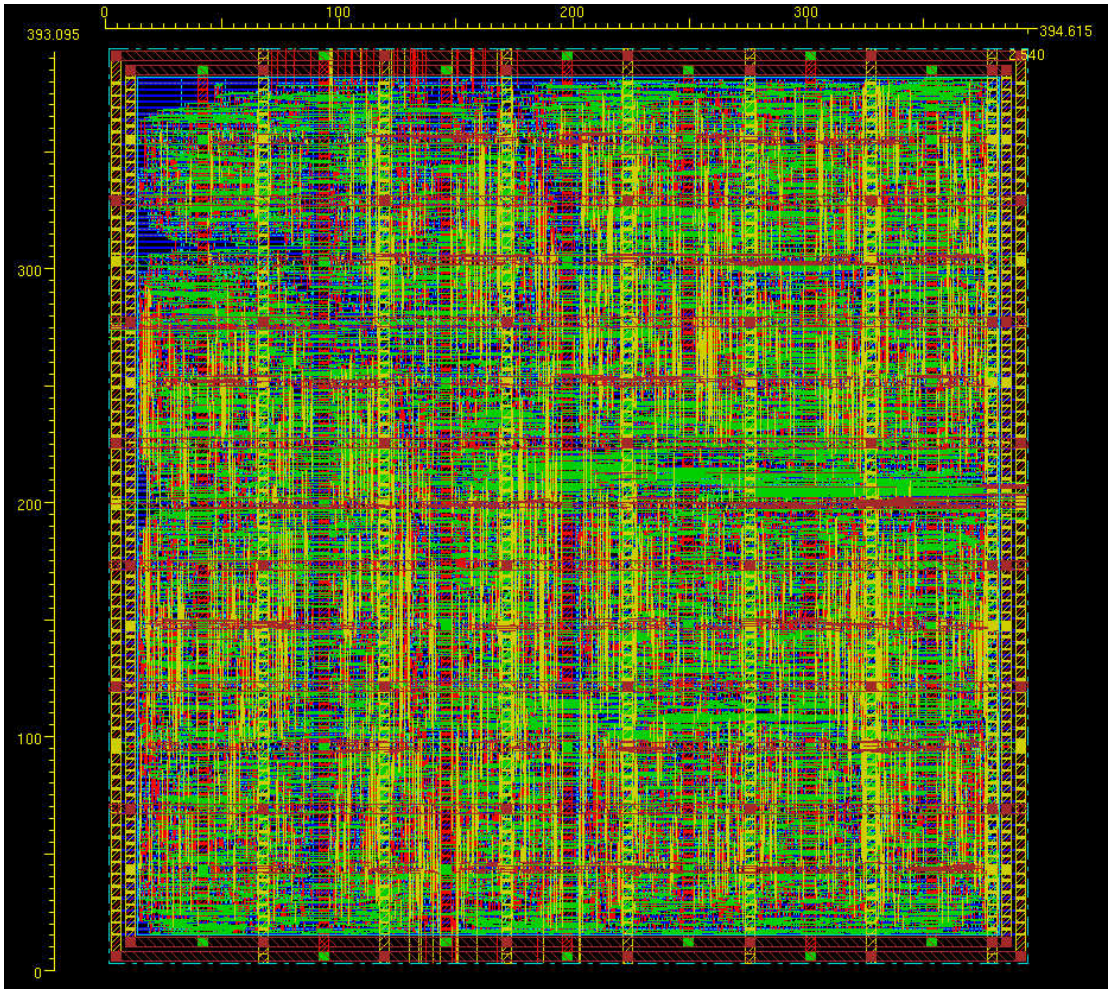

Figure 4.7: Layout of the Filter using Cao-mod4 moduli set

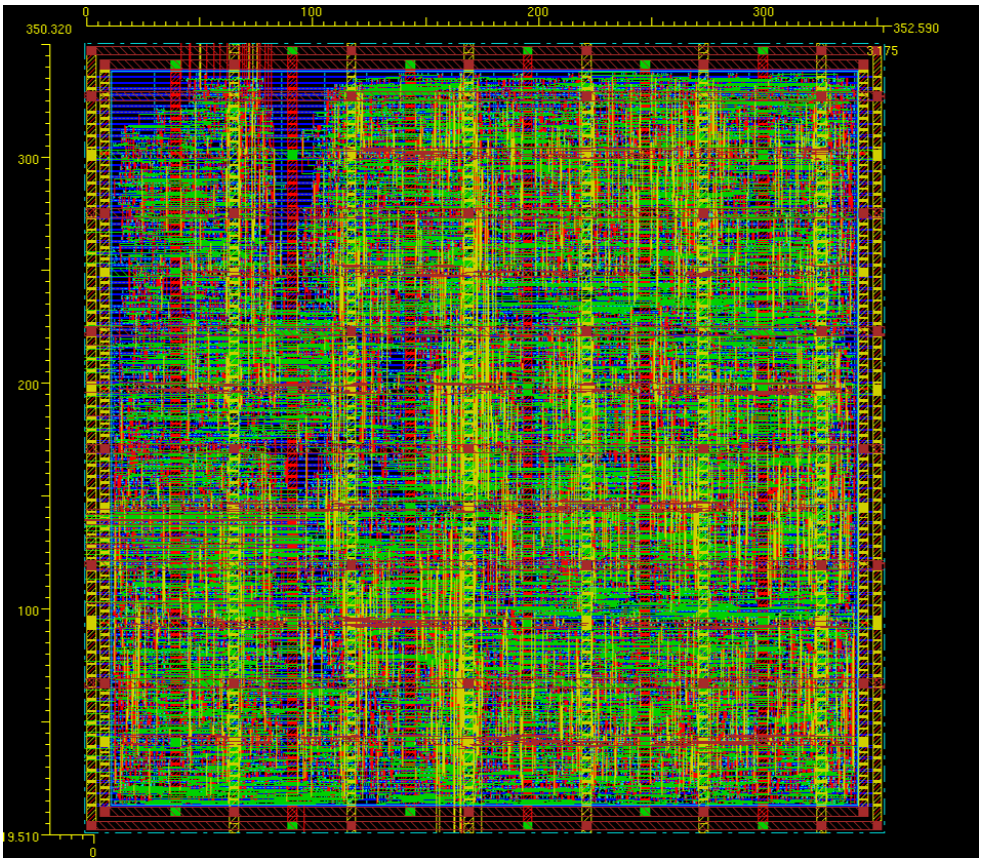

Figure 4.8: Layout of the Filter using k-mod4 moduli set

Chapter 5

Application of threshold logic

5.1 RC design using threshold logic

Reverse converter is the critical path unit in the filter. Pipelining is one alternative to speed up the reverse conversion but it costs in additional registers. To speed up the reverse conversion without sacrificing the power, it is implemented in threshold logic.

Threshold logic is an alternate logic style to CMOS that achieves better performance with lower power. The standard cell in threshold logic in called threshold logic latch (TLL). It is a clocked differential sense amplifier based circuit. TLL can implement threshold functions up to fan in of 13. A threshold function is defined as a boolean function $y = f(x_1, x_2, \cdots, x_n)$ if it can be represented as:

$$y = \begin{cases} 1, & \sum_{i=0}^{n-1} x_i w_i \geq T, \\ 0, & \text{otherwise} \end{cases} \tag{5.1}$$

where, w_i is the weight of the input x_i and T is the threshold. The circuit implementation of threshold logic latch [16] is shown in figure 5.1. The critical aspect in designing a circuit using TLL is identification of threshold functions in the circuit. The reverse converter comprises of CSA and modulo adders. A CSA tree can be replaced by set of counters. Counter function is a threshold function and can be implemented by few TLL gates. In the two stage implementation of reverse converter, a 5-input CSA tree feeding D flipflops shown in figure 5.2 is replaced by 5-input counter implemented using threshold gates shown in figure 5.3. Here the logic depth of 3-FAs and DFF is reduced to a single TLL gate depth. The delay of 6 XOR gates and the setup time of DFF is replaced by setup time of a TLL gate. This absorption in logic depth results in larger speed up at the cost of larger number of sequential gates.

The truth table of 5 bit counter is shown in table5.1. y_2, y_1, and y_0 represent the output of the 5-bit counter with number of 1's in the input bits represented as Cnt.

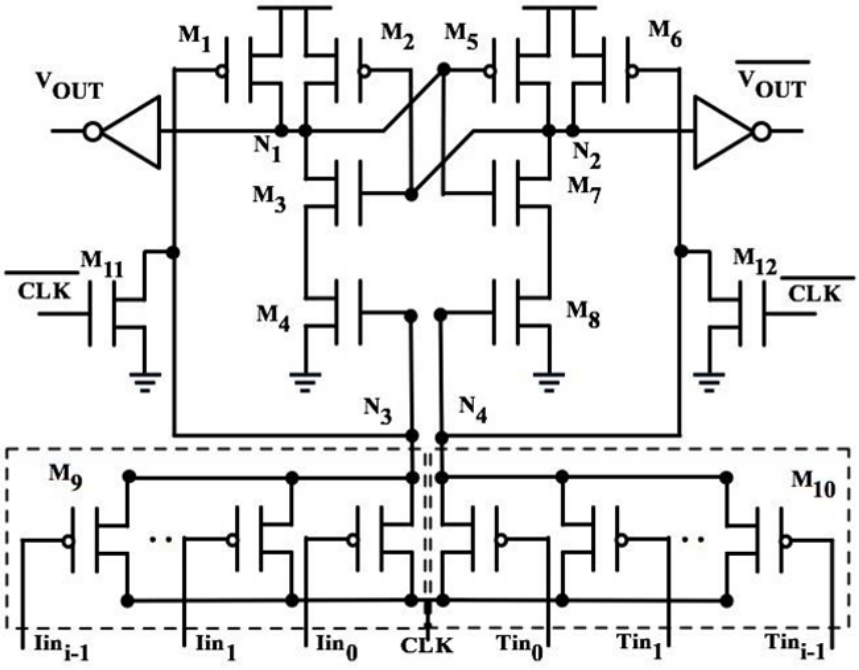

Figure 5.1: Threshold logic latch

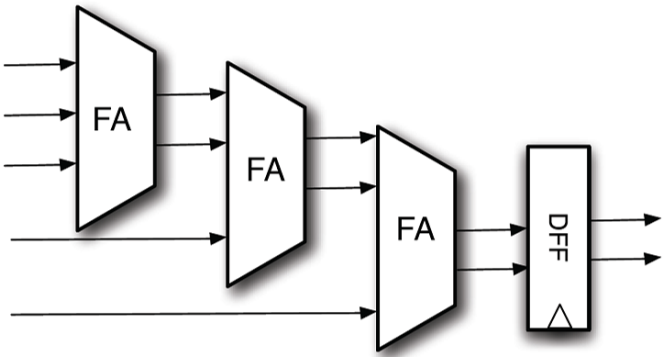

Figure 5.2: 5-input counter

From the truth table:

$y_0 = 1$, if (cnt ≥ 1 and cnt < 2) or (cnt ≥ 3 and cnt < 4) or (cnt ≥ 5),

$y_1 = 1$, if (cnt ≥ 2 and cnt < 4),

$y_2 = 1$, if (cnt ≥ 4).

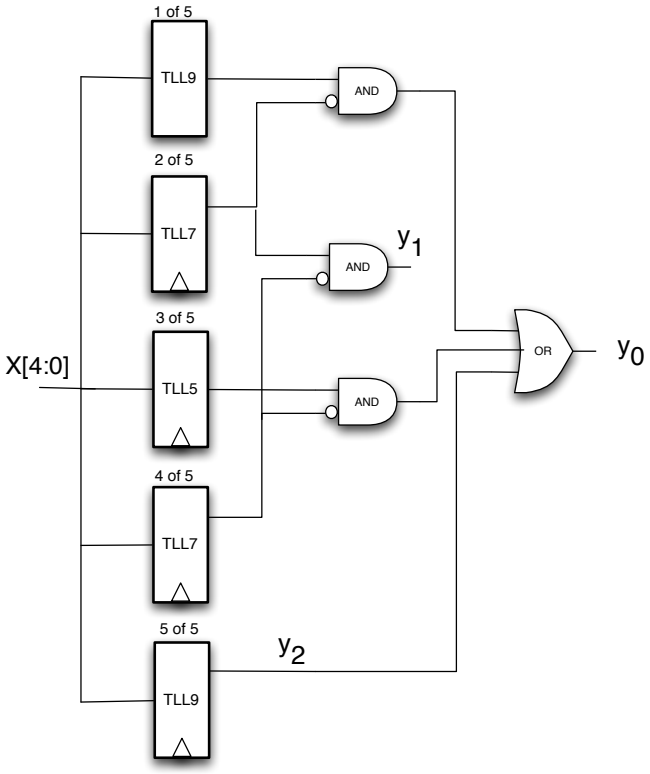

Figure 5.3: 5-input TLL counter

y2	y1	y0	Cnt
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5

Table 5.1: Truth table of 5-input counter

The output of the TLL counter is in the form of three bits and it is reduced to two bits using a CSA.

By identifying the 5-input counters in the reverse converter design, a hybrid implementation of the two stage reverse converter is built using TLL and CMOS gates. The CMOS and hybrid form of reverse converter of k-mod4 moduli set (n=8,k=10) in two stage is implemented in RTL and place and routed. Figure 5.4 and figure 5.5 show the area and power improvements of the hybrid reverse converter at different

frequencies. The TLL design provides faster design with smaller power. At the

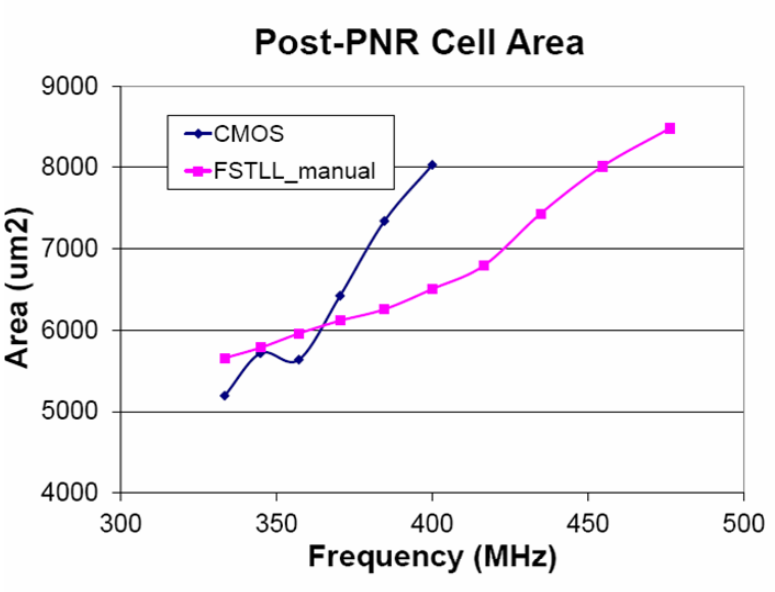

Figure 5.4: Area improvements of TLL over CMOS RC

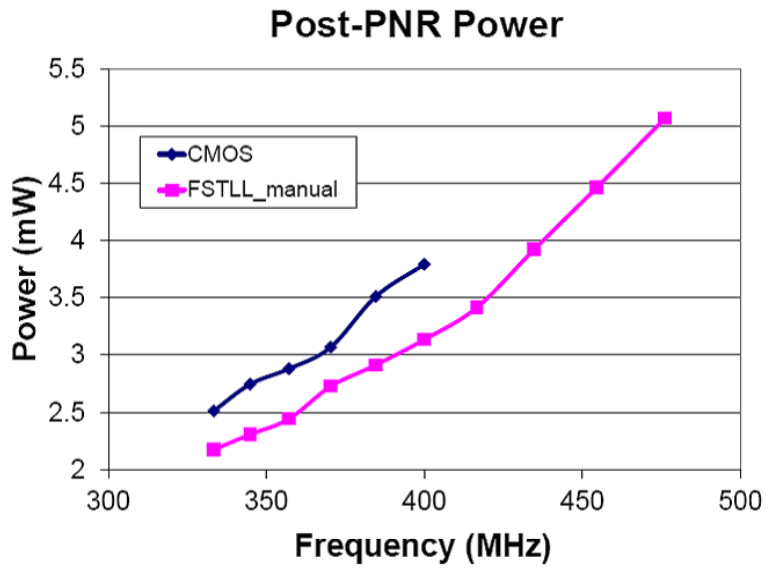

Figure 5.5: Power improvements of TLL over CMOS RC

closing frequency of CMOS design of 400MHz, the area and power improvements of TLL designs are 17.3% and 18.8% respectively. While for the same power consumption of 3.7mW, the TLL design can run 20% faster than the CMOS design.

5.2 Experimental Setup

The EDA tools used in the experiments are listed below:

- Verilog Simulator : Cadence NCverilog
- Synthesis tool : Cadence RC
- Timing Analysis tool : Synopsys PrimeTime
- Place and Route tool : Cadence Encounter
- Static power analysis : Synopsys PrimeTime PX

Chapter 6

Conclusions

The thesis is a study of application of residue number system in the implementation of FIR filters. The key contributions of the thesis are:

- It proposes a new RNS 4-moduli set for efficient implementation of FIR Filters that addresses the drawbacks of existing 4-moduli sets. Compared to the existing 4-moduli sets, the proposed moduli set is more balanced, programmable for various dynamic ranges and results in efficient distribution of computation load among the RNS modulo channels.

- A reverse converter model has been derived based on mixed radix conversion algorithm. The reverse converter model is CMOS implementation friendly. The reverse converter circuit for the proposed moduli set outperformed the reverse converter of state-of-the-art balanced moduli set both in terms of delay and area.

- MAC units implemented using the proposed moduli set and balanced 4-moduli set in literature are synthesized and compared for power and cell area. Power improvements of as much as 43% and cell area improvements of as much as 46% are observed over various dynamic ranges.

- RNS FIR filters of 6-taps are implemented following the complete ASIC design flow. The chip area and power are compared to demonstrate the advantages of using the proposed RNS moduli set over the existing balanced moduli set.

- Proposed an application of new logic style to improve the speed of the reverse converter. Using threshold logic, the reverse converter is sped up by 20% for the same power.

[1] B. Cao, C.H. Chang, and T. Srikanthan. An efficient reverse converter for 4-moduli set $\{2^n-1, 2^n, 2^n+1, 2^{2n}+1\}$ based on new Chinese Remainder Theorem. *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, 50:1296–1303, 2003.

[2] B. Cao, T. Srikantham, and C. H. Chang. Efficient reverse converters for four-moduli sets $\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1\}$ and $\{2^n-1, 2^n, 2^n+1, 2^{n-1}-1\}$. *IEE Proc. Comput. Digit. Tech.*, 152:687–696, 2005.

[3] R. Chaves and L. Sousa. $\{2^n-1, 2^{n+k}, 2^n+1\}$: A New RNS Moduli Set Extension. In *Proc. DSD*, pages 210–217, 2004.

[4] R. Chokshi, K.S. Berezowski, A. Shrivatsava, and S.J. Piestrak. Exploiting residue number system for power-efficient digital signal processing in embedded processors. In *Proc. International Conference on Compilers, Architecture, and Synthesis for Embedded systems*, pages 19–27, 2009.

[5] M. Re G.C. Cardarilli; A. Del Re; A. Nannarelli. Low Power and Low Leakage Implementation of RNS FIR Filters . In *Thirty-Ninth Asilomar Conference on Signals, Systems and Computers*, pages 1620–1624, 2005.

[6] A. Hiasat and H. S. Abdel-Aty-Zohdy. Residue-to-binary arithmetic converter for the moduli set $(2^k, 2^k-1, 2^{k-1}-1)$. *IEEE Trans. Circuits Syst. II*, 45:204–208, 1998.

[7] A. Hosangadi; F. Fallah; R. Kastner. Simultaneous optimization of delay and number of operations in multiplierless implementation of linear systems . In *Proc. Int. Workshop Logic Synthesis*, 2005.

[8] P.V.Ananda Mohan and A.B. Premkumar. RNS-to-Binary Converters for Two Four-Moduli sets $\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1\}$ and $\{2^n-1, 2^n, 2^n+1, 2^{n+1}+1\}$. *IEEE Trans. Circuits Syst. I*, 54(6):1245–1254, 2006.

[9] A.S. Molahosseini, K. Navi, C. Dadkhah, O. Kavehei, and S. Timarchi. Efficient Reverse Converter Designs for the New 4-Moduli Sets $\{2^n-1, 2^n, 2^n+1, 2^{2n+1}-1\}$ and $\{2^n-1, 2^n+1, 2^{2n}, 2^{2n}+1\}$ Based on New CRTs. *IEEE Trans. Circuits Syst. I*, 57:823–835, 2010.

[10] Alberto Nannarellli, Macro Re, and Gian Carlo Cardaralli. Tradeoffs Between Residue Number System and Traditional FIR Filters. In *International Symposium on Circuits and Systems*, pages II305–II308, 2001.

[11] S.J. Piestrak. Design of residue generators and multioperand modulo adders using carry-save adders. *IEEE Trans. Computers*, 43:68–77, 1994.

[12] F. Pourbigharaz and H. M. Yassine. A signed-digit architecture for residue to binary transformation. *IEEE Trans. Comput*, 46:1146–1150, 1997.

[13] A. B. Premkumar, E.L. Ang, and E. M.-K. Lai. A formal framework for conversion from binary to residue numbers. *IEEE Trans. Circuits Syst. II:Analog and Digital Signal Processing*, 49:135–144, 2002.

[14] A. B. Premkumar, E.L. Ang, and E. M.-K. Lai. Improved Memoryless RNS Forward Converter Based on the Periodicity of Residues. *IEEE Trans. Circuits Syst. II*, 53:133–137, 2006.

[15] B. Premkumar. An RNS to binary converter in 2n + 1; 2n; 2n 1 moduli set. *IEEE Trans. Circuits Syst. II*, 39:480–482, 1992.

[16] X.Yao G.Chalivendra S. Patel S. Leshner, K.Berezowski and S.Vrudhula. A low power, high performance threshold logic-based standard cell multiplier in 65nm CMOS. In *IEEE Symposium on VLSI (ISVLSI) 2010*, 2010.

[17] Ming-Hwa Sheu, Su-Hon Lin, Yung-Tai Chen, and Yu-Chun Chang. High-Speed and Reduced-Area RNS Forward Converter Based on $(2^n-1, 2^n, 2^n+1)$. In *Proc. APCCS*, pages 821–824, 2004.

[18] M.A. Soderstrand, W.K. Jenkins, G.A. Jullien, and F.J. Taylor. *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*. IEEE Press, Newyork, 1986.

[19] N Stamenkovi. Digital fir filter architecture based on the residue number system. *Facta Universitatis - series : Architecture and Civil Engineering*, 22:125, 2009.

[20] W. Wang, M. N. S. Swamy, M. O. Ahmad, and Y. Wang. A parallel residue-to-binary converter. In *IEEE Inte. Conf. Acoustics, Speech, and Signal Processing*, 1999.

[21] Wei Wang, M.N.S Swamy, and M.O. Ahmad. Moduli Selection in RNS for Efficient VLSI Implementation. In *International Symposium on Circuits and Systems*, pages IV–512–IV–515, 2003.

[22] Yuke Wang. Residue-to-Binary converters based on new Chinese remainder theourems. *IEEE Trans. Circuits Syst. II:Analog and Digital Signal Processing*, 47:197–205, 2000.

[23] R. Zimmermann. Efficient VLSI Implementation of Modulo $(2^n + 1)$ Addition and Multiplication. In *Proc. ISCA*, pages 158–167, 1999.