VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV MIKROELEKTRONIKY

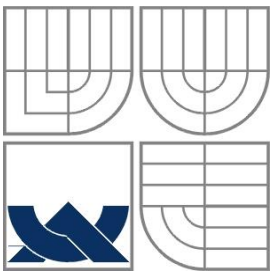
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF MICROELECTRONICS

RESIDUE NUMBER SYSTEM BASED BUILDING BLOCKS FOR APPLICATIONS IN DIGITAL SIGNAL PROCESSING

DIZERTAČNÍ PRÁCE
DOCTORAL THESIS

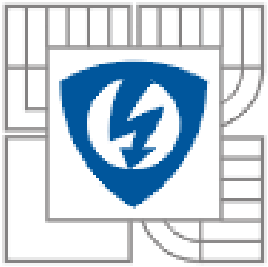
AUTOR PRÁCE Ing. DINA YOUNES
AUTHOR

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV MIKROELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF MICROELECTRONICS

RESIDUE NUMBER SYSTEM BASED BUILDING BLOCKS FOR APPLICATIONS IN DIGITAL SIGNAL PROCESSING

VYUŽITÍ SYSTÉMU ZBYTKOVÝCH TŘÍD PRO ZPRACOVÁNÍ DIGITÁLNÍCH SIGNÁLŮ

DIZERTAČNÍ PRÁCE
DOCTORAL THESIS

AUTOR PRÁCE — Ing. DINA YOUNES
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. PAVEL ŠTEFFAN, Ph.D.
SUPERVISOR

BRNO 2013

Abstract

This doctoral thesis deals with designing residue number system based building blocks to enhance the performance of digital signal processing applications.

The residue number system (RNS) is a non-weighted number system that provides carry-free, parallel, high speed, secure and fault tolerant arithmetic operations. These features make it very attractive to be used in high-performance and fault tolerant digital signal processing (DSP) applications.

A typical RNS system consists of three main components; the first one is the binary to residue converter that computes the RNS equivalent of the inputs represented in the binary number system. The second component in this system is parallel residue arithmetic units that perform arithmetic operations on the operands already represented in RNS. The last component is the residue to binary converter, which converts the outputs back into their binary representation.

The main aim of this thesis was to propose novel structures of the basic components of this system in order to be later used as fundamental units in DSP applications.

This thesis encloses improving and designing novel structures of these components, simulating and verifying their efficiency via FPGA implementation. In addition to suggesting novel structures of basic RNS components, a detailed study on different moduli sets that compares and determines the most efficient one for different dynamic range requirements is also presented. One of the main outcomes of this thesis is concluding and verifying the main condition that should be met when choosing a moduli set, in order to improve the timing performance of a DSP application. An RNS-based image processing application is also proposed. Its efficiency, in terms of timing performance and power consumption, is proved via comparing it with a binary-based one. Finally, the main considerations that should be taken into account when choosing to use the binary number system or RNS are also discussed in details.

Keywords

Residue number system, digital signal processing, modular arithmetic, moduli set, dynamic range, binary to RNS converter, RNS to binary converter, RNS-based application, parallel processing, power reduced DSP application, FPGA implementation.

Abstrakt

Předkládaná disertační práce se zabývá návrhem základních bloků v systému zbytkových tříd pro zvýšení výkonu aplikací určených pro digitální zpracování signálů (DSP).

Systém zbytkových tříd (RNS) je neváhová číselná soustava, jež umožňuje provádět paralelizovatelné, vysokorychlostní, bezpečné a proti chybám odolné aritmetické operace, které jsou zpracovávány bez přenosu mezi řády. Tyto vlastnosti jej činí značně perspektivním pro použití v DSP aplikacích náročných na výpočetní výkon a odolných proti chybám.

Typický RNS systém se skládá ze tří hlavních částí: převodníku z binárního kódu do RNS, který počítá ekvivalent vstupních binárních hodnot v systému zbytkových tříd, dále jsou to paralelně řazené RNS aritmetické jednotky, které provádějí aritmetické operace s operandy již převedenými do RNS. Poslední část pak tvoří převodník z RNS do binárního kódu, který převádí výsledek zpět do výchozího binárního kódu.

Hlavním cílem této disertační práce bylo navrhnout nové struktury základních bloků výše zmiňovaného systému zbytkových tříd, které mohou být využity v aplikacích DSP.

Tato disertační práce předkládá zlepšení a návrhy nových struktur komponent RNS, simulaci a také ověření jejich funkčnosti prostřednictvím implementace v obvodech FPGA. Kromě návrhů nové struktury základních komponentů RNS je prezentován také podrobný výzkum různých sad modulů, který je srovnává a determinuje nejefektivnější sadu pro různé dynamické rozsahy. Dalším z klíčových přínosů disertační práce je objevení a ověření podmínky určující výběr optimální sady modulů, která umožňuje zvýšit výkonnost aplikací DSP. Dále byla navržena aplikace pro zpracování obrazu využívající RNS, která má vůči klasické binární implementanci nižší spotřebu a vyšší maximální pracovní frekvenci. V závěru práce byla vyhodnocena hlavní kritéria při rozhodování, zda je vhodnější pro danou aplikaci využít binární číselnou soustavu nebo RNS.

Klíčová slova

Systém zbytkových tříd, digitální zpracování signálu, modulární aritmetika, sada modulů, dynamický rozsah, převodník z binární soustavy do RNS, převodník z RNS do binární soustavy, aplikace RNS, paralelní výpočty, aplikace DSP s nízkou spotřebou, implementace do FPGA.

YOUNES, D. *Residue number system based building blocks for applications in digital signal processing.* Brno: Brno University of Technology, Faculty of Electrical Engineering and Communication, Department of Microelectronics, 2013, 106 p. Supervised by doc. Ing. Pavel Šteffan, Ph.D.

Declaration

I declare that I have elaborated my doctoral thesis on the theme of "Residue number system based building blocks for applications in digital signal processing" independently, under the supervision of the doctoral thesis supervisor and with the use of technical literature and other sources of information which are all quoted in the thesis and detailed in the list of literature at the end of the thesis.

As the author of the doctoral thesis I furthermore declare that, concerning the creation of this doctoral thesis, I have not infringed any copyright. In particular, I have not unlawfully encroached on anyone's personal copyright and I am fully aware of the consequences in the case of breaking Regulation S 11 and the following of the Copyright Act No 121/2000 Vol., including the possible consequences of criminal law resulted from Regulation S 152 of Criminal Act No 140/1961 Vol.

Brno …… ……

(author's signature)

Acknowledgment

This doctoral thesis would not have been possible without the advice, guidance and help of the kind people around me who encouraged and supported me during this work.

First, I would like to express my deep gratitude to my advisor doc. Ing. Pavel Šteffan, Ph.D. for his support, guidance and help during the past four years I worked and studied at the Department of Microelectronics, Brno University of Technology.

I am very thankful to prof. Ing. Vladislav Musil, CSc. head of the Department of Microelectronics for his priceless advices, suggestions and continuous assistance during my work on Ph.D. thesis.

I would like to thank all other members of the Department of Microelectronics, who assisted me with advice or comment, to all my colleagues.

Finally, my great thanks belong to my parents, sister and husband who always believed in me and have always been my biggest supporter.

Table of Contents

List of Abbreviations

BAU	binary arithmetic unit
BNS	binary number system
CRT	Chinese remainder theorem
CPA	carry propagate adder
CSA	carry save adder
CSA-EAC	carry save adder with end around carry
DR	dynamic range
DSP	digital signal processing
EAC	end around carry
FA	full adder
FC	forward converter (binary to residue converter)
FIR filter	finite impulse response filter
FPGA	field programmable gate array
GCD	greatest common divisor
HA	half adder
IIR filter	infinite impulse response filter
LUT	look-up table
MRC	mixed radix conversion
MSB	most significant bit
RAM	random access memory
RAU	residue arithmetic unit
RC	reverse converter (residue to binary converter)
RNS	residue number system
ROM	read only memory
RRNS	redundant residue number system
VLSI	very large scale integration

List of Figures

List of Tables

1 Introduction

This thesis is concerned with an unconventional non-weighted number system that has gained a great scientific interest; the residue number system (RNS). Designing new and more efficient RNS based building blocks that improve digital signal processing (DSP) applications' performance is the main aim of this thesis.

Since the whole work will be devoted on the RNS, enclosing a brief introduction about this number system will be beneficial.

The RNS is a very old number system. It was found 1500 years ago by a Chinese scholar Sun Tzu. Since the last five decades, RNS's features have been rediscovered and thus the interest in this system has been renewed. The researchers have used the RNS in order to benefit from its features in designing high-speed and fault-tolerance applications.

The fundamental idea of the RNS is based on uniquely representing large binary numbers using a set of smaller residues, which results in carry-free, high-speed and parallel arithmetic [1].

This system is based on modulus operation, where the divider is called modulo and the remainder of the division operation is called residue. The basic notation in RNS is,

$$x_i = X \bmod m_i = \langle x_i \rangle_{m_i} \quad ; \qquad 0 \le x_i < m_i \tag{1.1}$$

Each integer in RNS is represented by a set of residues corresponding to a specified moduli set. The main condition is that the moduli within the moduli set should be relatively prime,

$$X \xrightarrow{RNS} \left(\langle x_1 \rangle_{m_1}, \langle x_2 \rangle_{m_2}, \ldots, \langle x_n \rangle_{m_n} \right) \quad ; \qquad GCD(m_i, m_j) = 1 \tag{1.2}$$

The RNS uniquely represents any integer X that locates in its dynamic range M, which is the product of the moduli within the moduli set.

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

Both signed and unsigned integers can be represented in the RNS. For unsigned RNS, the range of the representable integers is,

$$0 \le X < M \tag{1.4}$$

For signed RNS, the range of representable integers is partitioned into two equal intervals,

$$\begin{aligned} 0 \le X < \lfloor M/2 \rfloor \qquad & \textit{for positive numbers} \\ \lfloor M/2 \rfloor \le X < M \qquad & \textit{for negative numbers} \end{aligned} \tag{1.5}$$

In principle, any interval of M consecutive integers can be uniquely represented in the RNS. However, the standard conventions on representable integer ranges in the RNS are illustrated in equations (1.4) and (1.5).

The principal aspect that distinguishes the RNS from other number systems is that the standard arithmetic operations; addition, subtraction and multiplication are easily implemented, whereas operations such as division, root, comparison, scaling and overflow and sign detection are more complicated. Therefore, the RNS is extremely useful in applications that require a large number of addition and multiplication, and a minimum number of comparisons, divisions and scaling. In other words, the RNS is preferable in applications in which additions and multiplications are critical. Such applications are DSP, image processing, speech processing, cryptography and transforms [2].

The main RNS advantage is the absence of carry propagation between digits, which results in high-speed arithmetic needed in embedded processors. Another important feature of RNS is the digits independence, so an error in a digit does not propagate to other digits, which results in no error propagation, hence providing fault-tolerance systems. In addition, the RNS can be very efficient in complex-number arithmetic, because it simplifies and reduces the number of multiplications needed. All these features increase the scientific tendency toward the RNS especially for DSP applications. However, the RNS is still not popular in general-purpose processors, due the aforementioned difficulties.

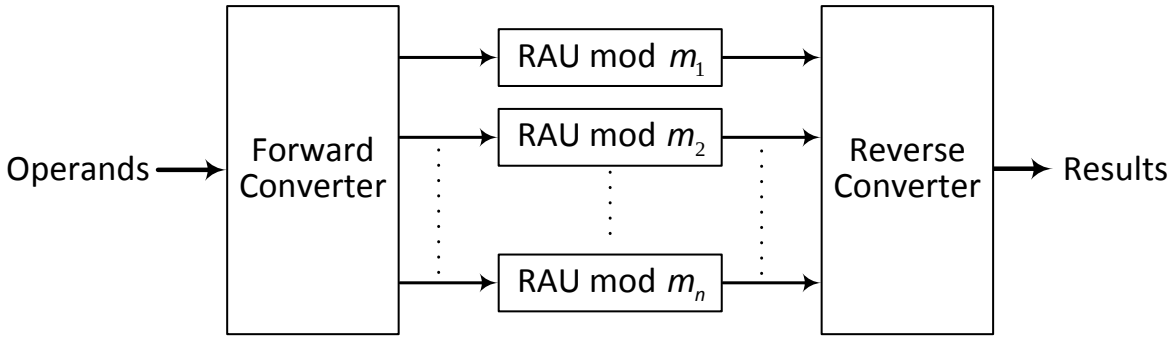


Fig. 1.1: The architecture of the residue number system (RNS)

The basic RNS processor's architecture is shown in Fig. 1.1. It consists of three main components; a forward converter (binary to residue converter), that converts the binary number to n equivalent RNS residues, corresponding to the n moduli. The n residues are then processed using n parallel residue arithmetic units (RAUs); each of them corresponds to one

modulo. The *n* outputs of these units represented in RNS are then converted back into their binary equivalent, by utilizing the reverse converter (residue to binary converter).

The structure of this desecration is organized as follows; Chapter 2 presents a brief survey about the most recently achievements in the RNS, concerning different proposed moduli sets that provide different dynamic ranges, the common means and structures to perform forward and reverse conversion, general structures of residue arithmetic units and applications where using the RNS is advantageous. Then, the main aims and purposes of this dissertation are stated in Chapter 3. Chapter 4 is dedicated to present the dissertation results including the proposed RNS components, proposed RNS-based applications, comparisons between RNS and binary-based applications, and the cases when RNS should be used. Moreover, some widely accepted concepts are proven wrong. Finally, the conclusions, outcomes and the final remarks of this dissertation are illustrated in Chapter 5.

2 State of the art

This chapter presents a summary of the fundamental principles of the RNS; RNS's components, their basics and the most commonly used designs. The most recent work and researches are also presented in this chapter. Each component in the RNS will be separately discussed, in order to provide a brief overview and state the recent achievements in its field.

The interest in RNS arithmetic has started since 1950's [1], [2]. The first hardware based on the RNS was built in 1967. The work in this field continued and many improvements in all areas of the RNS have arisen, in order to enhance its features, resolve its related problems and find suitable applications that benefit from RNS's features. Most of the early designs of RNS were based on read-only memories (ROM). However, the great advance in VLSI (very large scale integration) technology paved the way for new approaches in designing RNS systems.

New trends to design non-ROM based RNS have appeared. Subsequently, much work has been devoted for special moduli sets. Excellent results in terms of computational speed have been achieved in 2000 [2].

The most important issues that must be taking into account when designing an RNS system are, a proper moduli set selection, forward conversion, residue arithmetic units and reverse conversion. A brief of each of these issues and its recent achievements are separately discussed in the following sections.

2.1 Moduli set selection

Choosing a proper modulo set is an essential issue for building an efficient RNS with a sufficient dynamic range (DR). The number, form and value of the moduli affect the dynamic range, timing performance and hardware complexity of an RNS-based application [3].

The moduli set in the RNS can be either arbitrary or special. In principal, special moduli sets were suggested in order to simplify the implementation of arithmetic operations. This invariably means that arithmetic on residue digits should not deviate too far from conventional arithmetic, which is just arithmetic modulo a power of two [1]. On the other hand, arithmetic circuits based on arbitrary moduli sets are much more complex and time consuming. These sets are utilized in cases when using special moduli sets imposes some constraints.

The most famous moduli set is $\{2^n - 1, 2^n, 2^n + 1\}$ [4]. This set has been known as a means of simplifying the calculations necessary to implement the reverse converter (RC). However, this set has modulo $(2^n + 1)$ channel that represents the bottleneck of the system. Its arithmetic circuits suffer from the longest delay among all three channels.

In general, arithmetic circuits modulo $(2^k - 1)$ are more efficient than those modulo $(2^k + 1)$, therefore, it is better to reduce the number of moduli of the form $(2^k + 1)$ [5]. Thus, in order to simplify the complexity caused by modulo $(2^n + 1)$ in the set $\{2^n - 1, 2^n, 2^n + 1\}$ [4], new moduli sets $\{2^{n-1} - 1, 2^n - 1, 2^n\}$ [6] and $\{2^n - 1, 2^n, 2^{n+1} - 1\}$ [7], that substitute this modulo with another of the form $(2^k - 1)$, have been suggested. These three sets have a $3n$-bit DR, which is sufficient for applications that require medium DRs (less than 22 bits).

However, many DSP applications require larger DRs, therefore, new moduli sets $\{2^n - 1, 2^n, 2^{2n+1} - 1\}$ [8] and $\{2^n - 1, 2^n + 1, 2^{2n} + 1\}$ [9] that provide $4n$-bit DR and $\{2^n, 2^{2n} - 1, 2^{2n} + 1\}$ [10] that provides $5n$-bit DR, have been suggested. Although the DR is larger, the delay of the RAUs based on these sets has considerably increased, due to utilizing moduli with greater magnitudes. In order to eliminate this drawback and maintain the large DR, sets of four and five moduli have been suggested, such as $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$ [11]-I, $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} + 1\}$ [11]-II, $\{2^n - 1, 2^n, 2^n + 1, 2^{2n} + 1\}$[12], $\{2^n - 1, 2^n, 2^n + 1, 2^{2n+1} - 1\}$[13]-I, $\{2^n - 1, 2^n, 2^n + 1, 2^n - 2^{(n+1)/2} + 1, 2^n + 2^{(n+1)/2} + 1\}$ [14], $\{2^n - 1, 2^n + 1, 2^{2n} - 2, 2^{2n+1} - 3\}$ [15], $\{2^n - 1, 2^{2n}, 2^n + 1, 2^{2n} + 1\}$ [13]-II, $\{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1, 2^{n+1} + 1\}$ [16], $\{2^n, 2^{n/2} - 1, 2^{n/2} + 1, 2^n + 1, 2^{2n-1} - 1\}$ [17], $\{2^{n/2} - 1, 2^{n/2} + 1, 2^n + 1, 2^{2n+1} - 1\}$ [18], $\{2^n + 1, 2^n - 1, 2^{2n}, 2^{2n+1} - 1\}$ [19] and $\{2^{2n+1}, 2^{2n} + 1, 2^n + 1, 2^n - 1\}$ [20]. Each of these sets has its own advantages and disadvantages. Some of them offer higher DRs than others, while others have more parallelism. Some of them concentrated on designing efficient RCs, while others on efficient RAUs.

Tab. 2.1 illustrates the most recently published moduli sets, including the dynamic ranges they provide and possible n values that can be used in these sets.

Tab. 2.1: The most recently published moduli sets

Number of moduli	Modulo set	Dynamic range	*n* odd/even
Three moduli sets	$\{2^n - 1, 2^n, 2^n + 1\}$ [4]	$3n$	any
	$\{2^{n-1} - 1, 2^n - 1, 2^n\}$ [6]	$3n - 1$	any
	$\{2^n - 1, 2^n, 2^{n+1} - 1\}$ [7]	$3n + 1$	any
	$\{2^n - 1, 2^n, 2^{2n+1} - 1\}$ [8]	$4n + 1$	any
	$\{2^n - 1, 2^n + 1, 2^{2n} + 1\}$ [9]	$4n$	any
	$\{2^n, 2^{2n} - 1, 2^{2n} + 1\}$ [10]	$5n$	even
Four moduli sets	$\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$ [11]-I	$4n + 1$	even

	$\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} + 1\}$ [11]-II	$4n + 1$	odd
	$\{2^{n/2} - 1, 2^{n/2} + 1, 2^n + 1, 2^{2n+1} - 1\}$ [18]	$4n + 1$	even
	$\{2^n - 1, 2^n, 2^n + 1, 2^{2n} + 1\}$ [12]	$5n$	any
	$\{2^n - 1, 2^n, 2^n + 1, 2^{2n+1} - 1\}$ [13]-I	$5n + 1$	any
	$\{2^n - 1, 2^{2n}, 2^n + 1, 2^{2n} + 1\}$ [13]-II	$6n$	any
	$\{2^n - 1, 2^n + 1, 2^{2n} - 2, 2^{2n+1} - 3\}$ [15]	$6n + 1$	any
	$\{2^n + 1, 2^n - 1, 2^{2n}, 2^{2n+1} - 1\}$ [19]	$6n + 1$	any
	$\{2^{2n+1}, 2^{2n} + 1, 2^n + 1, 2^n - 1\}$ [20]	$6n + 1$	any
Five moduli sets	$\{2^n, 2^{n/2} - 1, 2^{n/2} + 1, 2^n + 1, 2^{2n-1} - 1\}$ [17]	$5n - 1$	even
	$\{2^n - 1, 2^n, 2^n + 1, 2^n - 2^{(n+1)/2} + 1, 2^n + 2^{(n+1)/2} + 1\}$ [14]	$5n$	odd
	$\{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1, 2^{n+1} + 1\}$ [16]	$5n$	even

2.2 RNS Converters

Every RNS system involves forward and reverse converters that convert weighted numbers into their equivalent RNS representation and vice versa, respectively.

The structures of these converters can be memory-based, conventional-based or mix of both. The choice is actually determined by the dynamic range required for the application being designed. For applications with small dynamic ranges, such as digital image processing where the range of pixel values is [0,255], the memory-based converters are the most efficient. Contrary, for applications with large dynamic ranges (greater than 22 bits), such as cryptography and some FIR filters, the combinational structure of the converters is preferred. The next two sections illustrate the converters based on the combinational structure.

2.2.1 Binary to residue converters

The structure of the binary to residue converter (forward converter) is rather simple. Hence, little work was devoted to this component [21].

In order to illustrate the forward conversion process, forward conversion equations corresponding to the special moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ will be stated [1], [2]. Actually, by modifying k, these equations can be applied with any modulo of the form $(2^k \pm 1)$ within the sets presented in Tab. 2.1.

Assuming X is a $3n$-bit integer. X can be written as follows,

$$X = (b_{3n-1}\, b_{3n-2} \ldots b_n\, b_{n-1} \ldots b_0)_2 = B_1 2^{2n} + B_2 2^n + B_3 \tag{2.1}$$

where, $b_{3n-1} \ldots b_0$ are the binary digits (bits) of X. B_1, B_2, B_3 are blocks, each of them contains n bits.

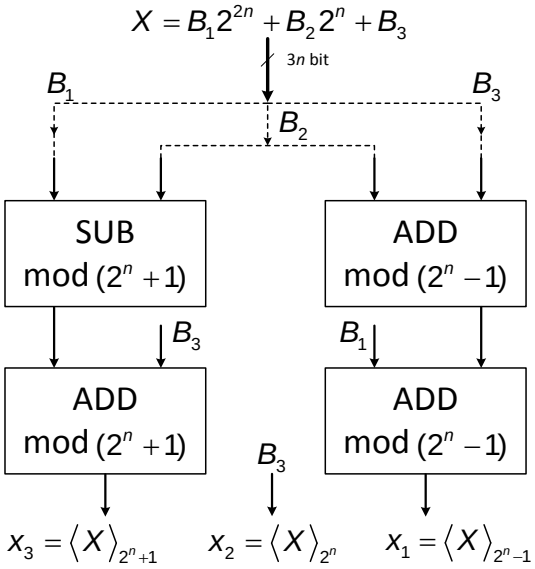


Fig. 2.1: RNS forward converter for the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ [1]

The RNS representation of X according to the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$,

$$\langle x_1 \rangle_{2^n-1} = \langle X \rangle_{2^n-1} = \left\langle B_1 2^{2n} + B_2 2^n + B_3 \right\rangle_{2^n-1} = \langle B_1 + B_2 + B_3 \rangle_{2^n-1} \tag{2.2}$$

$$\langle x_2 \rangle_{2^n} = \langle X \rangle_{2^n} = \left\langle B_1 2^{2n} + B_2 2^n + B_3 \right\rangle_{2^n} = \langle B_3 \rangle_{2^n} \tag{2.3}$$

$$\langle x_3 \rangle_{2^n+1} = \langle X \rangle_{2^n+1} = \left\langle B_1 2^{2n} + B_2 2^n + B_3 \right\rangle_{2^n+1} = \langle B_1 - B_2 + B_3 \rangle_{2^n+1} \tag{2.4}$$

Equations (2.2), (2.3) and (2.4) are extracted based on the following,

$$\begin{aligned} \left\langle 2^n \right\rangle_{2^n-1} &= \left\langle 2^n - 1 + 1 \right\rangle_{2^n-1} = \langle 1 \rangle_{2^n-1} \\ \left\langle 2^n \right\rangle_{2^n} &= \langle 0 \rangle_{2^n} \\ \left\langle 2^n \right\rangle_{2^n+1} &= \left\langle 2^n - 1 + 1 \right\rangle_{2^n-1} = \langle -1 \rangle_{2^n+1} \end{aligned} \tag{2.5}$$

According to equations (2.2), (2.3) and (2.4), the general structure of the forward converter for the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ is shown in Fig. 2.1 [1],

2.2.2 Residue to binary converters

Unlike forward converters, residue to binary converters (reverse converters) gained much more interest due to their complexity. This component is considered the most time consuming component in the whole RNS system. It is also used for performing difficult RNS operations (division, scaling, comparison, overflow and sign detection). Researchers continuously try to reduce the delay of the reverse converters, due to the reason that having a slow reverse converter may counteract the speed gain of the residue arithmetic unit, hence, ruining the whole advantages of using the RNS.

Reverse conversion algorithms are based on the Chinese remainder theorem (CRT), mixed-radix conversion (MRC) and new Chinese remainder theorems (new CRTs). Every algorithm has its own advantages and disadvantages. The decision to use any of them is based on the used moduli set, the application being designed and the design's requirements (time, area, power). All reverse conversion methods depend on computing multiplicative inverses. The multiplicative inverse x^{-1} of residue x relative to modulo m is defined as follows,

$$\left\langle x \times x^{-1} \right\rangle_m = 1 \quad ; \quad 0 \le x, x^{-1} < m \tag{2.6}$$

It is clear that finding x^{-1} is not a simple task. However, using special moduli sets can make the computation of multiplicative inverses easier.

According to the CRT, a weighted number X can be calculated from its residues (x_1, x_2,…, x_n) by the following equation [1], [2],

$$X = \left\langle \sum_{i=1}^{n} \left\langle x_i N_i \right\rangle_{m_i} M_i \right\rangle_M \tag{2.7}$$

where, $M = m_1 \times m_2 \times \cdots \times m_n$, $M_i = M / m_i$ and $N_i = \left\langle M_i^{-1} \right\rangle_{m_i}$ is the multiplicative inverse of M_i relative to modulo m_i.

The CRT-based converter can be implemented in parallel. However, it needs a large modular adder, which can be very difficult for hardware implementation. Reverse converters based on the CRT were proposed in [4], [6], [7], [10] and [14].

By using the MRC, a residue number (x_1, x_2,…, x_n) can be converted back into its weighted equivalent X by,

$$X = v_n \prod_{i=1}^{n-1} m_i + \ldots + v_3 m_2 m_1 + v_2 m_1 + v_1 \tag{2.8}$$

where,

$$v_1 = x_1 \tag{2.9}$$

$$v_2 = \left\langle (x_2 - v_1) \times \left\langle m_1^{-1} \right\rangle_{m_2} \right\rangle_{m_2} \tag{2.10}$$

$$v_3 = \left\langle ((x_3 - v_1) \times \left\langle m_1^{-1} \right\rangle_{m_3} - v_2) \times \left\langle m_2^{-1} \right\rangle_{m_3} \right\rangle_{m_3} \tag{2.11}$$

As illustrated in equation (2.8), the MRC does not need any special modular adder. However, it is a sequential algorithm, which makes it not suitable for systems with more than four moduli within the set [7], [8]. To overcome such a case (more than four moduli), a two-level structure consisting of the MRC and one of the CRTs is proposed in [11], [16], [17].

The new CRT-I is a modification of the original CRT, where the size of the final modular adder is reduced by one modulo. Using this algorithm, a residue number (x_1, x_2,…, x_n) can be converted back into its weighted equivalent X by,

$$X = x_1 + m_1 \times \left\langle k_1(x_2 - x_1) + k_2 m_2 (x_3 - x_2) + \cdots + k_{n-1} m_2 m_3 \cdots m_{n-1} (x_n - x_{n-1}) \right\rangle_{m_2 m_3 \cdots m_n} \tag{2.12}$$

where,

$$\left\langle k_1 \times m_1 \right\rangle_{m_2 m_3 \cdots m_n} = 1 \tag{2.13}$$

$$\left\langle k_2 \times m_1 \times m_2 \right\rangle_{m_3 \cdots m_n} = 1 \tag{2.14}$$

$$\left\langle k_{n-1} \times m_1 \times m_2 \times \cdots \times m_{n-1} \right\rangle_{m_n} = 1 \tag{2.15}$$

As can be noticed in equation (2.12), the final modular adder is reduced by one modulo. This can bring a great benefit when the first modulo is of the 2^k form, and the multiplication of the rest moduli is of the $(2^k - 1)$ form. Such reverse converters are reported in [9], [12].

The new CRT-II even further reduces the size of the final modular adder. A residue number (x_1, x_2,…, x_n) can be converted back into its weighted equivalent X by the new CRT-II by,

$$\begin{aligned} X &= Z + m_1 m_2 \left\langle k_1 (Y - Z) \right\rangle_{m_3 m_4} \\ Z &= x_1 + m_1 \left\langle k_2 (x_2 - x_1) \right\rangle_{m_2} \\ Y &= x_3 + m_3 \left\langle k_3 (x_4 - x_3) \right\rangle_{m_4} \end{aligned} \tag{2.16}$$

where, $\left\langle k_1 m_1 m_2 \right\rangle_{m_3 m_4} = 1$, $\left\langle k_2 m_1 \right\rangle_{m_2} = 1$, $\left\langle k_3 m_3 \right\rangle_{m_4} = 1$

This algorithm is very efficient. It is used with sets that have four or more moduli. Reverse converters based on the new CRT-II are presented in [13], [15].

2.3 Residue Arithmetic Units

The RNS contains a number of residue arithmetic units corresponding to the number of moduli. These RAUs are totally independent and perform arithmetic operations in parallel.

As aforementioned before, addition, subtraction and multiplication are easy operations in the RNS (RNS-friendly operations). On the other hand, division, scaling, comparison, overflow and sign detection are complex and preferred to be avoided as much as possible.

The RNS friendly operations are carried out by individually performing that operation on each residue corresponding to the moduli. Thus, no carry is propagated from one residue to another. This leads to parallel arithmetic operations, reduced carry propagation length in adders and smaller sizes of multipliers, hence, providing considerably reduced-delay and area applications.

$$\begin{aligned} &X \equiv (x_1, x_2, \ldots, x_n) \quad , \quad Y \equiv (y_1, y_2, \ldots, y_n) \\ &Z \equiv X \circ Y \equiv \left(\langle x_1 \circ y_1 \rangle_{m_1}, \langle x_2 \circ y_2 \rangle_{m_2}, \ldots, \langle x_n \circ y_n \rangle_{m_n} \right) \quad ; \quad \circ \equiv (+, -, \times) \\ &Z \equiv (z_1, z_2, \ldots, z_n) \end{aligned} \tag{2.17}$$

The structure of the RAU is based on one of the following three methods; a pure memory structure, a combinational structure or a mix of both [1], [2], [22]. The first approach is realized by using ROMs [23]. The main drawback of this approach is the exponential growth of the memory size for large moduli. Therefore, this approach is suitable only for small moduli. The second approach depends on pure combinational structure. This approach is suitable for large moduli [24]. A RAU based on the third approach, that uses both memory and combinational circuits, is presented in [25].

2.3.1 Modular addition

Modular addition is a fundamental operation in the RNS. It is used in almost every part of the RNS (forward converter, reverse converter, modular multipliers, modular subtractors and modular adders themselves). Therefore, designing efficient modular adders has gained a wide interest. The primary equation for performing general modular addition, which was hardware-realized in [22], is defined by,

$$\langle x + y \rangle_m = \begin{cases} x + y & ; \quad if \ 0 \le x + y < m \\ x + y - m & ; \quad if \ x + y \ge m \end{cases} \tag{2.18}$$

Assuming that the width of modulo *m* is *n* bits, the structure of the general modular adder is shown in Fig. 2.2 [22]. It consists of two *n*-bit adders, an *OR* gate and a multiplexer.

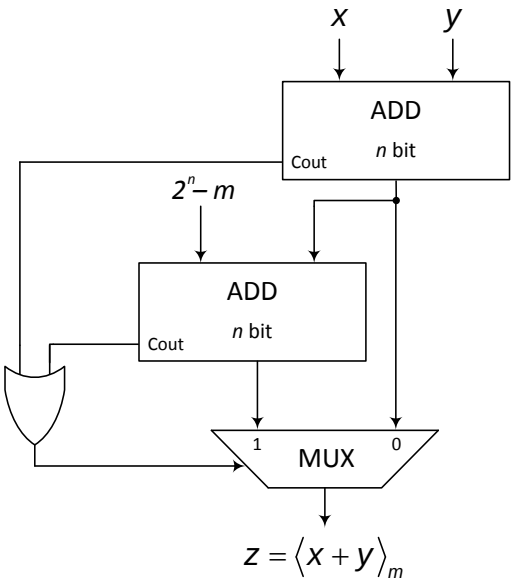


Fig. 2.2: The structure of general modulo adder [22]

As stated above, using special moduli sets can considerably simplify the realization of arithmetic circuits. Regarding the most famous moduli set $\{2^n - 1, 2^n, 2^n + 1\}$, three modular adders are specially designed corresponding to each modulo.

A modulo (2^n) adder can be simply realized using an *n*-bit binary adder, with ignored carry-out. A modulo ($2^n - 1$) adder, can also be simply realized using an *n*-bit binary adder with EAC (end around carry) [26].

However, modulo ($2^n + 1$) adder is considered to be more complex, due to the ($n + 1$)-bit operands and results. It represents the bottleneck of the system. Its arithmetic circuits suffer from the longest delay among all three channels. Therefore, many researchers have focused on this type of modular adders. Diminished-one number system has been used in [27], [28]. In this number system, ($n + 1$)-bit operands are represented using just *n* bits, which results in speeding-up the execution time. However, this speed-up is at the cost of more area consumption occupied by converters to/from the diminished-one representation and special treatment required for operands equal to zero [29]. A quite interest publication [26] has illustrated different structures of modular adders for both general and special moduli sets. In this publication, both standard binary and diminished-one representation for modulo ($2^n + 1$) adders have been used.

Much investigation and research was devoted in order to improve timing performance of modular adders. One of the suggested means is by utilizing faster binary adders, such as parallel prefix adders [28], [29] and [30].

In principal, the general structures of modular adders based on the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ are illustrated in Fig. 2.3 [1], [26]. These structures have been utilized during the study on different moduli sets, which is presented in Section 4.1.

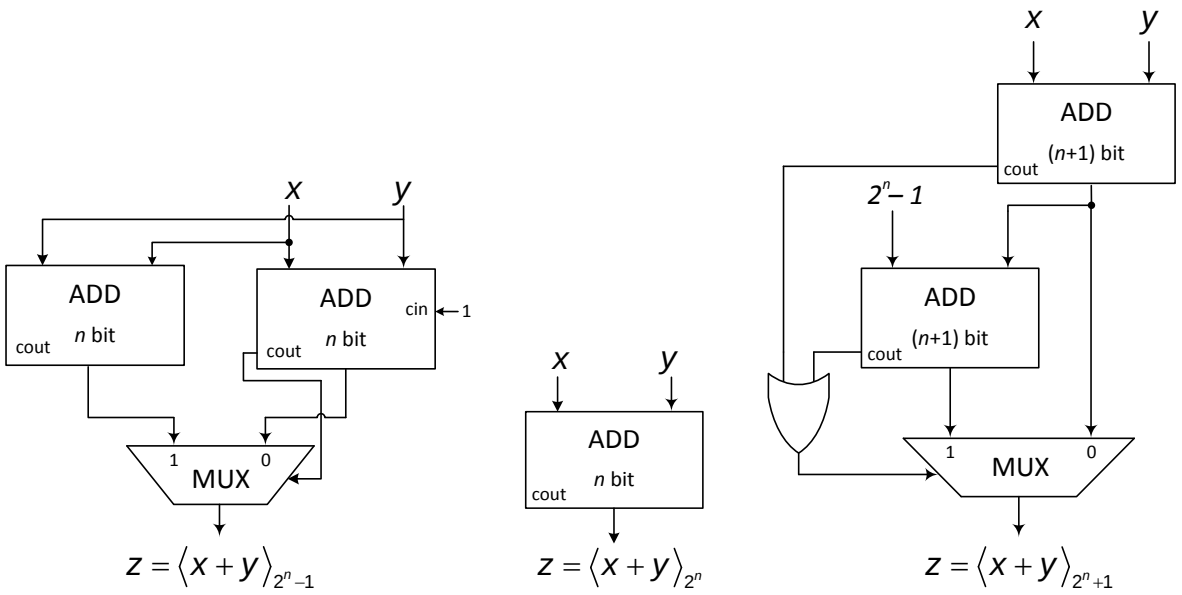


Fig. 2.3: General structures of modular adders based on the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ [1], [26]

2.3.2 Modular subtraction

RNS subtraction is an operation greatly used in many fields of DSP, such as, the mean error estimation, mean square error estimation and calculation of sum of absolute differences [1], [2]. Since modulo arithmetic is also frequently used in these types of applications, efficient modulo subtraction circuits are welcome. However, modular subtraction can be considered as a special case of modular addition, where an additive inverse is used. It is defined as follows,

$$\left\langle \bar{y} \right\rangle_m = \langle -y \rangle_m = \langle m - y \rangle_m \quad \Rightarrow \quad \langle x - y \rangle_m = \left\langle x + \bar{y} \right\rangle_m \tag{2.19}$$

Assuming the width of modulo m is n bits and according to equation (2.19), a general modulo subtractor can be designed using an n-bit subtractor followed by a general modulo m adder. The structure of this subtractor is illustrated in Fig. 2.4 [1], [2].

Very little work was dedicated for studying and designing modular subtractors. According to Property 1, a modulo $(2^n - 1)$ subtractor can be simply realized using modulo $(2^n - 1)$ adder and a few inverters.

Property 1: The residue of a negative residue number $(-x)$ in modulo $(2^n - 1)$ is the one's complement of x, where $0 \leq x < 2^n - 1$.

Therefore, most studies were dedicated for designing efficient modulo $(2^n + 1)$ subtractors, such as [31], [32]. The authors of [31] presented novel architectures of modulo $(2^n + 1)$ subtractors, which are efficient in terms of delay and area using both normal and diminished-one number representation. Moreover, zero handling was also taken into account and a special unit that treats the operands equal to zero was designed.

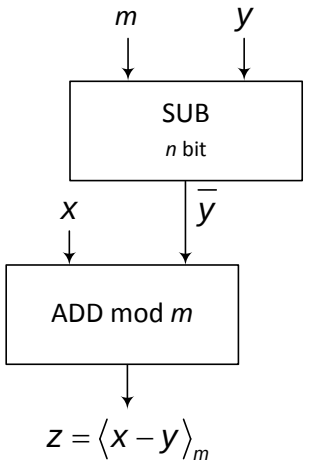


Fig. 2.4: The structure of general modular subtractor [1], [2]

2.3.3 Modular multiplication

Modular multiplication is a very important operation in the RNS. It is used in many applications such as FIR (finite impulse response) filters, Fourier transforms and digital image processing [1], [2]. The speed gain of modular multipliers is indeed the most attractive aspect for using RNS-based DSP applications.

There are two general methods for performing modular multiplication [33]; the first method depends on multiplication then reduction with regard to modulo. This method requires a large space to store the product of the multiplication in order to perform the reduction process thereafter [33]. The basic structure of the modular multiplier based on this method consists of a binary multiplier followed by a reduction unit. This reduction unit can be further simplified in case of using the special moduli sets. This approach has been utilized in [34], [35]. An example of this method is shown in Fig. 2.5. The structure of this multiplier is based on the product-partitioning approach presented in [1].

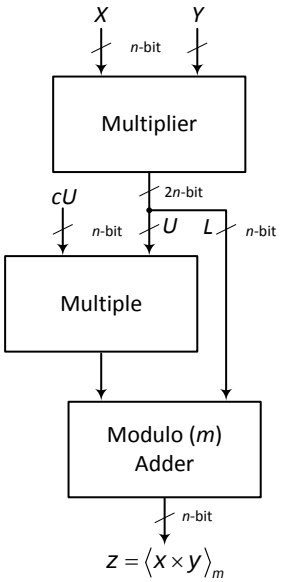


Fig. 2.5: The structure of general modulo multiplier [1]

The second approach is based on interleaving multiplication and reduction. Many publications used this approach. A modular multiplier based on the Montgomery reduction algorithm was presented in [36]. In this structure, the reduction is performed at each iteration step of the multiplication process. Montgomery reduction algorithm is efficient for very large dynamic ranges; where the width of modulo is several hundred bits. Another efficient hardware implementation method for multiplying integers – Wallace tree – was used in RNS multipliers for both moduli ($2^n - 1$, $2^n + 1$) [37], [38]. Two RNS multipliers for moduli ($2^n - 1$, $2^n + 1$) based on modified Booth were published in [39], [40]. According to the authors, these modular multipliers offer fast and completely regular structures, because the modified Booth algorithm reduces the number of partial products to about half of that of the Booth algorithm. An RNS multiplier that depends only on binary adders has been published in [41]. This multiplier is an improved design of [42], whose architecture is almost exclusively composed of full and half adders.

2.3.4 Complex operations in the RNS (overflow detection, sign detection and residue comparison)

As aforementioned before operations as division, overflow detection, sign detection and magnitude comparison are problematic and very complex in the RNS. In some cases, some of these operations are essential and cannot be avoided. Hence, a number of methods for solving these problems have been suggested. A survey of the methods for overflow detection, sign detection and magnitude comparison is briefly presented below.

In principle, the general way to detect overflow in the RNS is via comparing the result of addition with one of the addends. Suppose two integers *x*, *y*. These two integers are being added modulo *m*. If $x \geq 0$ and $y < m$ [1], [2],

$$Overflow = \begin{cases} 1 & ; \quad if \ \langle x+y \rangle_m < x \\ 0 & ; \quad otherwise \end{cases} \tag{2.20}$$

One of the fastest and most efficient ways to detect overflow in the RNS is via parity checking [2], [43], [44]. It indicates whether an integer is even or odd. However, this technique can only be used with odd dynamic ranges. Suppose two integers *x* and *y* have the same parity,

$$Overflow = \begin{cases} 1 & ; \quad if \ \langle x+y \rangle_m \ is \ odd \\ 0 & ; \quad otherwise \end{cases} \tag{2.21}$$

Contrary, if *x* and *y* have different parity,

$$Overflow = \begin{cases} 1 & ; \quad if \ \langle x+y \rangle_m \ is \ even \\ 0 & ; \quad otherwise \end{cases} \tag{2.22}$$

The parity checking technique is one of the best and fastest proposed methods to detect overflow in the RNS. It depends on look-up tables (LUTs) or on an extra modulo (a redundant modulo). However, this technique can only be used with moduli sets that have just odd members, i.e. odd dynamic range, which is not suitable for many moduli sets that uses 2^n as one of its moduli, especially the most famous moduli set $\{2^n - 1, 2^n, 2^n + 1\}$. RNSs with even dynamic ranges have more attractive features than those with odd ones. Due to the reason, that using 2^n modulo greatly simplifies and reduces the delay and complexity of the residue arithmetic operations and the residue-to-binary conversion. Thus, overflow detection in the RNS with even dynamic range is a very important issue.

On the other hand, according to equation (1.5) for signed RNS, the general way for sign detection in the RNS is via comparing the numbers, after reverse converting them back to the weighted form, with half of the dynamic range [1], [2].

Concerning magnitude comparison in the RNS, many comparators based on residue to binary converters were proposed. In principal, the straightforward way to compare two residue numbers in the RNS is to reverse convert them into weighted representation and then carry out a conventional comparison [1]. However, this method is costly; therefore, many approaches were suggested in order to perform comparison on the residue numbers [45], [48].

One of the first suggested approaches is based on a diagonal function that is defined as the sum of suitable quotients of the number named as SUM of Quotients Technique (SQT) [45]. This approach uses an extra modulo that is inserted in the used set of moduli. Another residue comparator, based on the new CRT-II, was suggested in [46]. Contrary to the previous comparator, this one does not use any extra modulo and provides smaller modulo operation. One of the simplest ways to perform residue comparison is based on the new CRT-I [4]. This approach utilizes two parallel binary comparators of $2n$ bits and n bits. A residue comparison algorithm based on the CRT for general moduli sets was suggested in [47]. In this paper, an efficient ROM-free residue comparator for $\{2^n - 1, 2^n, 2^n + 1\}$ was also presented. According to the authors, this comparator is faster and reduces the hardware close to the half of the one based on the new CRT-I [4]. In [48], [49], efficient methods based on the parity checking technique were proposed. However, these methods can only be used with odd moduli sets. Thus, they cannot be used in an RNS based on $\{2^n - 1, 2^n, 2^n + 1\}$.

2.4 RNS Applications

Due to the carry-free, residue independence and parallelism features of the RNS, it has been intensively used in many fields, such as digital signal processing, digital filtering, digital communications, cryptography, error detection and correction [1], [2]. Moreover, new trends to use the RNS in low-power design have also arisen. In principal, this system is of great benefit in areas where addition, subtraction and multiplication are dominant and division, comparison, overflow and sign detection are minor. Hence, the RNS has become a tough candidate for high-performance, fault tolerant and secure DSP applications.

One of the main fields for RNS-based applications is finite impulse response (FIR) filters [50]-[55]. In [50], the authors explore the design workspace of FIR filters with respect to structure, characteristics and number of taps. According to the authors, the proposed RNS-based filter operates at the same throughput as binary filters and has smaller area and power consumption, when the number of taps is larger than 16. In [51], a FIR filer was implemented using the RNS based on any number of moduli of the form $\{2^n - 1, 2^n, 2^n + 1\}$. As reported in this paper, the proposed filter provides a significant overall area-delay product gain ranged from 35% to 60% for a 16-tap filter with dynamic range from 20 to 40 bits. A very interesting and detailed FIR filter performance analysis between RNS and binary is presented in [52]. According to this study, the RNS-based FIR filter is more than 3 times faster and consumes only about 60% of the area of binary-based FIR, when the number of taps is larger than 32.

Furthermore, a number of attempts to design RNS-based infinite impulse response (IIR) filters have been also arisen [56], [57]. However, the results were not impressive as in FIR filters.

Trends to use the RNS for reducing power consumption have also appeared [50], [53]-[55]. In principal, the power dissipation is reduced due to the parallelism feature in the RNS. Since the filtering process is divided into a number of smaller word-length filters that operate in parallel, hence, these smaller computation units require lower supply voltage for specific frequency. One of the most efficient low-power RNS-based FIR filters is presented in [54]. This filter showed static power dissipation reduction of 50% and total power reduction of 40% compared to the binary filters. However, the dynamic range in this filter was limited to 20 bits only and the reduced power consumption was achieved for more than 15 taps.

Digital image processing is another field for benefiting of the RNS's features. Many researches were dedicated for exploiting the RNS features for enhancing digital image processing applications [58]-[62]. One of the first papers that suggested using the RNS in image processing are [58], [59]. However, the main concentration of these papers was on the security aspect rather than benefiting from the parallelism feature of the RNS. An RNS based application for filtering digital images was presented in [60]. The filtering is done in both spatial and frequency domains. Since pixel values have the range [0, 255], the authors suggested using the moduli set {5, 7, 8} as it provides a dynamic range [0, 279], which they considered to be enough for image filtering applications. However, during my study, I have found out that this is not true. An example that clarifies this confusion is presented in Section 4.7.2. In [61], [62], similar structures for edge detection and spatial filtering were also introduced.

In addition, error detection and correction applications greatly benefit from the RNS's features [63], [64]. Due to the carry-free and the lack of weighted significance of residue digits properties, an error in a digit does not propagate, hence, does not affect other digits. One of the suggested methods for detecting and correcting errors is via the redundant RNS (RRNs) [63]. This system uses redundant moduli, thus, errors can be precisely detected and corrected. In this system, the dynamic range is divided into two intervals; the legitimate range and illegitimate range. A single-digit error is detected if the binary result after reverse conversion belongs to the illegitimate range. In order to detect a single-digit error, a single redundant modulo is sufficient. On the other hand, in order to detect and correct a single-digit error, at least two redundant moduli should be utilized. In [64], a new technique for error detection and correction has been proposed. Contrary to [63], the proposed technique is based on dividing the legitimate range into two subsets; legitimate subset and illegitimate subset. Moreover, architecture of a FIR filter with error detection and correction capabilities has been also presented in that paper.

The RNS has also been used in communication for many purposes, such as parallel transmitting a set of orthogonal signals [65] and direct sequence spread spectrum [66]. Furthermore, cryptography is another area where RNS can be efficiently used. The major usage of the RNS is with RSA (Rivest, Shamir and Adleman) algorithm [67].

3 Aims of Dissertation

The main objective of this thesis is, *designing, simulation and FPGA implementation of RNS based building blocks for applications in the field of DSP (binary-to-residue converter, residue-to-binary converter, residue adder and residue multiplier).*

Since the RNS results in carry free arithmetic operations and supports high-speed concurrent computations, it will be useful to use RNS-based building blocks for DSP applications.

Therefore, the main objective of this thesis is improving these building blocks by developing new algorithms and improving existing ones. Hence, the aims of this thesis can be categorized as follows,

Studying different moduli sets, analyzing the relation between moduli number and the dynamic range it provides, and evaluating the most efficient ones for different applications with different dynamic range requirements.

Improving and designing novel RNS converters including both forward and reverse converters. However, the main focus will be concentrated on the reverse converters, since they are the most time and hardware consuming components in the RNS. Comparing ROM-based structures with combinational ones and analyzing the most suitable converters for different applications based on FPGA implementation.

Improving and designing novel structures of residue arithmetic units including modular adders, modular subtractors and modular multipliers with respect to different moduli sets.

Suggesting solutions to simplify RNS difficult operations needed in some DSP applications; such as comparison, overflow and sign detection.

Comparing RNS-based applications with binary-based ones and analyzing the cases when using the RNS will be the most efficient.

Verifying the functionality and efficiency of the proposed designs and comparing them against other published ones based on FPGA implementation.

4 Dissertation Results

This part of the thesis is devoted for presenting the proposed work, findings and results of the doctoral dissertation. In addition to the proposed designs, comparisons of known structures with their analyzing and evaluations are also presented in this Chapter.

Before beginning, aspects that have been taken into account during my research are presented below.

Blocks within the proposed designs have been described using VHDL. ROMs and RAMs have been designed using Xilinx core generator v. 13.4. The proposed designs were simulated using Xilinx ModelSim tool and implemented on different FPGA boards. The maximum frequencies and power consumptions were calculated using Xilinx Timing Analyzer and XPower Analyzer tools v. 13.4. The hardware consumptions are the ones shown in the post place and route reports in Xilinx ISE v. 13.4. The design main goal and strategy was mostly set to "balanced". The individual cases, when other strategies were adopted, are specifically mentioned.

Most of the designs were implemented on FPGA boards. However, for the sake of a fair comparison, the unit gate model was sometimes adopted [75], [76], [81] and [82]. Thus, the considerations that have been taken into account concerning the unit gate model are illustrated in the appendix in Tab. 6.1.

4.1 The most efficient moduli set for each dynamic range

In this section, a study on the effect of the moduli number in a moduli set on the overall speed of the RNS is presented. Choosing a proper moduli set greatly affects the performance of the whole system. The widely known issue is that as the number of moduli increases the speed of the residue arithmetic units increases, whereas the residue-to-binary converters become slower and more complex. It is a double-edged sword, since the greater this number is, the faster residue arithmetic units are and more complex and difficult reverse converters to design. Thus, I carried out a detailed study on different moduli sets with different moduli numbers and different dynamic ranges, and compared timing performance of systems based on them in order to determine the moduli number effect on the overall RNS timing performance and find out the most efficient set for each dynamic range. Timing performance of the reverse converters and residue arithmetic units based on three precise DRs, (12 bits (medium DR), 24 bits (large DR) and 60 bits (very large DR)) is compared, and the most efficient and inefficient set for each DR is evaluated. For the sake of a fair comparison, the unit gate model is adopted. Furthermore, I have used same basic blocks among all designs (e.g. adders and multipliers).

As aforementioned in Section 2.1, many moduli sets were suggested in order to enhance the dynamic range, timing performance and hardware complexity of the residue arithmetic units and reverse converters within the RNS. Generally, the main concentration of publications that introduced these sets was only on the reverse converters rather than the overall performance of the system based on these sets. Thus, I have observed a lack to a detailed comparative study that fairly compares these sets, in terms of DR, number of moduli, time and hardware requirements for implementing reverse converters and residue arithmetic units based on them.

The study has been published in an international conference in Dubai, UAE [75] and an extended version of it has been published in the international journal of Emerging Trends in Computing and Information Sciences [76]. The first paper [75] presents a detailed comparison between different moduli sets based on (delay × complexity) ratio for each component according to three precise DRs (12 bits, 24 bits and 60 bits), i.e. the main concentration of the study is on systems whose main goal and strategy are set to "balanced". Whereas, the second one [76] compares different moduli sets based on the delay of each component. Hence, it presents the moduli number effect on the timing performance of the overall system, i.e. the main concentration of the study is on systems whose main goal and strategy are set to "timing performance".

4.1.1 Time and hardware requirements of residue arithmetic units and reverse converters based on different moduli sets

In the appendix in Tab. 6.2, Tab. 6.3, Tab. 6.4 and Tab. 6.5, different moduli sets, the delay and hardware complexity of the RAUs and reverse converters based on them are illustrated. In these tables, different moduli sets, are categorized according to the dynamic range they provide ($3n$, $4n$, $5n$ and $6n$) bits. Each of these sets, its DR, the possible n values that can be used in this set, the number of its moduli, the critical channel that presents the longest delay, time and hardware requirements for implementing reverse converters, modular adders and modular multipliers are described in details. The values of the least delays and hardware requirements are bold and underlined in order to highlight them.

These tables show that modular adders and multipliers with respect to modulo $(2^k + 1)$ have longer delays than those based on modulo $(2^k - 1)$. For example, the unit gate delays of modulo $(2^n + 1)$ adder and multiplier are less than those of modulo $(2^{n+1} - 1)$, even though, modulo $(2^{n+1} - 1)$ has a greater amplitude than that of $(2^n + 1)$.

Tab. 4.1 and Tab. 4.2 summarize the reuslts illustrated in Tab. 6.2 - Tab. 6.5 and present only the most efficient sets with respect to the reverse converters and RAUs for each dynamic range category. The efficiency of these sets has been estimated based on (delay × complexity)

ratio. It is clear from Tab. 4.1, that the most efficient reverse converters in each DR category are based on the moduli sets whose members can be combined in such a way to produce a final modulo of the form $(2^k - 1)$. Hence, the structure of the reverse converter based on one of the new CRTs becomes rather simple. Regarding the residue arithmetic units, the moduli sets of more members are more efficient, as shown in Tab. 4.2. However, evaluating the performance of these components altogether and comparing these sets for three precise dynamic ranges are presented in details in Section 4.1.2.

Tab. 4.1: The most efficient moduli sets regarding reverse converters for each dynamic range category

Dynamic range	Moduli set	Moduli #	*RC*	
			Delay	Complexity
$3n$	$\{2^n - 1, 2^n, 2^n + 1\}$ [4]	3	$16n + 8$	$31n + 13$
$4n$	$\{2^n - 1, 2^n + 1, 2^{2n} + 1\}$ [9]	3	$32n + 8$	$62n + 8$
$5n$	$\{2^n, 2^{2n} - 1, 2^{2n} + 1\}$ [10]	3	$32n + 4$	$44n + 8$
$6n$	$\{2^n - 1, 2^{2n}, 2^n + 1, 2^{2n} + 1\}$ [13]-II	4	$32n + 12$	$88n + 24$

Tab. 4.2: The most efficient moduli sets regarding RAUs for each dynamic range category

Dynamic range	Moduli set	Moduli #	*Critical channel*	*Modular adders*		*Modular multipliers*	
				Delay	Complexity	Delay	Complexity
$3n - 1$	$\{2^{n-1} - 1, 2^n - 1, 2^n\}$ [6]	3	$(2^n - 1)$	$8n$	$21n - 7$	$16n - 7$	$24n^2 - 35n + 12$
$4n + 1$	$\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$ [11]-I	4	$(2^n + 1)$	$8n + 11$	$45n + 25$	$16n + 12$	$32n^2 + 19n + 19$
$5n$	$\{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1, 2^{n+1} + 1\}$ [16]	5	$(2^{n+1} + 1)$	$8n + 19$	$62n + 36$	$16n + 28$	$40n^2 + 25n + 72$
$6n + 1$	$\{2^n + 1, 2^n - 1, 2^{2n}, 2^{2n+1} - 1\}$ [19]	4	$(2^{2n+1} - 1)$	$16n + 8$	$52n + 25$	$32n + 9$	$80n^2 + 20n + 19$

4.1.2 The most efficient set for each dynamic range requirement

In order to find out the effect of moduli number on the system's performance and determine the most efficient set for each DR, I have calculated the delay of each component based on each of the studied sets for three precise DRs: medium (12 bits), large (24 bits) and very large (60 bits), as illustrated in Fig. 4.1, Fig. 4.2 and Fig. 4.3, respectively.

In each set, n has been chosen in order to provide the required DR. For example, for DR = 12 bits, n was (4, 3, 3, 2) for sets with DR ($3n$, $4n$, $5n$, $6n$). However, in some cases there was some inconsistency (e.g. for sets with DR = $5n$, n = 3 provides a DR greater than the required 12 bits). Nevertheless, as mentioned in [76], I have dealt with this issue and estimated the approximate delay for the required DR.

Not all sets are illustrated in the graphs, due to the reason that some of these sets can only be used with even values of n ([10], [11]-I, [16], [17] and [18]) or odd values of n ([11]-II and [14]), which does not fit the chosen value of n in order to acquire the required DR.

Fig. 4.1 shows the delays of the reverse converters, modular adders and modular multipliers based on each of the sets [4], [6] – [9], [11] – [15], [19] and [20], for DR = 12 bits (medium DR). In order to acquire this DR, n was chosen (4, 3, 3, 2) for sets with DR ($3n$, $4n$, $5n$, $6n$), respectively.

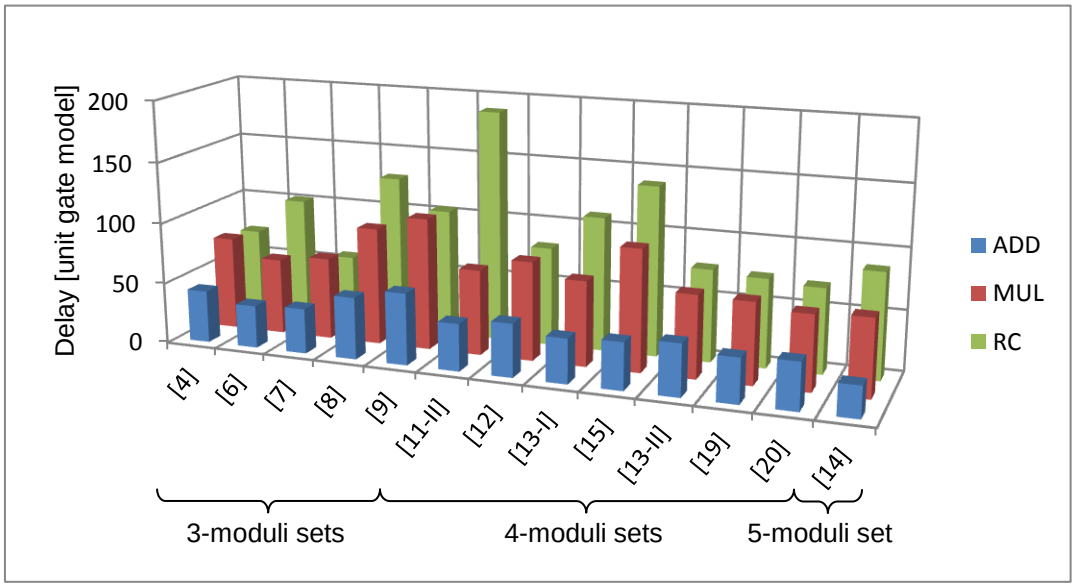


Fig. 4.1: The delay of each basic component based on the moduli sets for DR = 12 bits (medium DR)

As illustrated in Fig. 4.1, the adder with least delay was the one based on the five-moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^n - 2^{(n+1)/2} + 1, 2^n + 2^{(n+1)/2} + 1\}$ [14]. However, the unexpected thing was that the second fastest adder is the one based on the three-moduli set $\{2^{n-1} - 1, 2^n - 1, 2^n\}$

[6]. The fastest multipliers were the ones based on the three-moduli set $\{2^{n-1} - 1, 2^n - 1, 2^n\}$ [6] and four-moduli set $\{2^{2n+1}, 2^{2n} + 1, 2^n + 1, 2^n - 1\}$ [20]. Whereas, the slowest ones were based on the three-moduli set $\{2^n - 1, 2^n + 1, 2^{2n} + 1\}$ [9] and the four-moduli set $\{2^n - 1, 2^n + 1, 2^{2n} - 2, 2^{2n+1} - 3\}$ [15]. Concerning the reverse converters, the fastest one is $\{2^n - 1, 2^n, 2^{n+1} - 1\}$ [7] and the slowest one is based on the four-moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} + 1\}$ [11]-II.

Considering the delay of the three components, it is obvious that the three-moduli set $\{2^n - 1, 2^n, 2^{n+1} - 1\}$ [7] and the four-moduli set $\{2^{2n+1}, 2^{2n} + 1, 2^n + 1, 2^n - 1\}$ [20] are the most efficient, since the components based on these sets have relatively small delays. Thus, for DR = 12 bits, we see that the number of moduli does not affect the overall speed of the system.

Fig. 4.2 shows the delays of the RCs, modular adders and multipliers based on each of the sets [4], [6] – [9], [11] – [15] and [18] – [20], for DR = 24 bits (large DR). In order to acquire this DR, n was chosen (8, 6, 5, 4) for sets with DR ($3n$, $4n$, $5n$, $6n$), respectively.

Fig. 4.2 shows that the delay trends of the basic components are similar to those in DR = 12 bits. The fastest adder is the one based on the five-moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^n - 2^{(n+1)/2} + 1, 2^n + 2^{(n+1)/2} + 1\}$ [14]. The four-moduli set $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$ [11]-I has the fastest multiplier and one of the best adders. However, its RC is the worst. The fastest RC is the one based on the three-moduli set $\{2^n - 1, 2^n, 2^{n+1} - 1\}$ [7].

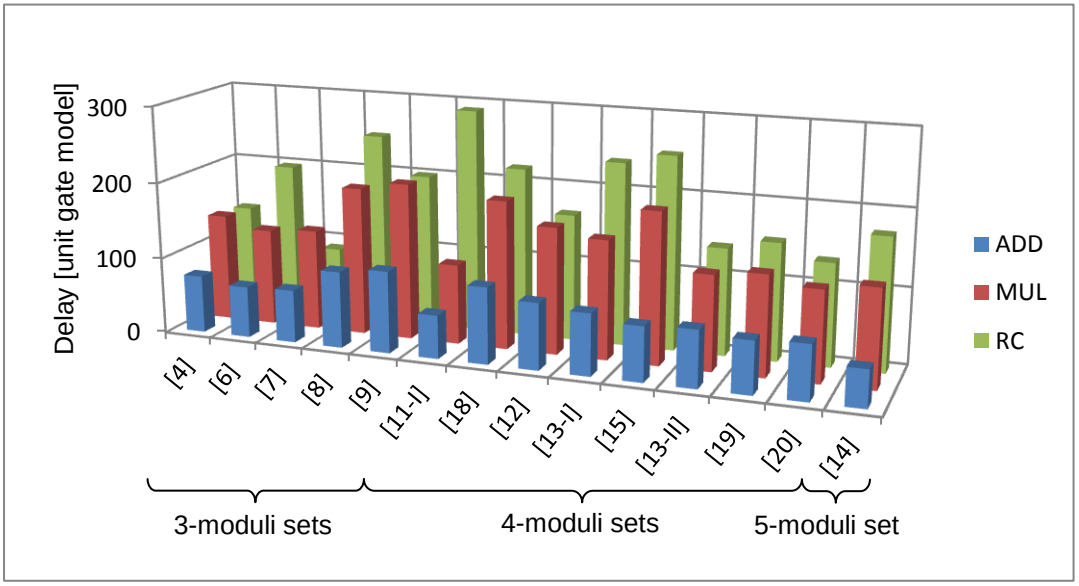


Fig. 4.2: The delay of each basic component based on the moduli sets for DR = 24 bits (large DR)

Considering the delay of the three components, we can say that the most efficient moduli sets for this DR = 24 bits are the three-moduli sets $\{2^n - 1, 2^n, 2^n + 1\}$ [4] and $\{2^n - 1, 2^n, 2^{n+1} - 1\}$ [7], and the four-moduli sets $\{2^n - 1, 2^{2n}, 2^n + 1, 2^{2n} + 1\}$ [13]-II and $\{2^{2n+1}, 2^{2n} + 1, 2^n +$

1, $2^n - 1$} [20], since the three components based on these sets have relatively small delays. Again, it is clear that five-moduli set {$2^n - 1$, 2^n, $2^n + 1$, $2^n - 2^{(n+1)/2} + 1$, $2^n + 2^{(n+1)/2} + 1$} does not show impressive timing performance.

Fig. 4.3 shows the delays of the RCs, modular adders and multipliers based on each of the sets [4], [6] – [13], [15] – [17], [19] and [20], for DR = 60 bits (very large DR). In order to acquire this DR, n was chosen (20, 15, 12, 10) for sets with DR ($3n$, $4n$, $5n$, $6n$), respectively.

Fig. 4.3 shows that the fastest adder and multiplier are the ones based on the five-moduli set {$2^n - 1$, 2^n, $2^n + 1$, $2^{n-1} - 1$, $2^{n+1} + 1$} [16]. However, their RC is the slowest one. The fastest RC is the one based on the three-moduli set {$2^n - 1$, 2^n, $2^{n+1} - 1$} [7]. This set also has rather fast adder and multiplier.

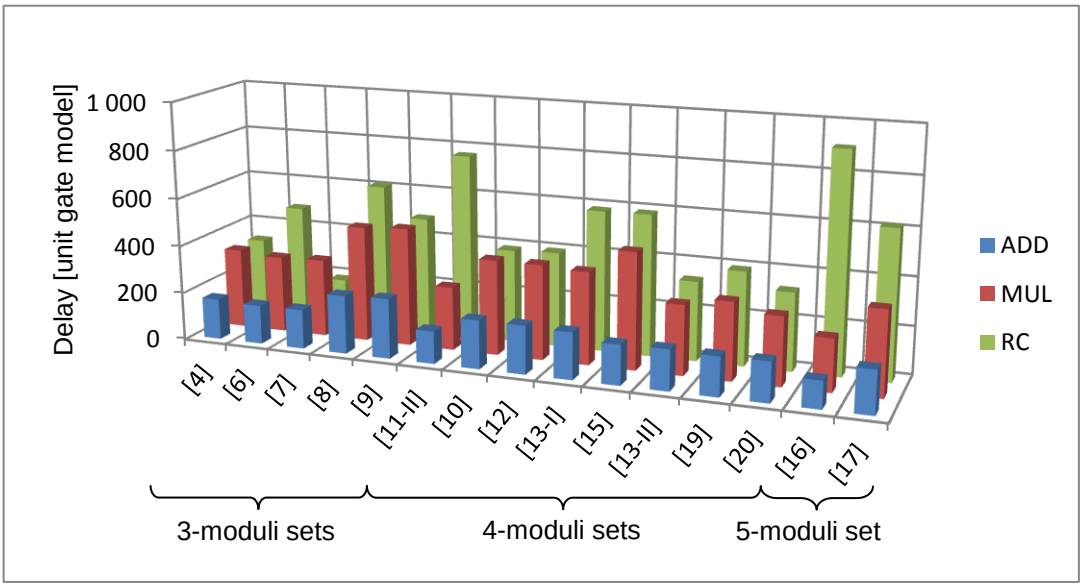


Fig. 4.3: The delay of each basic component based on the moduli sets for DR = 60 bits (very large DR)

Considering the delay of the three components, we can say that the most efficient moduli sets for this DR = 60 bits are the three-moduli sets {$2^n - 1$, 2^n, $2^n + 1$} [4] and {$2^n - 1$, 2^n, $2^{n+1} - 1$} [7], and the four-moduli sets {$2^n - 1$, 2^{2n}, $2^n + 1$, $2^{2n} + 1$} [13]-II and {2^{2n+1}, $2^{2n} + 1$, $2^n + 1$, $2^n - 1$} [20], as the components based on these sets have relatively small delays.

Since the most competent sets are not the five-moduli ones for all three DRs and the three moduli sets [4] and [7] showed the best timing performance concerning all the three components, we conclude that the number of moduli does not affect that much the overall delay of the system considering all its components. There is no point for choosing a five-moduli set if the overall timing performance will be worse than that based on three or four-moduli sets.

A comparison between the most efficient sets for different dynamic ranges based on timing performance [76] and delay × complexity ratio [75] is illustrated in Tab. 4.3. In [75], the most efficient set for medium DR, in terms of (delay × complexity) ratio, is the three-moduli set [6]. Whereas in [76], the most efficient set for the same DR, in terms of timing performance, is the three-moduli set [7]. Regarding the large dynamic range, the most efficient set in [75] is the four-moduli set [11]-I, whereas, it is the three-moduli set [7] in [76]. In a similar manner, the most competent set with the relatively best (D × C) ratios for very large DR was the four moduli set [13]-II. In [76], the most efficient set for the same DR is again the three-moduli sets [7].

It is obvious that five-moduli sets were not mentioned in the above comparison. These sets show better timing performance in medium and large DRs than that in very large DR. Although their RAUs were of the best ones for the very large DR, their RCs were the worst. According to this research, the unexpected issue I have ascertained is that five-moduli sets do not show any superiority over other sets taking into account the three components of RNS (modular adders, modular multipliers and RCs).

Tab. 4.3: Comparison between best moduli sets based on (D × C) [75] and timing performance [76]

Dynamic range	**Best moduli sets based on (D × C) ratio [75]**		**Best moduli sets based on timing performance [76]**	
	moduli set	**moduli #**	**moduli set**	**moduli #**
Medium (12 bits)	$\{2^{n-1}-1, 2^n-1, 2^n\}$ [6]	3	$\{2^n-1, 2^n, 2^{n+1}-1\}$ [7]	3
Large (24 bits)	$\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1\}$ [11]-I	4	$\{2^n-1, 2^n, 2^{n+1}-1\}$ [7]	3
Very large (60 bits)	$\{2^n-1, 2^{2n}, 2^n+1, 2^{2n}+1\}$ [13]-II	4	$\{2^n-1, 2^n, 2^{n+1}-1\}$ [7]	3

4.2 Proposed forward converter

Due to the fact that binary to residue converters are rather simple, little work has been dedicated to enhance their performance. Since my research dealt with special moduli sets rather than general moduli sets, the utilized components to obtain residues with respect to the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ are presented in this section.

Since the majority of moduli, within the moduli sets aforementioned in Section 4.1, have one of the following forms $(2^k - 1)$, (2^k) or $(2^k + 1)$, thus, the illustrated forward converters can be used to obtain the RNS representation with respect to any of those sets.

The most straightforward residue to obtain is the one with respect to modulo 2^n. According to equation (2.3), this residue represents the least n bits of the binary number. Thus, no adders or any logical components are needed.

However, according to equation (2.2), computing a residue with respect to modulo $(2^n - 1)$, demands two consecutive modulo $(2^n - 1)$ adders, as illustrated in Fig. 2.1. Instead of using this structure, a carry save adder with end around carry (CSA-EAC) followed by a carry propagate adder with end around carry (CPA-EAC) can perfectly fulfill the task. This structure is shown in Fig. 4.4.

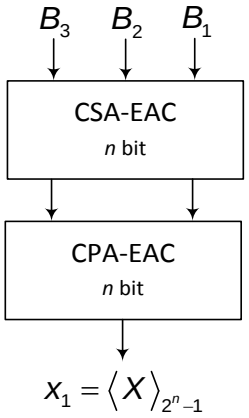


Fig. 4.4: Modulo $(2^n - 1)$ channel of the binary to residue converter

The most difficult residue to obtain is the one with respect to $(2^n + 1)$ modulo. According to equation (2.4), this one requires a modulo $(2^n + 1)$ subtractor followed by a modulo $(2^n + 1)$ adder, as illustrated in Fig. 2.1. This structure is rather complicated, since both components are complex and time consuming.

As stated in Section 2.2.1,

$$\langle x_3 \rangle_{2^n+1} = \langle X \rangle_{2^n+1} = \langle B_1 2^{2n} + B_2 2^n + B_3 \rangle_{2^n+1} = \langle B_1 - B_2 + B_3 \rangle_{2^n+1} \tag{4.1}$$

According to equation (2.19), $\langle -B_2 \rangle_{2^n+1}$ can be rewritten as follows,

$$\langle -B_2 \rangle_{2^n+1} = \langle 2^n + 1 - B_2 \rangle_{2^n+1} \tag{4.2}$$

Based on the two's complement properties,

$$\langle 2^n + 1 - B_2 \rangle_{2^n+1} = \langle 2^n + 1 + \overline{B_2} + 1 \rangle_{2^n+1} = \langle 2^n + \overline{B_2} + 2 \rangle_{2^n+1} \tag{4.3}$$

where, $\overline{B_2}$ refers to the one's complement of B_2.

Hence, the structure of the component that computes the residue with respect to modulo $(2^n + 1)$ can be realized using two parallel binary adders followed by a modulo $(2^n + 1)$ adder. The structure is shown in Fig. 4.5.

Since the adder that adds $"100\cdots010" + \overline{B_2}$ has one constant operand, this structure can be further simplified. According to the full adder equations for obtaining the sum and carry-out,

$$sum = a \oplus b \oplus cin \quad , \quad cout = (a \cdot b) + (b \cdot cin) + (a \cdot cin) \tag{4.4}$$

where, *a*, *b*, *cin*, *sum* and *cout* refer to the two addends, carry-in, sum of the two addends and carry-out, respectively. The width of each of these operands is 1 bit. $(\oplus, \cdot, +)$ refer to the logical gates, *XOR*, *AND*, *OR*, respectively.

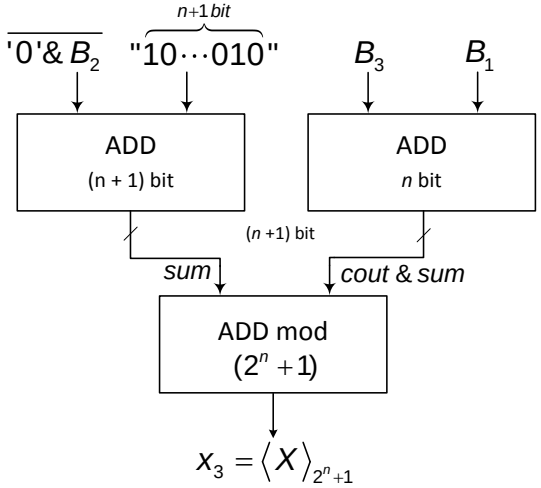


Fig. 4.5: Proposed component for computing a residue with respect to modulo $(2^n + 1)$ channel of the binary to residue converter

By replacing each bit of the second addend by its corresponding of $"100\cdots010"$,

$$\overline{'0'\&B_2} + "10\cdots010" = '1'\&\overline{B_2} + "10\cdots010" \tag{4.5}$$

$$\begin{aligned} &sum_0 = \overline{B_{2,0}} \quad , \quad cout_0 = 0 \\ &sum_1 = B_{2,1} \quad , \quad cout_1 = \overline{B_{2,1}} \\ &sum_i = B_{2,i} \oplus cout_{i-1} \quad , \quad cout_i = \overline{B_{2,i}} \cdot cout_{i-1} \quad ; \quad i \in [2, n-1] \\ &sum_n = cout_{n-1} \quad , \quad cout_n = 1 \end{aligned} \tag{4.6}$$

where, $B_{2,i}$ refers to the i^{th} bit of B_2.

From equations (4.6), it is obvious that the (n + 1) full adders can be replaced by (n – 2) half adders. However, this simplification does not reduce the delay (due to the second adder that adds $B_1 + B_3$), but the overall hardware complexity decreases.

The proposed forward converter along with pure ROM-based one has been implemented on Virtex-4 XC4VSX25 FPGA. The proposed design was implemented for different dynamic range requirements (12 bits, 15 bits, 24 bits and 33 bits). Timing performance of the proposed design was very impressive. The maximum frequency of this converter was (353.4 MHz, 292.8 MHz, 275.8 MHz and 231.3 MHz) for (n = 4, 5, 8 and 11), respectively.

A ROM-based converter was also implemented on Virtex-4 FPGA. However, due to the lack of the integrated BRAM count, this converter could only be implemented for two dynamic ranges (12 bits and 15 bits). The maximum frequency of this design was (383.4 MHz and 258.1 MHz) for (n = 4 and 5), respectively. However, the unexpected issue that has been observed is, that timing performance of the combinational converter for dynamic range = 15 bits is better than the ROM-based one by 13.4%.

Therefore, for large dynamic range requirements, ROM-based converters are not efficient to be implemented (at least on this FPGA device), due to the lack of the integrated BRAM count. Moreover, using external ROMs is not preferable, since they are considerably slower than the built-in ones.

Thus, for applications that require small dynamic ranges, e.g. digital image processing, ROM-based converters are sufficient and are able to provide better performance than combinational ones. However, for applications that require larger dynamic ranges, combinational forward converters are preferable to be used.

4.3 Proposed residue arithmetic units

The proposed residue arithmetic units including modular adders, modular subtractors and modular multipliers are introduced in this section. All proposed designs can be used with any modulo of the form ($2^k \pm 1$), hence, they can be used with majority of the moduli sets mentioned in Section 4.1. The proposed designs that deal with complex operations in the RNS are stated after the reverse conversion section, since these operations are partially based on this process. The proposed designs have been published in different international conferences and journals [77] – [80].

4.3.1 Proposed modular adders

This section contains the proposed structures of modular adders. Three modular adders were proposed; two of them are specified for modulo ($2^n + 1$), and one for modulo ($2^n - 1$).

Modulo (2^n) adders, as mentioned before, have the simplest structures. They can be realized using an *n*-bit binary adder with ignored carry-out. Therefore, my research is focused on modulo ($2^n \pm 1$) adders.

Modulo ($2^n - 1$) adder

Majority of the published structures of modulo ($2^n - 1$) adder perform addition first, and then apply the necessary correction, in order to get the correct result that corresponds to this modulo. The standard structure of this adder depends on two binary adders and a multiplexer. However, the proposed modular adder employs the prefix adders' concept in order to pre-calculate the carry-out needed for the correction process. Hence, this proposed adder uses only one binary adder and a prefix carry-out computation unit. That is why it is considered more time/area efficient. This design has been published in an international conference in Brno [77] and an extended version has been published in ElectroScope journal [78].

Assuming x and y are two modulo ($2^n - 1$) residues of n bits. Their modulo ($2^n - 1$) addition is defined by,

$$\langle x+y \rangle_{2^n-1} = \begin{cases} x+y & if \quad x+y < 2^n-1 \quad \Rightarrow \quad x+y+1 < 2^n \\ x+y+1 & if \quad x+y \geq 2^n-1 \quad \Rightarrow \quad x+y+1 \geq 2^n \end{cases} \tag{4.7}$$

Therefore, if the carry-out of ($x + y + 1$) is computed in advance, then this carry-out can be fed into a binary adder in order to compute the correct result.

The prefix carry computation depends on carry generating and propagating signals denoted as G, P respectively.

$$g(i) = x(i) \; and \; y(i) \qquad , \qquad p(i) = x(i) \; xor \; y(i) \tag{4.8}$$

$$\begin{aligned} c(i+1) = {} & g(i) + p(i)\cdot g(i-1) + p(i)\cdot p(i-1)\cdot g(i-2) + \ldots + \\ & + p(i)\cdot p(i-1)\cdots p(1)\cdot g(0) + \\ & + p(i)\cdot p(i-1)\cdot p(i-2)\cdots p(1)\cdot p(0)\cdot C_{-1} \end{aligned} \tag{4.9}$$

where, $x(i)$, $y(i)$ is the i^{th} bit of the n-bit residues x, y, respectively. C_{-1} is the carry-in, $c(i)$ is the carry from the i^{th} to the $(i + 1)^{th}$ bit, and (+, •) refer to the logical operators: *inclusive OR*, *AND*, respectively.

By using equations (4.8) and (4.9), the carry-out can be calculated as follows,

$$Cout = \sum_{k=n-1}^{0} \left[\prod_{i=k+1}^{n-1} p(i) \right] \cdot g(k) + \left[\prod_{i=n-1}^{0} p(i) \right] \cdot C_{-1} \tag{4.10}$$

where, (Σ, Π) refer to a sequence of logical operators: *inclusive OR*, *AND*, respectively. g, p are carry generate and carry propagate signals, respectively, computed by equation (4.8). $C_{-1} = 1$. The proposed adder's structure is illustrated in Fig. 4.6. In this figure, thick lines refer to n-bit buses, while thin lines refer to 1 bit.

As illustrated in Fig. 4.6, this design contains only one binary adder and a carry-out computation unit, instead of two adders and a multiplexer as stated in [26]. This decreases time and area consumptions in the FPGA, especially when using a carry propagate adder (CPA), due to its features when implemented on FPGAs. The FPGA has dedicated carry ripple logic built-in FPGA [69].

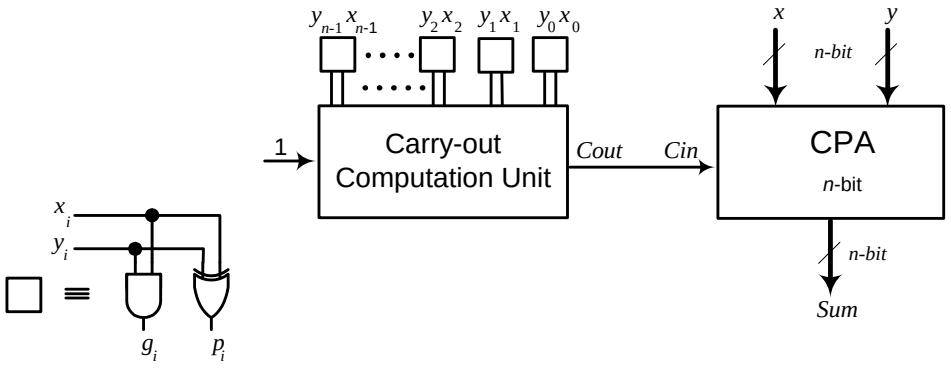


Fig. 4.6: Proposed modulo $(2^n - 1)$ adder – based on prefix carry-out computation [78]

The proposed adder was compared with an already published design [26], which was denoted as (f). The choice of this adder (f) has been done based on its superiority over other adders stated in [26]. Both adders were implemented on Spartan-3 xc3s200 FPGA. According to the implementation results stated in Tab. 4.4 and Tab. 4.5, the proposed adder has proven its superiority, with savings up to (14.7%, 14.3%) in time, area consumptions, respectively.

Tab. 4.4: Comparison between the proposed modulo $(2^n - 1)$ adder with its counterpart (f) in [26] in terms of critical path delay [ns]

n	Dynamic range	Proposed	(f) in [26]	Improvements%
4	12 bits (medium DR)	10.5	12.2	13.9%
8	24 bits (large DR)	15.4	17.5	12%
16	48 bits (very large DR)	23.2	27.2	14.7%

Tab. 4.5: Comparison between the proposed modulo $(2^n - 1)$ adder with its counterpart (f) in [26] in terms of area consumption [slices]

n	Dynamic range	Proposed	(f) in [26]	Savings %
4	12 bits (medium DR)	6	7	14.3%
8	24 bits (large DR)	13	15	13.3%
16	48 bits (very large DR)	29	31	6.5%

Modulo $(2^n + 1)$ adder

It is well known, that modulo $(2^n + 1)$ channel is the most time consuming among all other moduli channels, due to the $(n + 1)$-bit operands and results that this channel deals with. That is why; the major part of my research on residue arithmetic units was concentrated on modulo $(2^n + 1)$ arithmetic.

Two different architectures of modulo $(2^n + 1)$ adder were designed. Both adders use normal binary representation instead of diminished-one representation that has two main problems: difficulties in zero representation, and the necessity to converters that convert from/to diminished-one representation. Therefore, I have focused on acquiring the benefits of both representations, i.e. how to speed up the computation process and not face the difficulties in diminished-one representation.

Simple modulo $(2^n + 1)$ adder - by using only n-bit circuits

The structure of this adder is an improved version of that published in an international conference in Brno [79]. The feature of this design is the usage of only *n*-bit circuits instead of $(n + 1)$-bit. In other words, this design uses normal binary representation and at the same time utilizes just *n*-bit circuits, thus, it has the benefits of the two representation methods simultaneously.

Assuming x and y are two modulo $(2^n + 1)$ residues; ((n + 1)-bit numbers). Modulo $(2^n + 1)$ addition of them is defined as,

$$\langle x+y\rangle_{2^n+1}=\begin{cases}x+y & if \quad x+y<2^n+1\\ x+y-2^n-1 & if \quad x+y\geq 2^n+1\end{cases} \tag{4.11}$$

The structure of this adder encloses an n-bit binary adder, an n-bit binary subtractor and a multiplexer. The output is obtained by separately processing the first n bits and the MSBs of the operands. The structure of this adder is illustrated in Fig. 4.7. Again, thick lines refer to n-bit buses, whereas thin lines refer to 1 bit.

This adder was compared with the second proposed modulo $(2^n + 1)$ adder – based on prefix carry computation. This comparison is stated at the end of the next section.

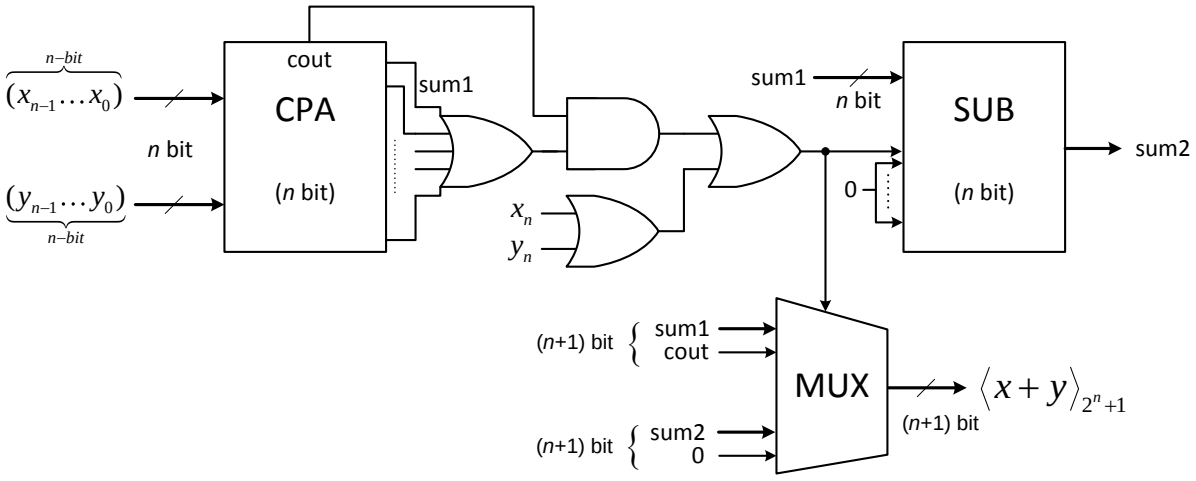


Fig. 4.7: Improved structure of the proposed modulo $(2^n + 1)$ adder – that uses n-bit components [79]

Modulo $(2^n + 1)$ adder – based on prefix carry computation

Contrary to the previously proposed modulo $(2^n + 1)$ adder, this one consists of $(n + 1)$-bit circuits. However, it utilizes the concept of prefix carry computation used in parallel prefix adders in order to speed-up the computation process. This modular adder has been published in an international conference in Brno [77] and an extended version has been published in ElectroScope journal [78]. This adder depends on prefix carry computation, in order to compute the n^{th} bit (MSB) of $(x + y – 1)$.

From equation (4.11),

$$\langle x+y\rangle_{2^n+1}=\begin{cases}x+y & if \quad x+y<2^n+1 \quad \Rightarrow \quad x+y-1<2^n\\ x+y-2^n-1 & if \quad x+y\geq 2^n+1 \quad \Rightarrow \quad x+y-1\geq 2^n\end{cases} \tag{4.12}$$

As can be seen in equation (4.12), the structure of the proposed adder depends on the prefix carry computation, in order to compute the n^{th} bit (MSB; most significant bit) of $(x + y - 1)$. This bit cannot be computed the same way as in the proposed modulo $(2^n - 1)$ adder, since the operands (inputs) here are represented using $(n + 1)$ bits, therefore the carry-out from the $(n - 1)^{th}$ bit to the n^{th} bit will not be sufficient. MSB of the sum of $(x + y - 1)$ can be calculated as follows,

$$MSB = n^{th}\ bit = x(n) \oplus y(n) \oplus c(n-1) \tag{4.13}$$

where, $x(n)$, $y(n)$ represent the n^{th} bit of x, y, respectively. $\oplus$ refers to the logical operator *XOR*. $c(n - 1)$ is the carry from the $(n - 1)^{th}$ bit to the n^{th} bit, and can be computed by equation (4.9).

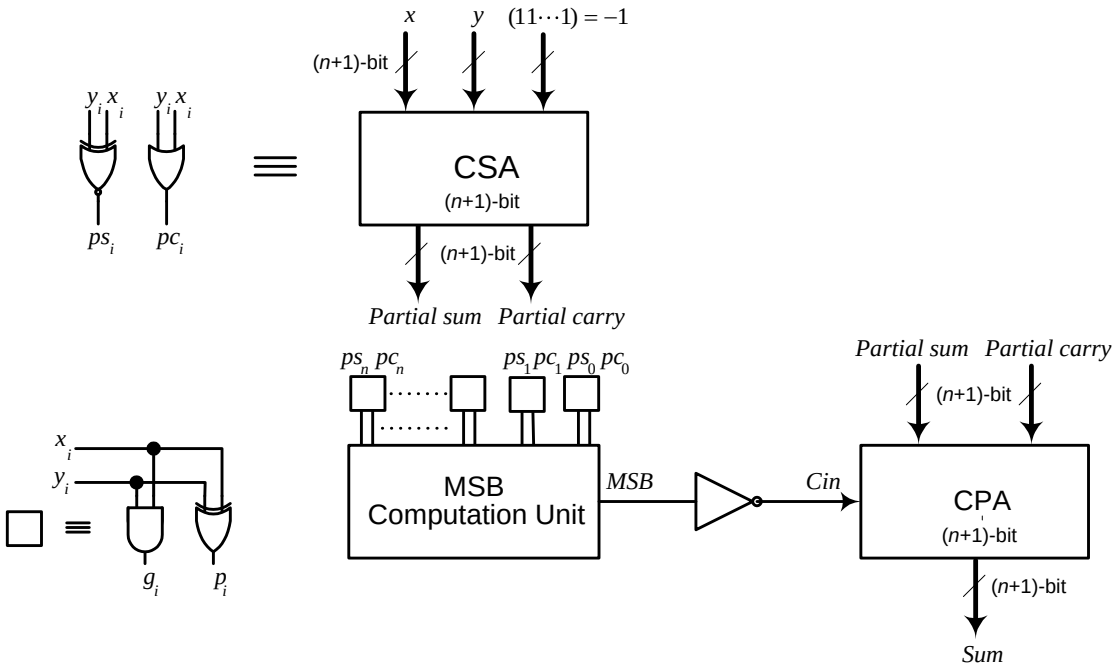


Fig. 4.8: Proposed modulo $(2^n + 1)$ adder - based on the prefix computation [78]

The structure of the proposed adder is illustrated in Fig. 4.8. It consists of a CSA that has a delay equal to that of a half-adder (HA) instead of a full-adder (FA), a prefix computation unit of MSB and a CPA. As mentioned before, the CPA was chosen due to its features when implemented on Spartan-3 FPGA. Hence, the main concept of this adder is based on the prefix computation of the MSB of $(x + y - 1)$, and then applying the necessary correction. This correction is represented in applying the correct carry-in into the CPA.

To prove the efficiency of this adder, it was compared with another already published one, which was published in [26] and denoted as (k). This Modular adder (k) was chosen due

to its superiority over other modular adders stated in [26]. However, this adder produces an increased-by-one result. Therefore, an additional component has been added, in order to make it produce the correct sum.

Both adders were implemented on Spartan-3 xc3s200 FPGA. The implementation results showed savings up to (37.5%, 13.3%) in time, area consumptions, respectively. These savings, as well as the detailed consumptions for different values of *n* are illustrated in Tab. 4.6 and Tab. 4.7.

Tab. 4.6: Comparison between the proposed modulo $(2^n + 1)$ adder with its counterpart (k) in [26] in terms of critical path delay [ns]

n	**Dynamic range**	**Proposed**	**(*k*) in [26]**		**Improvements%**	
			x + y + 1	***x + y***	***x + y + 1***	***x + y***
4	12 bits (medium DR)	14.4	13.5	15.5	-	7.1%
8	24 bits (large DR)	15.7	24.3	25.1	35.4%	37.5%
16	48 bits (very large DR)	25.8	39.6	40.9	34.9%	36.9%

Tab. 4.7: Comparison between the proposed modulo $(2^n + 1)$ adder with its counterpart (k) in [26] in terms of area consumption [slices]

n	**Dynamic range**	**Proposed**	**(*k*) in [26]**		**Savings %**	
			x + y + 1	***x + y***	***x + y + 1***	***x + y***
4	12 bits (medium DR)	13	8	15	-	13.3%
8	24 bits (large DR)	24	16	26	-	7.7%
16	48 bits (very large DR)	50	32	51	-	1.2%

Concerning comparing the two proposed modulo $(2^n + 1)$ adders; the one that uses *n*-bit components [79] with the one based on the prefix computation [78]. The implementation results showed that [79] is superior in terms of area consumption, whereas [78] is superior in terms of time consumption (due to the usage of a CSA that has a delay equal to that of a HA, and the usage of a prefix carry computation). However, both proposed adders have better performance than that of adder (k) in [26] in terms of time and area consumptions.

4.3.2 Proposed modular subtractor

This section introduces the proposed structure of a modulo (2^n + 1) subtractor. As aforementioned in Section 2.3.2, a modulo (2^n – 1) subtractor can be simply realized using a modulo (2^n – 1) adder and a few inverters. Therefore, only a modulo (2^n + 1) subtractor has been proposed. It has been published in an international conference in St. Maarten, the Netherlands Antilles [80]. However, the presented structure is further improved than the one stated in [80]. This subtractor was intentionally designed to be used in the proposed modulo (2^n + 1) multiplier, which has been published in the same paper [80], as will be described later.

Assuming *x* and *y* are two modulo (2^n + 1) residues; ((*n* + 1)-bit numbers). Modulo (2^n + 1) subtraction of them is defined by,

$$\langle x-y\rangle_{2^n+1}=\begin{cases} x-y & if \quad x-y\geq 0 \\ x-y+2^n+1 & if \quad x-y<0 \end{cases} \tag{4.14}$$

The structure of the improved modular subtractor consists of an (*n* + 1)-bit binary subtractor followed by an (*n* + 1)-bit binary adder, as shown in Fig. 4.9. The output "*Bout*" refers to the borrow-out, that indicates whether the difference is less than zero or not. In case of a negative result (*Bout* = 1), modulo (2^n + 1) should be added back, in order to correct the result. This subtractor has been efficiently utilized in the proposed modulo (2^n + 1) multiplier.

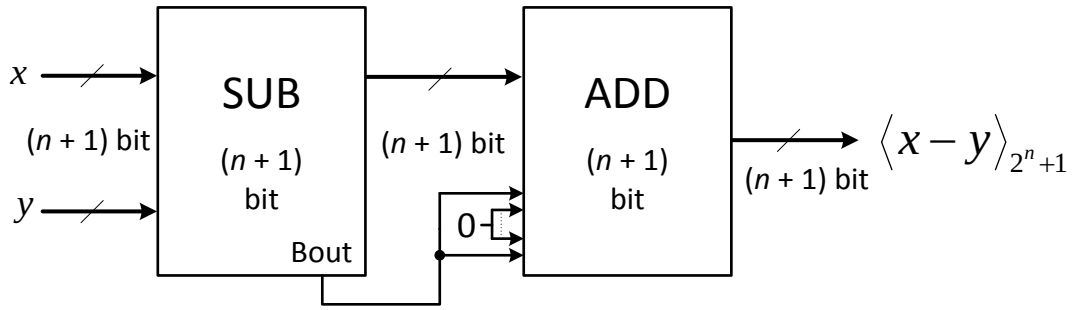


Fig. 4.9: Improved structure of proposed modulo (2^n + 1) subtractor

4.3.3 Proposed modular multipliers

The proposed structures of modulo (2^n ± 1) multipliers are presented in this section. The first part contains a comparison between two structures of a modulo (2^n – 1) multiplier. Each of these structures belongs to a different category of modular multipliers, mentioned in Section 2.3.3. The second part illustrates the proposed modulo (2^n + 1) multiplier that uses the above mentioned modulo (2^n + 1) subtractor as a fundamental component.

Modulo (2^n – 1) multipliers

A modulo (2^n – 1) multiplier can be designed according to the following equations,

$$z = \langle x \times y \rangle_{2^n-1} = \left\langle \sum_{i=0}^{2n-1} z_i 2^i \right\rangle_{2^n-1} = \left\langle \sum_{i=0}^{n-1} z_i 2^i + \langle 2^n \rangle_{2^n-1} \sum_{i=n}^{2n-1} z_i 2^{i-n} \right\rangle_{2^n-1} \tag{4.15}$$

$$z = \langle a + 2^n \cdot b \rangle_{2^n-1} = \langle a + b \rangle_{2^n-1} \tag{4.16}$$

This means that this modular multiplier consists of an *n*-bit binary multiplier followed by a modulo (2^n – 1) adder. The structure of this multiplier is illustrated in Fig. 4.10.

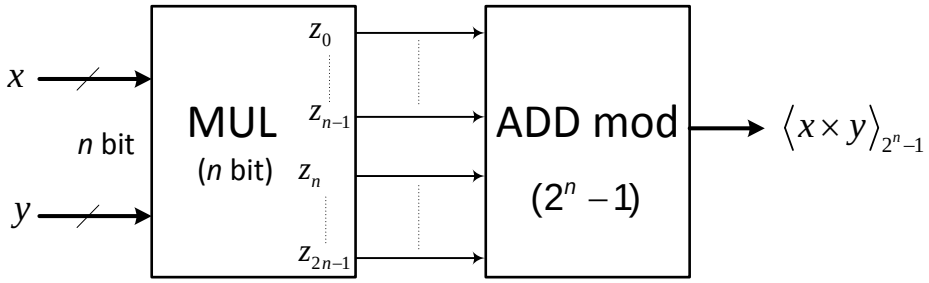


Fig. 4.10: Proposed modulo (2^n – 1) multiplier – based on multiplication-then-reduction approach

This multiplier belongs to the multiplication-then-reduction category mentioned in Section 2.3.3. However, such a multiplier is quite expensive comparing to the one based on Property 2.

Property 2: The multiplication of a residue number *x* by 2^P in modulo (2^n − 1) is carried out by *P* bit circular left shift, where *P* is a natural number [71].

According to the above property and assuming *x* and *y* are two modulo (2^n – 1) residues (*n*-bit numbers). Their modulo (2^n – 1) multiplication can be written as follows,

$$\begin{aligned} \langle x \times y \rangle_{2^n-1} &= \langle (x_{n-1} \cdots x_0) \times (y_{n-1} \cdots y_0) \rangle_{2^n-1} = \left\langle \sum_{P=0}^{n-1} (x_{n-1} \cdots x_0) \times (2^P \times y_P) \right\rangle_{2^n-1} \\ &= \langle (x_{n-1} \cdots x_0) \times y_0 + (x_{n-2} \cdots x_0 x_{n-1}) \times y_1 + \cdots + (x_0 x_{n-1} \cdots x_1) \times y_{n-1} \rangle_{2^n-1} \end{aligned} \tag{4.17}$$

Thus, the structure of this multiplier consists of a rotation unit, a Wallace tree adder to perform multi-operand addition and a modulo (2^n – 1) adder. This structure is shown in Fig. 4.11. Contrary to the previous design, this one belongs to the interleaving multiplication and reduction category.

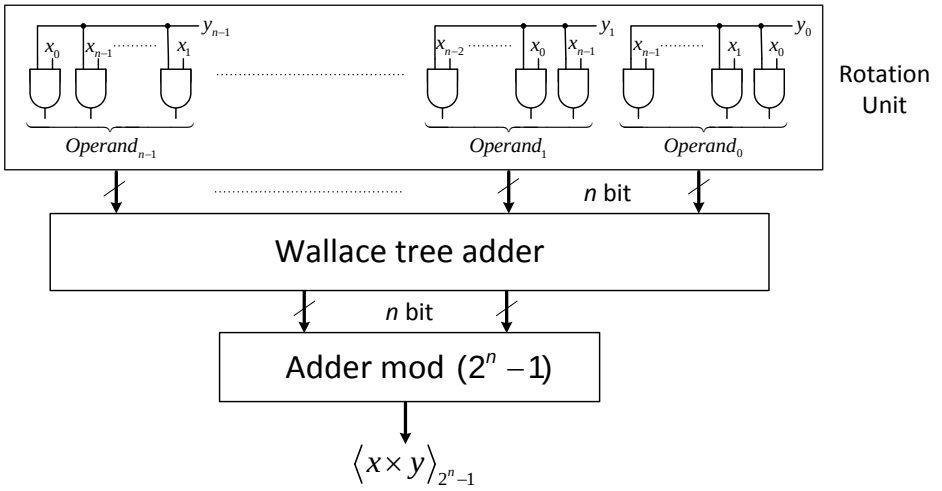


Fig. 4.11: Proposed modulo $(2^n - 1)$ multiplier – based on interleaving multiplication and reduction approach

Tab. 4.8 illustrates time and area improvements when using the structure shown in Fig. 4.11 over the one shown in Fig. 4.10. It is obvious that the second structure shows better timing performance as the dynamic range increases, whereas, area saving gradually decreases. Therefore, for systems with very large dynamic ranges (more than 60 bits) and whose main goal and strategy is "area reduction", the structure shown in Fig. 4.10 is more effective.

Tab. 4.8: Time and area improvements of multiplier's structure Fig. 4.11 over Fig. 4.10

Dynamic range	Time improvements %	Area improvements %
12 bits, $n = 4$	28.1%	10.7%
21 bits, $n = 7$	30.5%	5.8%
30 bits, $n = 10$	34%	3.9%
60 bits, $n = 20$	39.6%	1.9%

Modulo $(2^n + 1)$ multiplier

This modulo $(2^n + 1)$ multiplier has been published along with the previous modular subtractor in [80]. Its structure is based on the multiplication-then-reduction approach mentioned in Section 2.3.3.

Assuming x and y are two modulo $(2^n + 1)$ residues; ($(n + 1)$-bit numbers). Their modulo $(2^n + 1)$ multiplication is defined by,

$$z = x \times y = \sum_{i=0}^{2n} z_i 2^i = \left\langle \sum_{i=0}^{n-1} z_i 2^i + \left\langle 2^n \right\rangle_{2^n+1} \sum_{i=n}^{2n} z_i 2^{i-n} \right\rangle_{2^n+1} \tag{4.18}$$

$$z = \left\langle a + 2^n \cdot b \right\rangle_{2^n+1} = \left\langle a - b \right\rangle_{2^n+1} \tag{4.19}$$

Equation (4.19) was obtained by splitting the product of $x \times y$ into two parts a and b, and then applying equation (2.5), that converts $(+ 2^n \cdot b)$ into $(- b)$.

The necessity to a modulo $(2^n + 1)$ subtractor is clear in equation (4.19). This step performs the reduction process according to modulo $(2^n + 1)$. The structure of the proposed multiplier is illustrated in Fig. 4.12.

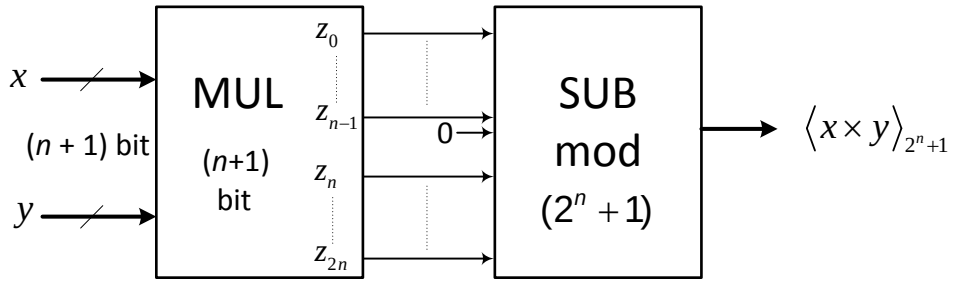


Fig. 4.12: Proposed modulo $(2^n + 1)$ multiplier [80]

The proposed multiplier was compared with an already published modulo $(2^n + 1)$ multiplier [70]. This multiplier was realized using modulo-reduced partial products, a modulo CSA and a modulo CPA. However, the operands and results in this structure are presented using n bits only, where the value 0 is not used, and the value 2^n is presented as "00..0" [70]. Both multipliers were implemented on Spartan-3 FPGA. The implementation results along with the comparison are illustrated in Tab. 4.9.

Tab. 4.9: Comparison between the proposed multiplier and its counterpart in terms of critical path delay [ns]

n	Dynamic range	Proposed multiplier [80]	[70]	Improvements %
4	12 bits	18.3	25	26.8%
8	24 bits	27.5	31	11.3%
11	33 bits	34.9	36	3.1%
12	36 bits	36.8	36.2	-
14	42 bits	43.7	40.6	-

The results showed time saving up to 26.8% for medium dynamic range = 12 bits. However, the proposed design shows better timing performance than its counterpart for dynamic range up to 33 bits. The reason is that, for DR ≥ 36 bits, the delay of the binary

multiplier considerably increases, hence the overall delay of the whole design does. However, for systems with dynamic rages less than 36 bits, the proposed modular multiplier is superior over [70].

4.4 Proposed Reverse converters

This section is dedicated for presenting my work on residue to binary converters. It is divided into two subsections; the first one presents a comparison between two well-known algorithms for residue to binary conversion based on the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$. The comparison is based on FPGA implementation of these two algorithms.

The second part presents a novel algorithm for reverse conversion in the RNS and proposes an efficient reverse converter based on it. This algorithm is dedicated for the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$.

4.4.1 Comparison between the new CRT-I and MRC

This section presents a comparison between two well-known algorithms for residue-to-binary conversion, namely the new CRT-I and MRC. The comparison is done in order to highlight the differences between the two algorithms when implemented on FPGA. Both converters are dedicated for the special moduli set $(2^n - 1, 2^n, 2^n + 1)$. The comparison was carried out in terms of delay and area consumptions on Spartan-3 FPGA. This study has been published in the Electronics journal [83].

The two proposed structures of the reverse converters use Property 1 and Property 2 mentioned in Section 2.3.2 and Section 4.3.3, respectively. By using these two properties, the necessity to multiplication during the reverse conversion process was eliminated.

Proposed structure of residue to binary converter based on the new CRT-I

The new CRT-I is a parallel algorithm, but it requires a special modular adder (a rather large one), in order to compute the equivalent binary number. The following equations illustrate how the multiplication was eliminated and converted into addition. The moduli set that has been used for this converter is ordered as follows,

$$m_1 = 2^n \quad , \quad m_2 = 2^n + 1 \quad , \quad m_3 = 2^n - 1 \tag{4.20}$$

Based on equations (2.13) and (2.14) in Section 2.2.2, the multiplicative inverses k_1, k_2 are computed as follows,

$$\left\langle k_1 \times 2^n \right\rangle_{2^{2n}-1} = 1 \quad \Rightarrow \quad k_1 = 2^n \tag{4.21}$$

$$\left\langle k_2 \times 2^n \times (2^n + 1) \right\rangle_{2^n-1} = 1 \quad \Rightarrow \quad k_2 = 2^{n-1} \tag{4.22}$$

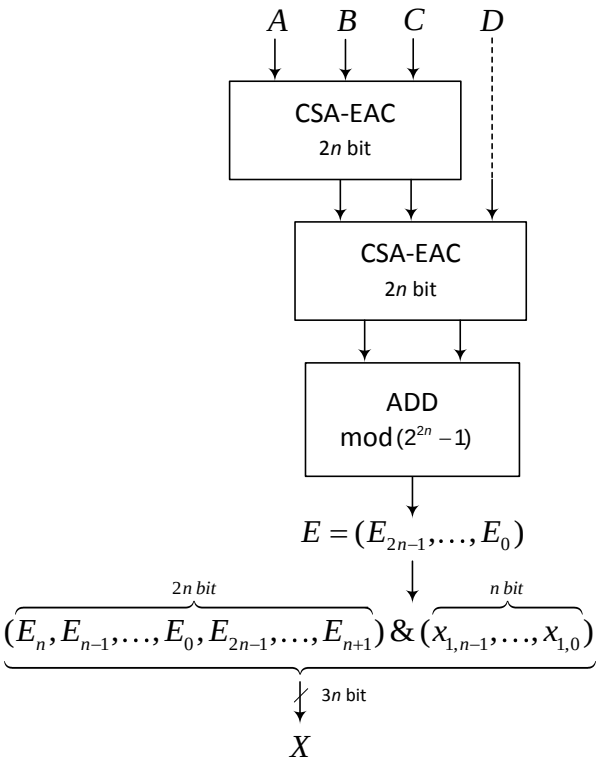


Fig. 4.13: Proposed structure of reverse converter - based on the new CRT-I [83]

By substituting equations (4.20), (4.21) and (4.22) into the main CRT-I equation (2.12), the equivalent binary number X is computed as follows,

$$X = x_1 + 2^n \times \left\langle 2^{n-1}(\underbrace{x_2}_{C} - \underbrace{2x_1}_{B} + \underbrace{2^n x_3 + x_3}_{A} - \underbrace{2^n x_2}_{D}) \right\rangle_{2^{2n}-1} \tag{4.23}$$

By splitting equation (4.23) and according to Property 1 and Property 2,

$$A = 2^n x_3 + x_3 = \underset{n-bit}{x_3} \;\&\; \underset{n-bit}{x_3} \tag{4.24}$$

$$B = \left\langle -2x_1 \right\rangle_{2^{2n}-1} = \left\langle \overline{0...0 \,\&\, x_1 \,\&\, 0} \right\rangle_{2^{2n}-1} = \left\langle \underset{n-1bit}{1...1} \,\&\, \underset{n-bit}{\overline{x_1}} \,\&1 \right\rangle_{2^{2n}-1} \tag{4.25}$$

$$C = \underset{n-1bit}{0...0} \,\&\, \underset{n+1bit}{x_2} \tag{4.26}$$

$$D = \left\langle -2^n \times C \right\rangle_{2^{2n}-1} = \left\langle \overline{C_{n-1},...,C_0,C_{2n-1},...,C_n} \right\rangle_{2^{2n}-1} \tag{4.27}$$

$$E = A + B + C + D \tag{4.28}$$

By substituting equation (4.28) in (4.23),

$$X = x_1 + 2^n \times \left\langle 2^{n-1} \times E \right\rangle_{2^{2n}-1} \tag{4.29}$$

Again, according to Property 2,

$$Z = \left\langle 2^{n-1} \times E \right\rangle_{2^{2n}-1} = \underbrace{E_n, E_{n-1}, ..., E_0, E_{2n-1}, ..., E_{n+1}}_{2n\,bit} \tag{4.30}$$

$$X = \overbrace{\underbrace{Z}_{2n\,bi} \;\&\; \underbrace{x_1}_{n\,bit}}^{3n\,bit} \tag{4.31}$$

According to the above equations, the reverse converter based on new CRT-I requires two (2*n*-bit) CSA-EACs and a modulo (2^{2n} – 1) adder. This modular adder has the same structure as that of the modulo (2^n – 1) adder, but it uses 2*n*-bit circuits instead of *n*-bit. Utilizing CSAs results in better timing performance of the design. The structure of this converter is illustrated in Fig. 4.13.

Proposed structure of residue to binary converter based on the MRC

This section illustrates the second reverse converter based on the MRC algorithm. This method does not require any special large modular adders, as in the previous converter. The same modular adders, which are used in the arithmetic unit, are utilized in this converter.

The below equations are extracted from the MRC main equation, stated in Section 2.2.2. The utilized moduli set for this converter is ordered as follows,

$$m_1 = 2^n + 1, \quad m_2 = 2^n, \quad m_3 = 2^n - 1 \tag{4.32}$$

The order of the moduli within the set is important, since a proper order of the moduli can considerably simplify the multiplicative inverses needed for the reverse conversion process. These multiplicative inverses are computed as follows,

$$\left\langle m_1^{-1} \right\rangle_{m_2} = \left\langle (2^n + 1)^{-1} \right\rangle_{2^n} = 1 \tag{4.33}$$

$$\left\langle m_1^{-1} \right\rangle_{m_3} = \left\langle (2^n + 1)^{-1} \right\rangle_{2^n - 1} = 2^{n-1} \tag{4.34}$$

$$\left\langle m_2^{-1} \right\rangle_{m_3} = \left\langle (2^n)^{-1} \right\rangle_{2^n - 1} = 1 \tag{4.35}$$

The above three equations proved the fact that, a proper ordering of the moduli results in an essential simplification of the multiplicative inverses. Two of the three multiplicative inverses are equal to 1.

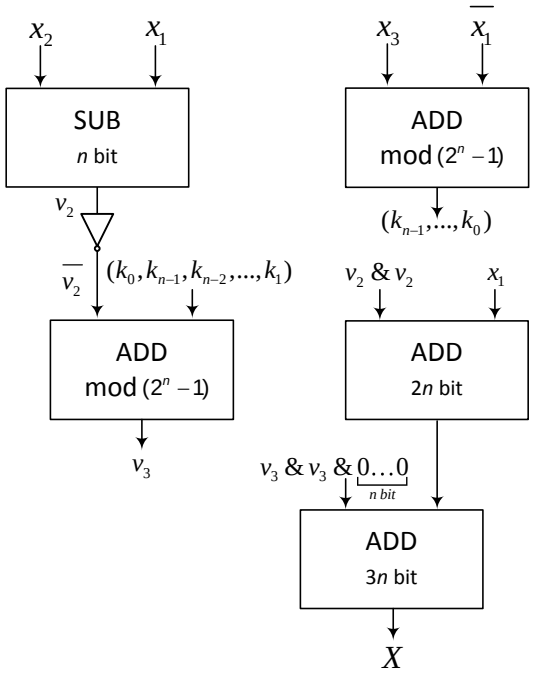


Fig. 4.14: Proposed structure of reverse converter - based on the MRC [83]

By substituting equations (4.33), (4.34) and (4.35) into (2.10), (2.11), and by utilizing Property 1,

$$v_2 = \left\langle (x_2 - x_1)\times 1 \right\rangle_{2^n} = \left\langle x_2 - x_1 \right\rangle_{2^n} \tag{4.36}$$

$$v_3 = \left\langle (x_3 - x_1)\times 2^{n-1} + \overline{v_2} \right\rangle_{2^n-1} \tag{4.37}$$

Hence,

$$X = \underbrace{v_3 \times 2^n \times (2^n+1)}_{t_1} + \underbrace{v_2 \times (2^n+1)}_{t_2} + v_1 \tag{4.38}$$

$$X = t_1 + t_2 + v_1 \tag{4.39}$$

$$t_1 = 2^n \times (v_3 \times 2^n + v_3) = v_3 \,\&\, v_3 \,\&\, \underset{n-bit}{0..0} \tag{4.40}$$

$$t_2 = v_2 \times 2^n + v_2 = \underset{n-bit}{0..0} \,\&\, v_2 \,\&\, v_2 \tag{4.41}$$

$$X = t_1 + t_2 + v_1 = t_1 + t_2 + x_1 \tag{4.42}$$

As inferred from the above equations, the multipliers were eliminated and replaced by concatenations. The structure of this converter is illustrated in Fig. 4.14.

Comparison between the new CRT-I and MRC converters - based on FPGA implementation

The two converters were implemented on Spartan-3 xc3s200 FPGA. The implementation results have proven the theoretical considerations, that the new CRT-I is a parallel algorithm, but has more area consumption due to the usage of two 2*n*-bit CSAs and a large modular adder (modulo $(2^{2n} - 1)$ adder). Although the MRC is a sequential algorithm, it does not require any special large modular adders, which results in less area consumption but longer delay.

Time and area consumptions, for various dynamic range requirements, are stated in Tab. 4.10 and Tab. 4.11 for the new CRT-I and MRC-based converters, respectively.

Tab. 4.10: Comparison between the new CRT-I and MRC-based converters in terms of pad-to-pad delay [ns]

Dynamic range	12 bits	24 bits	48 bits	60 bits
Based on the new CRT-I	6.9	7.1	9.7	9.1
Based on the MRC	11.5	13.2	16.8	17.1
Time improvements % (new CRT-I has shorter delay than MRC by)	40%	46.2%	42.3%	46.8%

Tab. 4.11: Comparison between the new CRT-I and MRC-based converters in terms of area consumption [slices]

Dynamic range	12 bits	24 bits	48 bits	60 bits
Based on the new CRT-I	67	129	253	315
Based on the MRC	30	56	108	134
Area improvements % (MRC has less hardware complexity than new CRT-I by)	55.2%	56.6%	57.3%	57.5%

Both converters were designed based on pure combinational structures, thus, timing performance is illustrated using pad-to-pad delay. Tab. 4.10 shows that the new CRT-I has shorter pad-to-pad delays than those of the MRC. The difference between the two algorithms gradually increases as the dynamic range increases. Time saving percentage of the new CRT-I over the MRC has a maximum value for the very large dynamic range (60 bits).

In a similar manner, Tab. 4.11 shows that the MRC has considerably less hardware complexity than that of the new CRT-I. The difference between hardware complexity of the

new CRT-I and MRC gradually increases as the dynamic range increases. This difference achieves its greatest value (57.5 %) for the very large dynamic range (60 bits).

Based on these percentages shown in Tab. 4.10 and Tab. 4.11, we conclude that the MRC is preferred over the new CRT-I for designs with balanced and minimum-area strategies. Whereas, new CRT-I should be used for designs that have critical timing requirements.

4.4.2 Proposed algorithm for residue to binary conversion

In this section, a novel algorithm for reverse conversion based on the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ is presented. The majority of papers regarding reverse converters are principally based on one of the widespread algorithms; the MRC, the CRT or the new CRTs. The proposed algorithm is simpler and does not require multiplicative inverses neither multiplication. A paper, which presents the proposed algorithm, a reverse converter and a residue comparator based on it, is still under review in IEICE Electronics Express journal.

Proposed algorithm for reverse conversion

This section presents the proposed algorithm for performing residue to binary conversion. This algorithm does not need any multiplicative inverses neither multiplication processes. These calculations always have been the main obstacles in the reverse conversion methods. The proposed algorithm is based on the fact that, the numbers within the dynamic range $[0, M - 1]$ can be divided into $(2^{2n} - 1)$ groups.

Suppose, x_1, x_2 and x_3 are the residues of the binary number X corresponding to the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$. According to Tab. 4.12, the numbers within the dynamic range $[0, M - 1]$ are divided into $(2^{2n} - 1)$ groups.

Tab. 4.12: The Groups within the dynamic range M of the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$

X	$\langle x_1 \rangle_{2^n-1}$	$\langle x_2 \rangle_{2^n}$	$\langle x_3 \rangle_{2^n+1}$
0 $\vdots$ $2^n - 1$	$\langle x_2 + 0 \rangle_{2^n-1}$	$\langle x_2 \rangle_{2^n}$	$\langle x_2 - 0 \rangle_{2^n+1}$
2^n $\vdots$ $(2 \times 2^n) - 1$	$\langle x_2 + 1 \rangle_{2^n-1}$	$\langle x_2 \rangle_{2^n}$	$\langle x_2 - 1 \rangle_{2^n+1}$

(2×2^n) $\vdots$ $(3\times 2^n)-1$	$\langle x_2+2\rangle_{2^n-1}$	$\langle x_2\rangle_{2^n}$	$\langle x_2-2\rangle_{2^n+1}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$(2^{2n}-2)\times 2^n$ $\vdots$ $((2^{2n}-1)\times 2^n)-1$	$\langle x_2+(2^{2n}-2)\rangle_{2^n-1}$	$\langle x_2\rangle_{2^n}$	$\langle x_2-(2^{2n}-2)\rangle_{2^n+1}$

It is obvious, that residues x_1, x_3 corresponding to moduli $(2^n - 1)$, $(2^n + 1)$, respectively, can be directly computed using x_2 and the group that the binary number X belongs to. According to Tab. 4.12, x_1 and x_3 can be computed as follows,

$$x_1 = \langle x_2 + G\rangle_{2^n-1} \quad ; \quad G \in [0, 2^{2n}-2] \tag{4.43}$$

$$x_3 = \langle x_2 - G\rangle_{2^n+1} \quad ; \quad G \in [0, 2^{2n}-2] \tag{4.44}$$

where, G is the group number that the binary number X belongs to.

Afterwards, G will be used in order to acquire the binary number X from its residues x_1, x_2 and x_3. From equations (4.43), (4.44) and congruence properties,

$$\langle g\rangle_{2^n-1} = \langle x_1 - x_2\rangle_{2^n-1} \tag{4.45}$$

$$\langle g\rangle_{2^n+1} = \langle x_2 - x_3\rangle_{2^n+1} \tag{4.46}$$

where, $\left(\langle g\rangle_{2^n-1}, \langle g\rangle_{2^n+1}\right)$ are the residues of G corresponding to moduli $(2^n - 1, 2^n + 1)$, respectively.

Since moduli in equations (4.45) and (4.46) are different, the obtained results $\left(\langle g\rangle_{2^n-1}, \langle g\rangle_{2^n+1}\right)$ might be different, too. Therefore, the following step should be repeated in order to acquire the correct group number.

$$\langle g\rangle_{2^n-1} + k_1(2^n-1) = \langle g\rangle_{2^n+1} + k_2(2^n+1) \tag{4.47}$$

where, $k_1, k_2 = 0,1,\ldots.$

The binary number X can be now easily computed using the group number G and x_2,

$$X = G \times 2^n + x_2 \tag{4.48}$$

Example: Consider n = 3, thus, the moduli set is {7, 8, 9}. The dynamic range of this system is [0, 503]. The number of groups whining this dynamic range is 63, which is [0, 62].

Consider the following RNS number should be converted into its binary equivalent (x_1, x_2, x_3) = (0, 1, 2).

According to equations (4.45) and (4.46),

$$\langle g \rangle_{2^n-1} = \langle x_1 - x_2 \rangle_{2^n-1} = \langle 0-1 \rangle_7 = 6$$

$$\langle g \rangle_{2^n+1} = \langle x_2 - x_3 \rangle_{2^n+1} = \langle 1-2 \rangle_9 = 8$$

Since $\left(\langle g \rangle_{2^n-1}, \langle g \rangle_{2^n+1}\right)$ are different, equation (4.47) should be repeated until we obtain the correct G.

$$k_1 = 8, k_2 = 6 \Rightarrow G = 62$$

Hence, the binary equivalent of (0, 1, 2) is,

$$X = 62 \times 8 + 1 = 497$$

Proposed residue to binary converter based on the proposed algorithm

Generally, the structure of this converter is rather simple; it consists of two parallel modular subtractors followed by a binary (n + 1)-bit subtractor, a 2n-bit binary adder and a read only memory (ROM). The proposed structure is illustrated in Fig. 4.15.

The step that computes G should be repeated many times in order to acquire the correct values of k_1 and k_2, thus obtaining the correct G. In order to get rid of this recurrence step that may consume much time, a ROM was used. This ROM contains the correct value of k_2, according to the values of $\left(\langle g \rangle_{2^n-1}, \langle g \rangle_{2^n+1}\right)$. The values range of $\langle g \rangle_{2^n-1}$ and $\langle g \rangle_{2^n+1}$ are [0, 2^n – 2] and [0, 2^n], respectively. Thus, the values range of $\left(\langle g \rangle_{2^n-1} - \langle g \rangle_{2^n+1}\right)$ is [-2^n, 2^n – 2], which can be represented using (n + 1) bits. The width of k_2 is n bits. Thus, the ROM size is ($2^{n+1} \times n$) bits.

According to equations (4.47) and (4.48),

$$G = \langle g \rangle_{2^n+1} + k_2(2^n+1) = \langle g \rangle_{2^n+1} + k_2 \,\&\, k_2 \tag{4.49}$$

$$X = G \times 2^n + x_2 = G \,\&\, x_2 \tag{4.50}$$

According to Property 1 stated in Section 2.3.2, a modulo $(2^n - 1)$ subtractor can be easily implemented using a modulo $(2^n - 1)$ adder and n invertors. On the other hand, the structure of the modulo $(2^n + 1)$ subtractor is the same illustrated in Fig. 4.9. It consists of an $(n + 1)$-bit subtractor followed by an $(n + 1)$-bit adder.

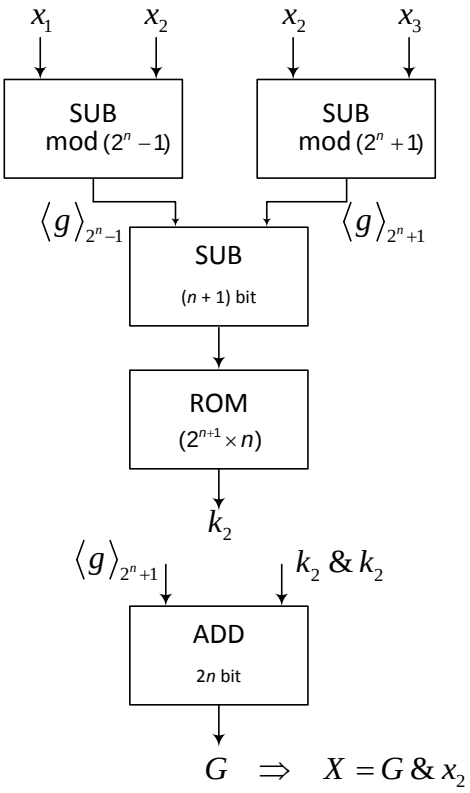


Fig. 4.15: The structure of the reverse converter based on the proposed algorithm [86]

The proposed reverse converter was implemented on Virtex-4 XC4VSX25 FPGA. The ROM containing k_2 values was designed as a single port ROM using Xilinx core generator v.13.4. The proposed design was implemented for different dynamic range requirements, 12 bits, 15 bits (medium dynamic range), 24 bits and 33 bits (large dynamic range), 45 bits and 48 bits (very large dynamic range). The proposed design could not be implemented for dynamic ranges greater than 48 bits, due to the BRAM limitation built-in Virtex-4 XC4VSX25, where the maximum RAM size is 2304 Kb.

A comparison between the proposed reverse converter and another one based on the new CRT-I [4] has been carried out. The implementation results are illustrated in Tab. 4.13. The superiority of the proposed reverse converter timing performance is clear. It can operate at higher frequencies up to 78.5% than the one based on the new CRT-I [4]. Concerning the area consumption, the savings are not very impressive. The total number of the 4-input look-up

tables (4-LUTs) used in the proposed converter is less for the dynamic ranges up to 33 bits. However, this number considerably increases for the very large dynamic ranges. Furthermore, the proposed design uses a number of BRAMS of 18 Kb. Nevertheless, the speed gain makes the proposed converter very attractive for further enhancements and improvements.

The speed gain of the proposed reverse converter is about 23.4% for medium dynamic ranges. Then, it extensively increases to 78.5% for DR = 24 bits. This gain afterwards begins to gradually decrease until it reaches a break point (DR = 45 bits), where the speed gain becomes 0.2%. For DR = 48 bits, timing performance of the proposed reverse converter becomes worse than the new CRT-I based one [4]. However, according to my researches stated in [75] and [76], the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ is not efficient to be used in applications that require very large dynamic ranges.

Tab. 4.13: Comparison between reverse converters for different dynamic range requirements

Dynamic range	New CRT-I [4]			Proposed converter			Time improving %
	Max. freq. [MHz]	4-LUTs	BRAMs [18 Kb]	Max. freq. [MHz]	4-LUTs	BRAMs [18 Kb]	
n = 4, DR = 12 bits (medium DR)	241.6	54	-	296.1	47	1	22.6%
n = 5, DR = 15 bits (medium DR)	232.7	74	-	288.8	50	1	24.1%
n = 8, DR = 24 bits (large DR)	156.8	86	-	279.9	78	1	78.5%
n = 11, DR = 33 bits (large DR)	148.9	118	-	257.3	105	3	72.8%
n = 15, DR = 45 bits (very large DR)	136.3	132	-	136.5	322	56	0.2%
n = 16, DR = 48 bits (very large DR)	118.2	173	-	107.3	530	120	-

Moreover, the proposed reverse converter has been also compared with pure ROM-based reverse converter. Since the least n bits of the binary X is obtained by appending $\langle x_2 \rangle_{2^n}$ to the

$2n$-bit operand, the size of this ROM-based reverse converter can be reduced from $(2^{3n+1} \times 3n)$ bits to $(2^{3n+1} \times 2n)$ bits.

In a similar manner, this reverse converter has been implemented on Virtex-4 XC4VSX25 FPGA. This converter could only be implemented for DR = 12 bits and 15 bits, due to the limitations of BRAMs built-in this device. Therefore, it is obvious that the application fields of this type of reverse converters are very limited. The implementation results and comparison between the proposed reverse converter and pure R0M-based one are illustrated in Tab. 4.14.

Tab. 4.14: Comparison between proposed reverse converter and pure-ROM based one

Dynamic range	Pure ROM-based converter			Proposed converter			Time improving %
	Max. freq. [MHz]	4-LUTs	BRAMs [18 Kb]	Max. freq. [MHz]	4-LUTs	BRAMs [18 Kb]	
n = 4, DR = 12 bits (medium DR)	401.3	-	6	296.1	47	1	-
n = 5, DR = 15 bits (medium DR)	189.4	-	36	288.8	50	1	52.5%
n = 8, DR = 24 bits (large DR)	could not be implemented	-	-	279.9	78	1	-

For DR = 12 bits, the maximum frequency is 401.3 MHz. It is obvious that this converter is abundantly faster than the proposed one. However, DR = 15 bits, the maximum frequency of this converter considerably decreases (the proposed converter becomes faster by 52.5%).

Hence, the above discussion leads to the fact that the proposed reverse converter is more efficient than both pure ROM-based and pure combinatorial ones. The main drawback of pure ROM-based converters is their size, whereas pure combinatorial converters suffer from poor timing performance comparing to the proposed one. However, for very large dynamic ranges (larger than 48 bits), these converters are more efficient.

4.5 Proposed residue comparator

The proposed algorithm for reverse conversion has been also used to compare residue numbers in their RNS representation. According to equation (4.50), in order to compare two RNS numbers (x_1, x_2, x_3), (y_1, y_2, y_3), the following values should be compared, (G_x and G_y) and (x_2 and y_2). This comparison is done using binary comparators of 2n bits and n bits.

However, since G is the sum of k_2&k_2 and $\langle g \rangle_{2^n+1}$, the 2n-bit binary comparator is split into two parallel binary comparators of n bits and (n + 1) bits. The proposed structure of the residue comparator based on the proposed algorithm is shown in Fig. 4.16.

It is obvious that, the proposed residue comparator has the same structure as that of the proposed reverse converter except the 2n-bit adder that computes G.

For the sake of fair comparison, a part of the internal structure of the residue comparator (the binary comparators, multiplexers and a 3-to-1 *AND* gate) was designed in a same manner as the one in [47].

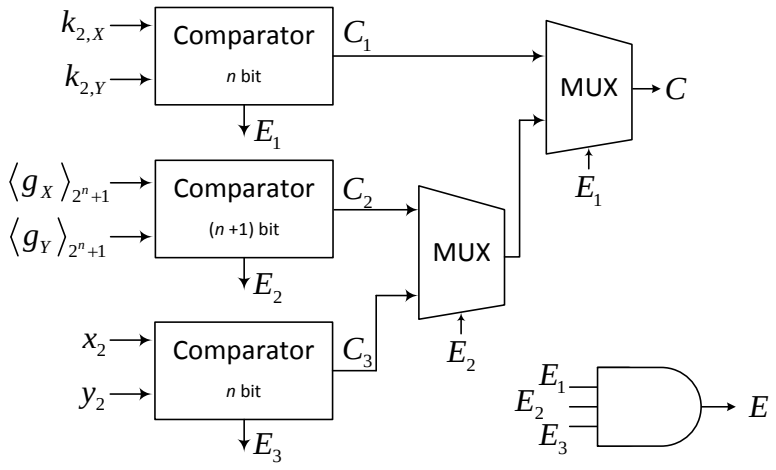


Fig. 4.16: The structure of the residue comparator based on the proposed algorithm [86]

The proposed residue comparator was implemented on Virtex-4 XC4VSX25 FPGA for different dynamic range requirements, 12 bits (medium dynamic range), 24 bits, 33 bits and 39 bits (large dynamic ranges) and 48 bits (very large dynamic range). Similarly to the reverse converter proposed in Section 4.4.2, the proposed comparator could not be implemented for dynamic ranges greater than 48 bits, due to the BRAM limitation built-in Virtex-4 XC4VSX25, where the maximum RAM size is 2304 Kb.

The structure of the proposed comparator is very similar to the one of the proposed reverse converter shown in Fig. 4.15. The only differences are the ROMs and the final 2n-bit adder. The ROMs, containing k_2 of the comparable numbers X and Y, were designed as dual

port ROMs and the final 2*n*-bit adder was omitted. Tab. 4.15 shows the implementation results of the proposed residue comparator and its counterparts [4], [47] for different dynamic range requirements.

According to Tab. 4.15, the residue comparator [47] can operate at higher frequencies than the one based on the reverse converter [4]. However, [4] utilizes less LUTs than [47]. Thus, comparator [47] can be considered as more time efficient, whereas, the one based on [4] can be considered as more area efficient.

Since the proposed comparator has a very similar structure to the one of the proposed converter and since this converter has been already compared with [4] in the previous section, thus, timing improving percentages of the proposed design have been computed in association with [47] only. The speed gain of the proposed comparator is about 9% for DR = 12 bits. Then, it extensively increases to 42.4% for DR = 33 bits. This gain afterwards begins to gradually decrease until it reaches a break point (DR = 39 bits), where the speed gain becomes 0.2%. From DR = 42 bits and larger, timing performance of the proposed reverse converter becomes worse than that of [47]. Hence, the proposed comparator is efficient for dynamic ranges up to 39 bits, whereas the proposed reverse converter based on the same algorithm is efficient for dynamic ranges up to 45 bits.

As stated in the previous section, the implementation results show that the proposed comparator is not suitable to be used in RNSs that provide very large dynamic ranges. Again, we can overlook this fact since the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ is not efficient to be used in applications that require very large dynamic ranges [75], [76].

Tab. 4.15: Comparison between different residue comparators for different dynamic range requirements

Dynamic range	New CRT-I based comparator [4]		[47]		Proposed comparator			Time improving % proposed vs. [47]
	Max. freq. [MHz]	4-LUTs	Max. freq. [MHz]	4-LUTs	Max. freq. [MHz]	4-LUTs	BRAMs [18 Kb]	
n = 4, DR = 12 bit	226.2	132	269.8	137	294.1	107	1	9%
n = 8, DR = 24 bit	160.4	222	184.9	232	259.5	178	1	40.4%
n = 11, DR = 33 bit	141.8	304	166.4	339	236.9	238	3	42.4%
n = 13, DR = 39 bit	126.2	308	159.8	337	160.1	366	12	0.2%
n = 16, DR = 48 bit	128.9	443	137.1	446	98.8	1114	120	-

4.6 Proposed designs for overflow and sign detection and correction in both signed and unsigned RNS systems

This section presents two universal efficient approaches for overflow and sign detection and correction after adding of two numbers in unsigned and signed RNS. Both methods are designed to be used in systems based on the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ that provides an even dynamic range. The importance of overflow detection in such systems has been already described in Section 2.3.4.

Moreover, by applying a tiny modification, these designs can be used in any system that has (2^n) as one of its moduli (i.e. has an even dynamic range). The proposed methods depend on a simple structure that provides fast and accurate detection and correction of the sign and overflow. The proposed designs are published in international conferences in Brno [81] and Seville, Spain [82].

4.6.1 Proposed component for overflow detection and correction in unsigned RNS

As aforementioned in Section 2.3.4, the general way to detect overflow is via comparing the sum with one of the addends, i.e. If $X \geq 0$ and $Y < M$ then $(X + Y)$ mod M causes overflow if and only if the sum is less than X.

The proposed method also depends on comparison; however, it compares each of the addends with half of the RNS dynamic range (M/2).

To detect overflow during the addition of two addends X and Y in unsigned RNS based on the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$, a single bit, that indicates to which half of the dynamic range M that addend belongs, is used. Based on this bit, the following three cases should be considered.

- The overflow will definitely occur if both of the addends are equal or greater than the half of dynamic range M/2.
- No overflow will definitely occur if both of the addends are less than M/2.
- If just one of the addends is equal to or greater than M/2, then the overflow prediction becomes complex and requires further processing and evaluation of the sum Z.

The magnitude evaluation of the addends (X and Y) is represented by a single bit (*evlt_bit*).

$$evlt\_bit_X = \begin{cases} 0 & ; \quad X < M/2 \\ 1 & ; \quad X \geq M/2 \end{cases} \tag{4.51}$$

The processing of *evlt_bit* of the two addends results in the three following cases,

$$overflow = \begin{cases} 0 & ; \quad evlt\_bit_X + evlt\_bit_Y = 0 \\ 1 & ; \quad evlt\_bit_X \bullet evlt\_bit_Y = 1 \\ 'u' & ; \quad evlt\_bit_X \oplus evlt\_bit_Y = 1 \end{cases} \tag{4.52}$$

where, '*u*' indicates the undetermined case of overflow occurrence and $(+,\bullet,\oplus)$ refer to the logical gates (*OR*, *AND*, *XOR*), respectively.

In order to solve the undetermined case '*u*', *evlt_bit* of the binary sum *Z* should be calculated by equation (4.51). Then the overflow can be indeed detected,

$$overflow = \begin{cases} 0 & ; \quad 'u' \quad \& \quad evlt\_bit_Z = 1 \\ 1 & ; \quad 'u' \quad \& \quad evlt\_bit_Z = 0 \end{cases} \tag{4.53}$$

Fig. 4.17 shows the structure of the proposed design that detects the overflow in unsigned RNS based on $\{2^n - 1, 2^n, 2^n + 1\}$. The magnitude evaluation of the addends and their sum, based on equation (4.51), is realized by a 2*n*-input *AND* gate and a 2-input *OR* gate. The magnitude evaluation unit is shown in Fig. 4.17 (a).

The overflow detection unit, based on equations (4.52) and (4.53), is realized by a 2:1 multiplexer and a *XOR* gate. This unit is shown in Fig. 4.17 (b)

The last component of the proposed design is the overflow correction unit, which is shown in Fig. 4.17 (c). This unit adds back *M* to the sum *Z* in order to correct the overflow and obtain the final accurate result. The adder that performs the final addition can be of any type, according to the design's goal and strategy.

Example: Consider $n = 3$, thus the moduli set {7, 8, 9}, $M = 504$. In case of unsigned RNS, the dynamic range of representable numbers is [0, 503]. The two numbers to be added are $X = 210$, $Y = 360$.

The outputs of magnitude evaluation units of the addends are $evlt\_bit_X = 0$, $evlt\_bit_Y = 1$.

$X = 210 = (0, 2, 3)$, $Y = 360 = (3, 0, 0)$.

The output of RAUs is (3, 2, 3). After residue to binary conversion $Z = 66$.

The output of magnitude evaluation unit of the sum *Z* is $evlt\_bit_Z = 0$.

The output of overflow detection unit is, *overflow* = 1.

Thus, overflow has been detected and further corrected as illustrated in Fig. 4.17.

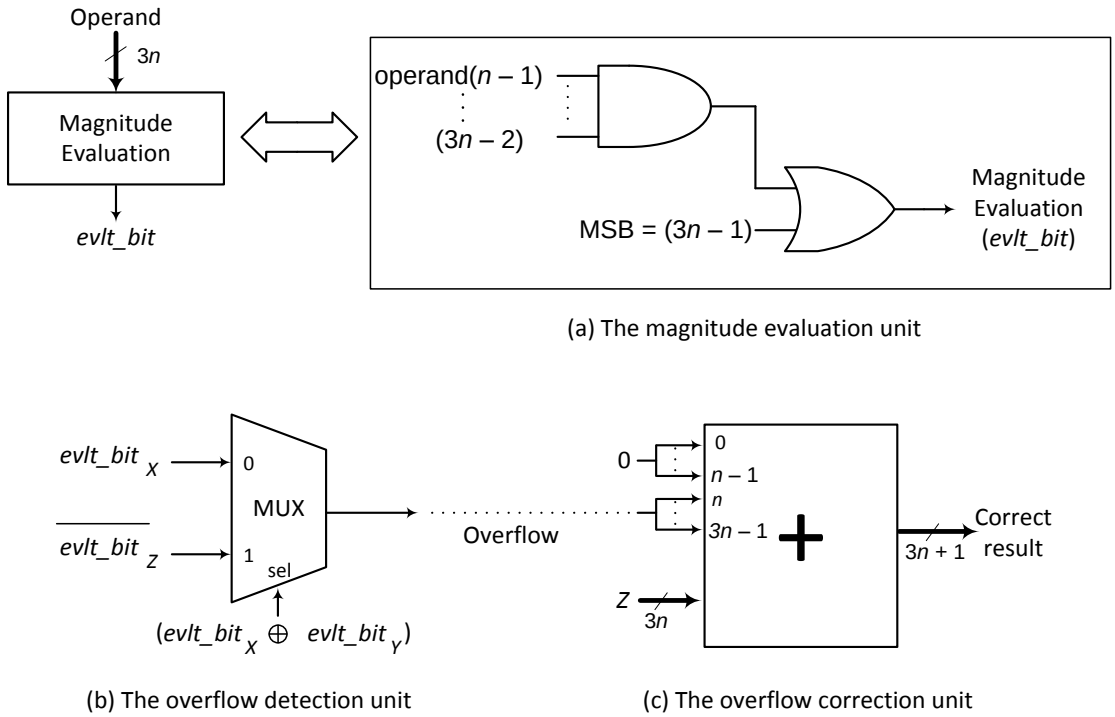


Fig. 4.17: The internal structure of the proposed overflow detection & correction component for unsigned numbers [82]
(a) Magnitude evaluation unit. (b) Overflow detection unit. (c) Overflow correction unit

4.6.2 Proposed component for overflow and sign detection and correction in signed RNS

In a similar manner, to detect overflow and sign in the addition of two addends X and Y in signed RNS based on the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$, a single bit (*evlt_bit*), that indicates to which half of the dynamic range M that addend belongs, is used.

As mentioned previously, in signed RNS, the positive numbers fall in the first half of the dynamic range, whereas, the negative ones fall in the second half. Thus, we have the following two cases that should be considered.

- No overflow will definitely occur if each of the addends has a different sign (fall in a different interval of M).
- Overflow may or may not occur if both addends have the same sign. Consequently, further processing should be done.

The sign evaluation of the addends is also represented by a single bit *evlt_bit* that is calculated by (4.51).

The processing of *evlt_bit* of the two addends results in the two following cases,

$$overflow = \begin{cases} 0 & ; \quad evlt\_bit_X \oplus evlt\_bit_Y = 1 \\ 'u' & ; \quad evlt\_bit_X \oplus evlt\_bit_Y = 0 \end{cases} \tag{4.54}$$

where, '*u*' indicates the undetermined case of overflow occurrence and $\oplus$ refers to the logical gate *XOR*.

In order to solve the undetermined case '*u*', *evlt_bit*, that determines the sign of the binary sum *Z* should be calculated by (4.51). Then the overflow can be indeed detected,

$$overflow = \begin{cases} 0 & ; \quad 'u' \quad \& \quad evlt\_bit_Z = evlt\_bit_X \\ 1 & ; \quad 'u' \quad \& \quad evlt\_bit_Z = \overline{evlt\_bit_X} \end{cases} \tag{4.55}$$

where, $\overline{evlt\_bit_X}$ refers to the logical negation of $evlt\_bit_X$.

Fig. 4.18 shows the structure of the proposed design that detects the sign and overflow in signed RNS based on $\{2^n - 1, 2^n, 2^n + 1\}$.

The sign evaluation of the addends and their sum, based on equation (4.51), is realized by an identical structure to that of the magnitude evaluation unit of the proposed design for unsigned RNS. It is shown in Fig. 4.18 (a).

The overflow detection unit, based on equations (4.54) and (4.55), is realized by a similar structure to that shown in Fig. 4.17 (b). It consists of a 2:1 multiplexer and two *XOR* gates. This unit is shown in Fig. 4.18 (b).

The overflow correction unit has an identical structure to that of the unsigned RNS. It is shown in Fig. 4.18 (c). Similarly, the adder that performs the final addition can be of any type.

Example: Consider $n = 3$, thus the moduli set {7, 8, 9}, $M = 504$. In case of signed RNS, the dynamic range of representable numbers is [-252, 251]. The two numbers to be added are $X = -150$, $Y = -240$.

Since the calculations within the RNS are based on unsigned arithmetic,

$X = 504 - 150 = 354$, $Y = 504 - 240 = 264$.

It is clear that both addends belong to the second half of the dynamic range as stated in equation (1.5).

The outputs of sign evaluation units of the addends are $evlt\_bit_X = 1$, $evlt\_bit_Y = 1$.

$X = 354 = (4, 2, 3)$, $Y = 264 = (5, 0, 3)$.

The output of RAUs is (2, 2, 6). After residue to binary conversion $Z = 114$.

The output of the sign evaluation unit of the sum Z is $evlt\_bit_Z = 0$.

$evlt\_bit_X \oplus evlt\_bit_Y = 0 \Rightarrow overflow = evlt\_bit_X \oplus evlt\_bit_Z = 1$. Thus, the output of overflow detection unit is, o*verflow* = 1.

Thus, overflow has been detected and further corrected as illustrated in Fig. 4.18.

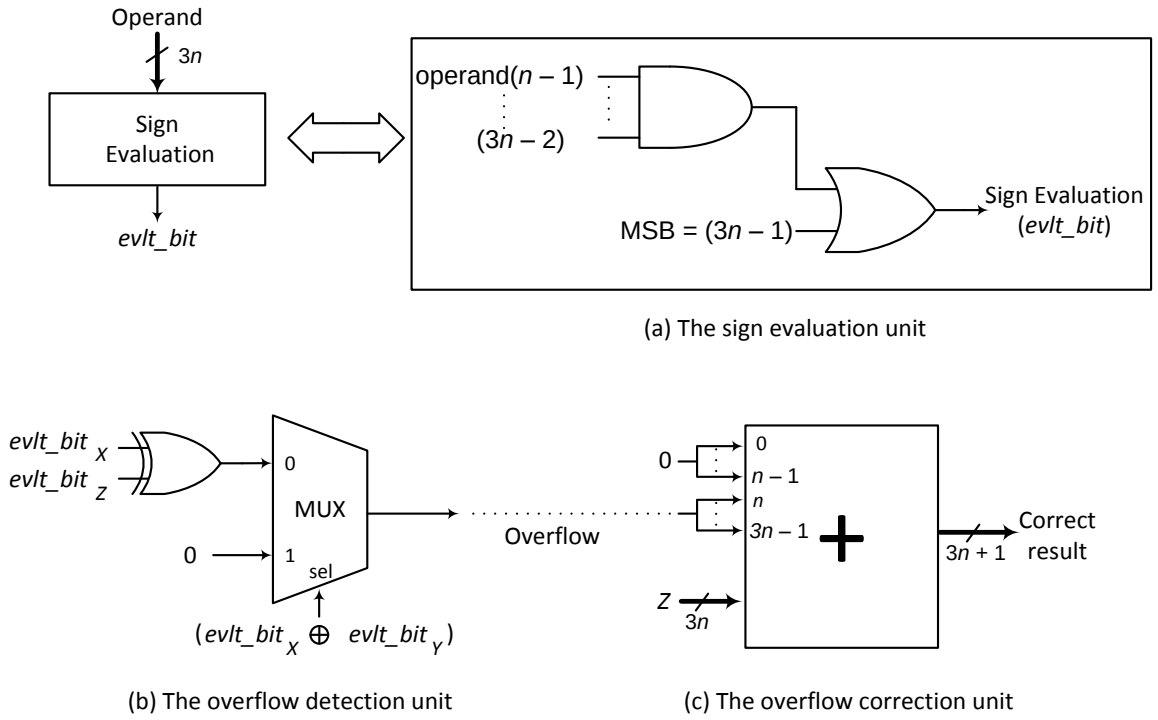


Fig. 4.18: The internal structure of the sign and overflow detection & correction component for signed numbers [82]
(a) Sign evaluation unit. (b) Overflow detection unit. (c) Overflow correction unit

4.6.3 Evaluating the proposed overflow and sign detection and correction designs

Since overflow and sign detection can be considered as a special case of comparison, the proposed designs were compared with other two. The first one represents the general approach for overflow detection, which consists of a binary comparator based on the residue-to-binary converter presented in [4]. Whereas, the second one is an efficient residue comparator for general moduli set introduced in [47].

Since the authors of the counterparts designs have presented the delay and area consumption of their designs using the unit gate model, and for the sake of fair comparison, I have estimated the delay and area consumption of my designs by using it. Tab. 4.16 illustrates the delay and complexity of the proposed designs and the analogous ones.

The structures of the proposed designs are very similar; the only difference is the additional *XOR* gate in the overflow detection unit for signed RNS. Both designs enclose the following components, the residue-to-binary converter proposed in [4], the operand evaluation units of the addends and their sum and the overflow detection unit. However, the evaluation of the addends is performed in parallel with the binary-to-residue conversion. Thus, no extra delay of these two units is presented. *evlt_bit* of the addends are stored in two cells of RAM, each of them has a size of 1 bit. The correction unit was not included in the comparison. Thus, the critical path is composed of the residue-to-binary converter, the operand evaluation unit of the sum and the overflow detection unit.

The first analogous design is a binary comparator based on the reverse converter proposed in [4]. This method uses two binary comparators with a 2:1 multiplexer. The sizes of the binary comparators are $2n$ bits and n bits. Thus, the critical path in this design is composed of the residue-to-binary converter, the $2n$-bit comparator and the 2:1 multiplexer.

The second analogous design presented in [47] uses a special component for generating two numbers (A_x and B_x) which are further used in the comparison. Moreover, this method uses three binary comparators and two 2:1 multiplexers. The sizes if these comparators are n bits, n bits and $(n + 1)$ bits. Thus, the critical path of this circuit is composed of A_x and B_x generator, the $(n + 1)$-bit binary comparator and the two 2:1 multiplexers.

Tab. 4.16: Performance comparison between the proposed designs and the analogous ones

Design	Delay	Area consumption
Residue comparator based on [4]	$20n + 10$	$48n + 3$
[47]	$18n + 14$	$40n + 8$
Proposed-unsigned	$16n + \log n + 13$	$37n + 18$
Proposed-signed	$16n + \log n + 15$	$37n + 20$

According to Tab. 4.16, both proposed designs have less delay and complexity without compromising on accuracy. Generally, this lower area requirements leads to lower power consumption.

In case of overflow occurrence, M is added back to the binary sum, in order to correct the sum (Z) and get the final accurate result. The adder that performs this addition can be of any type, based on the design's goal and strategy. Moreover, the size of this adder is $2n$ bits instead of $3n$ bit, since the first n bits of M for the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ are '0'.

$$\begin{aligned} overflow = '0' \Rightarrow \quad & final\ result = Z + "\underbrace{00\cdots00}_{2n}\underbrace{0\cdots0}_{n}" \\ overflow = '1' \Rightarrow \quad & final\ result = Z + "\underbrace{11\cdots11}_{2n}\underbrace{0\cdots0}_{n}" \end{aligned} \tag{4.56}$$

Both proposed designs can be used with any RNS that uses any moduli set, which has (2^n) as one of its moduli, i.e. has an even dynamic range. This can be simply performed by applying a tiny modification on the evaluation units, represented in changing the input number of the *AND* gate, according to the dynamic range provided by the used moduli set. Moreover, in case of changing the number and the order of the *AND* gate's inputs, these two designs can be used with any other moduli set, even if it provides an odd dynamic range. However, for such systems, the parity checking technique will be faster and simpler than the proposed one.

4.7 Proposed RNS-based application

An RNS-based image processing application is presented in this section. This application applies a number of filters in spatial domain, such as sharpen, edge detection and enhancing on a gray-scale image. All the processing is done using the RNS instead of the binary number system (BNS). The proposed application proves that using the RNS results in faster and power-reduced image filtering applications. Moreover, the negative effects of using improper moduli sets in an RNS based image-processing applications are highlighted. This work has been published in Gdynia, Poland [84]. An extended version of this work is under review to be published in ElectroScope journal [85].

4.7.1 The proposed RNS-based image processing application

For performing filtering in spatial domain, a mask should be moved on the image according to the following equation [72],

$$y(i,j) = \sum_{k=-a}^{a} \sum_{l=-b}^{b} h(k,l)x(i+k, j+l) \tag{4.57}$$

where, *x* and *y* are the input and output images, respectively. *h* is the mask that is going to be applied on the image. *a* and *b* are positive integers.

Theoretically, any spatial filter can be used based on the RNS, since the concept of its application is the same.

During my research, I have implemented the following filters,

- Sharpening filter: $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$
- Edge enhancement filters: $\begin{bmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$, $\begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ and $\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$, for horizontal, vertical and diagonal edge enhancement, respectively.
- Edge detection filters: $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ and $\begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$

During this research, using ROM-based converters from binary to RNS and vice versa turned out to be the most efficient method for this application and its dynamic range. The proposed design is divided into three stages; the first one includes a ROM-based forward converter that converts the input pixels form binary to RNS. In the second stage, three parallel residue arithmetic units, corresponding to the three moduli, perform filtering operations (multiplying by the filter's coefficients and adding). The third stage converts the output residues of the three RAUs into their binary equivalent using ROM-based reverse converter.

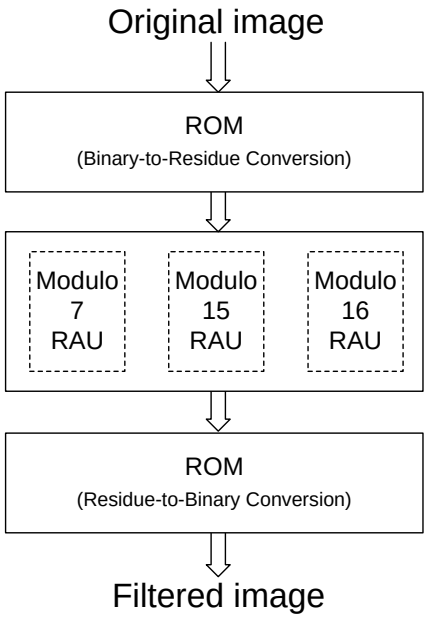


Fig. 4.19: The structure of the proposed RNS-based image-processing application [84]

According to [75], the most efficient moduli set for applications that require medium dynamic ranges (less than 22 bits) is $\{2^{n-1} - 1, 2^n - 1, 2^n\}$. We chose $n = 4$, so the used moduli set during our work is {7, 15, 16}. Its dynamic range = 1680 which is sufficient for image filtering application and eliminates the necessity to a special component for overflow detection and correction.

As stated in Section 2.3, since the moduli set is of the form $\{2^{n-1} - 1, 2^n - 1, 2^n\}$, their residue arithmetic units are high-speed and efficient. Fig. 4.19 illustrates the structure of the proposed design.

4.7.2 The moduli set effect on the output of image processing application

In this section, the importance of using a proper moduli set that provides a sufficient dynamic range for a digital image-processing application is presented.

Two examples are illustrated in order to show the effects of using a moduli set with insufficient dynamic range.

As aforementioned in Section 4.7.1, the utilized moduli set is {7, 15, 16}. Its dynamic range = 1680 which is sufficient for image filtering application and eliminates the necessity to a special component for overflow detection.

Many papers suggested using moduli sets with smaller dynamic ranges, such as {5, 7, 8} and {7, 8, 9} that provide $M = 280$ and $M = 504$, respectively [60], [59]. Since the possible pixel values in a digital image processing application have the range [0, 255], these papers considered that using these sets would be sufficient. However, the following two examples clarify the fact that this is not always true, except the case when using a special component for overflow detection, which was not mentioned in any of those papers.

Example 1: the filtered pixel has a negative value

Suppose the following pixel values in a part of a gray-scale image,

$$\begin{bmatrix} 23 & 20 & 35 \\ 23 & 16 & 34 \\ 24 & 16 & 32 \end{bmatrix}$$

Suppose the following sharpening filter that is going to be applied on that image,

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

1. Using the weighted number system

   According to equation (4.57), the filtered pixel value is,

$$\begin{bmatrix} 23 & 20 & 35 \\ 23 & 16 & 34 \\ 24 & 16 & 32 \end{bmatrix} \times \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} = -13$$

   In standard image processing applications, negative numbers (in our case –13) are considered to be 0 (which refers to black color).

2. Using the RNS based on the moduli set {5, 7, 8}

   After forward conversion according to the moduli set {5, 7, 8},

$$\begin{bmatrix} (3,2,7) & (0,6,4) & (0,0,3) \\ (3,2,7) & (1,2,0) & (4,6,2) \\ (4,3,0) & (1,2,0) & (2,4,0) \end{bmatrix} \times \begin{bmatrix} (0,0,0) & (4,6,7) & (0,0,0) \\ (4,6,7) & (0,5,5) & (4,6,7) \\ (0,0,0) & (4,6,7) & (0,0,0) \end{bmatrix} = (2,1,3) = 267$$

   Positive integers greater than 255 (in our case 267) are considered 255 (which refers to white color) in standard image processing applications.

   It is obvious that the difference between the two results is huge; by using the weighted system, the result is a black pixel, but by using RNS based on {5, 7, 8} the result is a white pixel.

3. Using the RNS based on the moduli set {7, 15 ,16}

   According to equation (1.5), I have divided the dynamic range into 2 parts; [0, 839] for positive integers, and [840, 1679] for negative ones.

$$\begin{bmatrix} (2,8,7) & (6,5,4) & (0,5,3) \\ (2,8,7) & (2,1,0) & (6,4,2) \\ (3,9,8) & (2,1,0) & (4,2,0) \end{bmatrix} \times \begin{bmatrix} (0,0,0) & (6,14,15) & (0,0,0) \\ (6,14,15) & (5,5,5) & (6,14,15) \\ (0,0,0) & (6,14,15) & (0,0,0) \end{bmatrix} = (1,2,3) = 1667$$

   Since 1667 locates in the second half of the dynamic range, this means that it has a negative value, so we consider it 0, which is the same as the one computed using the weighted number system.

   A question can arise here, what if the dynamic range in the case of using {5,7,8} was divided into two parts for representing negative and positive values. Since the dynamic range of {5, 7, 8} = 280, therefore, it is not enough to be divided into two parts [0, 139] for positive values and [140, 279] for negative ones.

Example 2: the filtered pixel has a large positive value

Suppose the following pixel values in a part of a gray-scale image,

$$\begin{bmatrix} 193 & 185 & 118 \\ 203 & 214 & 200 \\ 201 & 189 & 217 \end{bmatrix}$$

1. Using the weighted number system

   Suppose using the same sharpening filter,

   $$\begin{bmatrix} 193 & 185 & 118 \\ 203 & 214 & 200 \\ 201 & 189 & 217 \end{bmatrix} \times \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} = 293$$

   293 is considered 255 (white color) in standard image processing applications.

2. Using the RNS based on the moduli set {5,7,8}

   $$\begin{bmatrix} (3,4,1) & (0,3,1) & (3,6,6) \\ (3,0,3) & (4,4,6) & (0,4,0) \\ (1,5,1) & (4,0,5) & (2,0,1) \end{bmatrix} \times \begin{bmatrix} (0,0,0) & (4,6,7) & (0,0,0) \\ (4,6,7) & (0,5,5) & (4,6,7) \\ (0,0,0) & (4,6,7) & (0,0,0) \end{bmatrix} = (3,6,5) = 13$$

   13 represents a dark color near to black. Again, the huge difference between the two results is clear.

3. Using the RNS based on the moduli set {7,15,16}

   $$\begin{bmatrix} (4,13,1) & (3,5,9) & (6,13,6) \\ (0,8,11) & (4,4,6) & (4,5,8) \\ (5,6,9) & (0,9,13) & (0,7,9) \end{bmatrix} \times \begin{bmatrix} (0,0,0) & (6,14,15) & (0,0,0) \\ (6,14,15) & (5,5,5) & (6,14,15) \\ (0,0,0) & (6,14,15) & (0,0,0) \end{bmatrix} = (6,8,5) = 293$$

   293 belongs to the first part of the dynamic range, which represents positive integers. The result based on our moduli set is the same as the one computed using the weighted system.

From the above two examples, the importance of choosing a proper moduli set with a sufficient dynamic range is clear. Even though the pixel values range in [0, 255], the overflow case should be taken into account, or at least a component that detects overflow should be utilized, which is not an easy task when using moduli sets with even dynamic ranges and presents further delay and complexity to the design [1], [75].

4.7.3 Performance evaluation and comparison

The proposed design was compiled and implemented on XC4VLX15 Virtex-4 FPGA device. A 256×256 gray-scale image was stored in a RAM, which was designed using Xilinx core generator v.13.4. Both forward and reverse converters were implemented as ROMs. Since the concept of spatial filters is the same, I have only illustrated the implementation results of applying sharpening and edge detection filters.

Tab. 4.17 presents the maximum frequency and power consumption at clock frequency of 100 MHz after implementing the filters using the BNS and the RNS based on the moduli set {7, 15, 16}. The proposed design has shown considerably more impressive characteristics than its counterpart. It can operate at higher frequency (by approx. 39.1%) and has less power consumption (by approx. 23.7%) when operating at frequency of 100 MHz.

Tab. 4.17: Comparison between binary and RNS-based image processing application that applies spatial filters on a gray-scale image

	Binary-based application	**RNS-based application**	**Improvements %**
Max. frequency [MHz]	127.08	176.75	39.1%
Power consumption at 100 MHz [mW]	489	373	23.7%

Fig. 4.20 shows the results of applying edge detection and sharpening filters using the RNS based on {7, 15, 16} and {5, 7, 8}. The original input gray-scale image is shown in Fig. 4.20 (a). The output-filtered images using the RNS based on the moduli sets {7, 15, 16} and {5, 7, 8} are show in Fig. 4.20 (b) – (e).

Again, the negative effect of using a moduli set with insufficient dynamic range is clear. Fig. 4.20 (c) and (e) show the distorted output filtered images after applying edge detection and sharpening filters using the RNS based on the set {5, 7, 8}. Whereas, these images based on the proposed design, which uses the moduli set {7, 15, 16}, are identical to those based on the BNS.

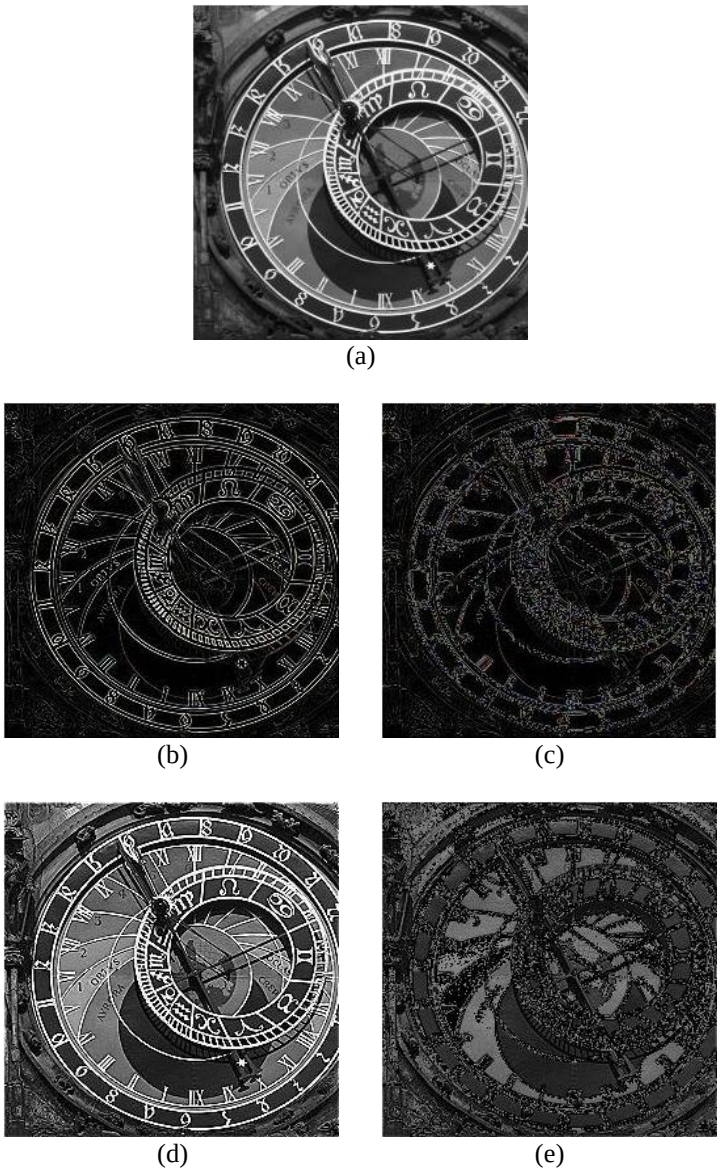


Fig. 4.20: The Output images after applying edge detection and sharpening filters,
(a) The original gray-scale image.
(b) After applying edge detection filter using the RNS based on the moduli set {7,15,16} [84].
(c) After applying edge detection filter using the RNS based on the moduli set {5,7,8} [60].
(d) After applying sharpening filter using the RNS based on the moduli set {7,15,16} [84].
(e) After applying sharpening filter using the RNS based on the moduli set {5,7,8} [60].

4.8 When to use the RNS (Binary vs. RNS)

This section can be considered as the conclusive outcome of this thesis. It illustrates the main issues that should be taken into account when deciding to use the RNS instead of BNS.

In Section 4.1, a detailed comparison between different moduli sets is presented. However, during my research, I have observed that a number of these sets result in applications with worse timing performance than that of its equivalent binary-based one. In other words, there is no point of using the RNS based on these sets, at least in cases when the

main goal of the design is "timing performance". Therefore, the first part of this section establishes the main aspect that should be considered when choosing a moduli set in order to achieve better timing performance than that of the BNS.

On the other hand, the second part studies the cases when utilizing the RNS would be the most advantageous. As aforementioned before, using the RNS is of great benefit in applications where addition, subtraction and multiplication are dominant. However, dominant is an abstracted word. Thus, a detailed discussion about this issue is presented below. It compares the performance of binary and RNS-based applications and highlights the areas when using the RNS can be extremely useful in order to achieve higher timing performance and less power consumption.

4.8.1 The effect of the critical modulo within a moduli set

The key concept of the RNS is that the delay of the RAU corresponding to the critical channel is less than the delay of its equivalent binary arithmetic unit (BAU). However, the presence of RNS converters affects the whole RNS, since their delay is rather long. Typically, an RNS that performs one arithmetic operation suffers from longer delay than that of its binary equivalent. Therefore, the essential point of using the RNS is ascertained via performing many arithmetic operations, so the delay of BAUs exceeds the delay of residue arithmetic units plus the RNS converters. Nevertheless, the issue of performing many arithmetic operations is discussed in details in the next section.

As aforementioned before, many moduli sets of different forms and moduli number have been suggested. However, the unpredictable issue is that, using some of these moduli sets results in worse timing performance than that of binary-based ones. In systems based on such sets, the delay of RAU corresponding to the critical channel exceeds the delay of its equivalent BAU, regardless the delay of RNS convertors. An example that clarifies this issue is presented below.

Example

Considering a system that uses the moduli set $\{2^n - 1, 2^n + 1, 2^{2n} + 1\}$ [9]. The width of its dynamic range is $4n$ bits. The critical modulo within this system is $(2^{2n} + 1)$. Its width is $(2n + 1)$ bits

According to Tab. 6.1, the delay of a modulo $(2^{2n} + 1)$ adder $T_{\mathrm{mod}\,(2^{2n}+1)\,add}$ and a modulo $(2^{2n} + 1)$ multiplier $T_{\mathrm{mod}\,(2^{2n}+1)\,mul}$ are as follows,

$$T_{\mathrm{mod}\,(2^{2n}+1)\,add} = 16n + \log_2 n + 5 \ , \qquad T_{\mathrm{mod}\,(2^{2n}+1)\,mul} = 32n + 9$$

On the other hand, since the DR width is 4*n* bits, the delay of the binary adder and multiplier on operands of 4*n* bits are as follows,

$$T_{bin\,add\,(4n\,bit)} = 16n \;, \qquad T_{bin\,mul\,(4n\,bit)} = 32n - 7$$

In order to get the benefit of using the RNS, the following two conditions should be met,

$$\begin{aligned} T_{critical\ \ modulo\ \ add} &< T_{bin\ \ add} \\ T_{critical\ \ modulo\ mul} &< T_{bin\ \ mul} \end{aligned} \tag{4.58}$$

According to the above conditions and by substituting the delay values of both modular and binary adders,

$$T_{\mathrm{mod}\,(2^{2n}+1)\,add} < T_{bin\,add} \;\Rightarrow\; 16n + \log_2 n + 5 \not< 16n$$

It is obvious, that this inequality is not true. The delay of the binary adder is longer than that of the critical modular one.

In a similar manner, by substituting the delay values of both modular and binary multipliers,

$$T_{\mathrm{mod}\,(2^{2n}+1)\,mul} < T_{bin\,mul} \;\Rightarrow\; 32n + 12 \not< 32n + 9$$

Again, the above inequality is also not true. The delay of the binary multiplier is longer than that of the critical modular one.

Therefore, we can see that using the moduli set $\{2^n - 1, 2^n + 1, 2^{2n} + 1\}$ [9] results in RAU with longer delay than that of the BAU, regardless the delay presented by the RNS converters. i.e. there is no benefit of using the RNS based on this set in order to obtain higher timing performance.

End of example

According to the above discussion, setting a concrete condition, that should be met when choosing a moduli set, is obligatory.

Assuming an RNS-based application that uses one of the previously stated moduli sets, the width of its DR is *m* bits. According to the studied moduli sets stated in Tab. 2.1, the smallest critical channel is of the form $(2^n - 1)$.

According to the unit gate model and Tab. 6.1, the delay of the modular adder corresponding to the critical channel $(2^n - 1)$ is as follows,

$$T_{\mathrm{mod}(2^n-1)\,add} = 8n$$

Since the width of the DR is *m* bits, the delay of binary adder is as follows,

$$T_{bin\ add\ (m\ bit)} = 4m$$

Again, in order to get the benefit of using the RNS, inequality (4.58) should be met. Hence,

$$T_{critical\ \ modulo\ \ add} < T_{bin\ \ add} \ \Rightarrow \ \ 8n < 4m \ \Rightarrow \ \ 2n < m \tag{4.59}$$

Consequently, inequality (4.59) clarifies the fact, that in order to obtain better timing performance of addition using the RNS, the dynamic range width should be greater than the critical channel width by more than two times.

In a similar manner, the condition that should be met in order to obtain higher timing performance of multipliers than that of the BNS, is as follows,

$$T_{critical\ \ modulo\ \ mul} < T_{bin\ \ mul} \ \Rightarrow \ \ 16n - 7 < 8m - 7 \ \Rightarrow \ \ 2n < m \tag{4.60}$$

Inequality (4.60) also illustrates the same fact that in order to obtain better timing performance of multiplication using the RNS, the dynamic range width should be greater than the critical channel width by more than two times.

However, the above-concluded results are only based on a theoretical point of view. Therefore, in order to verify these results, a comparison based on FPGA implementation has been held. Timing performance of three parallel RAUs based on the moduli set $\{2^n - 1, 2^n + 1, 2^{2n} + 1\}$ [9] has been compared with their equivalent BAU. The RNS system has been implemented without converters, only RAUs with respect to the three moduli within the set. The implementation results are illustrated in Tab. 4.18.

Tab. 4.18: A comparison between binary arithmetic unit and parallel RAUs (in terms of timing performance) based on moduli set $\{2^n - 1, 2^n + 1, 2^{2n} + 1\}$ [9], with DR = 4*n* bit

DR	Addition		Multiplication	
	Binary [MHz]	RNS [MHz]	Binary [MHz]	RNS [MHz]
n = 4, DR = 16 bit	465.1	253.7	317.5	131
n = 8, DR = 32 bit	330.4	225.5	149.6	115.9
n = 15, DR = 60 bit	174.6	171.6	79.3	65.3

According to Tab. 4.18, it is clear that the binary arithmetic unit can operate at higher frequencies than that of RAU. Again, the reverse converters were not included in the comparison. Thus, it is clear that using the RNS based on this moduli set does not improve timing performance.

Hence, the main condition before choosing a moduli set in order to obtain better timing performance of addition or multiplication using the RNS, is as follows, "the critical modulo width (bits number) in the moduli set should be less than half of the dynamic range width this set provides". Therefore, the moduli sets that should not be used in systems than concern about timing performance are those that do not meet this issue. During my research, I have observed a number of already published papers suggesting moduli sets that can be disadvantageous to use, such as $\{2^n - 1, 2^n, 2^{2n+1} - 1\}$ [8], $\{2^n - 1, 2^n + 1, 2^{2n} + 1\}$ [9] and $\{2^{n/2} - 1, 2^{n/2} + 1, 2^n + 1, 2^{2n+1} - 1\}$ [18]. These sets are illustrated in Tab. 4.19. The inefficiency proofs, based on theoretical and implementation results, of using moduli set $\{2^n - 1, 2^n + 1, 2^{2n} + 1\}$ [9], are stated. The inefficiency of using the moduli sets [8] and [18] can be similarly proved.

Tab. 4.19: Moduli sets that result in applications with worse timing performance than binary-based ones

Moduli #	Modulo set	Critical modulo channel	Critical channel width	Dynamic range width
3	$\{2^n - 1, 2^n, 2^{2n+1} - 1\}$ [8]	$(2^{2n+1} - 1)$	$(2n + 1)$ bits	$(4n + 1)$ bits
	$\{2^n - 1, 2^n + 1, 2^{2n} + 1\}$ [9]	$(2^{2n} + 1)$	$(2n + 1)$ bits	$(4n)$ bits
4	$\{2^{n/2} - 1, 2^{n/2} + 1, 2^n + 1, 2^{2n+1} - 1\}$ [18]	$(2^{2n+1} - 1)$	$(2n + 1)$ bits	$(4n + 1)$ bits

4.8.2 When is RNS superior than binary number system

As previously stated, the main advantageous field of using the RNS instead of BNS is in applications that contain a dominant number of additions, subtractions and multiplications. Hence, this section discusses the issue of how many additions/multiplications should an application contain in order to obtain an RNS-based application faster than a binary-based one.

Theoretically, in case of performing only one addition or multiplication, the binary-based application will be faster than the RNS-based one, due to presence of both converters.

Therefore, in order to get the benefit of the RNS's properties, these arithmetic operations should be performed at least a certain number of times in order to make the profits gained in the RAUs exceeds the overhead of the converters.

Assuming an application that performs a certain arithmetic operation, this application based on the RNS will be faster, if the delay of the binary arithmetic unit exceeds the delay of the residue arithmetic one corresponding to the critical modulo channel plus the delay of converters.

$$T_{bin\ \ operation} > T_{critical\ \ modulo\ \ operation} + T_{conv} \tag{4.61}$$

where, $T_{bin\ \ operation}$ refers to the delay of binary arithmetic unit that performs that certain operation, $T_{critical\ \ modulo\ \ operation}$ refers to the delay of the RAU corresponding to the critical modulo and T_{conv} refers to the delay of both forward and reverse converters.

Since the delay of the converters has always a rather big value, the above expression can only be achieved by repeating this operation a certain number of times *k*,

$$k \times \left(T_{bin\ \ operation} \right) > k \times \left(T_{critical\ \ modulo\ \ operation} \right) + T_{conv} \tag{4.62}$$

Example

Assuming an RNS-based application that uses the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ with DR = 3*n* bits. The critical channel in this system is modulo $(2^n + 1)$.

According to the unit gate model and Tab. 6.1 stated in the appendix, timing requirements of the forward converter, reverse converter, modulo $(2^n + 1)$ adder and modulo $(2^n + 1)$ multiplier within this system are as follows,

$$T_{FC} = 12n + 9, \quad T_{RC} = 20n + 12 \Rightarrow T_{conv} = 32n + 21$$
$$T_{\mathrm{mod}\,(2^n+1)\,add} = 8n + \log_2 n + 4, \quad T_{\mathrm{mod}\,(2^n+1)\,mul} = 16n + 9$$

Since the width of the DR is 3*n* bits, timing requirements for performing addition and multiplication in a binary system are as follows,

$$T_{(3n\,bit)\,add} = 12n, \quad T_{(3n\,bit)\,mul} = 24n - 7$$

According to equation (4.62),

$$k \times \left(T_{bin\ \ add} \right) > k \times \left(T_{critical\ \ modulo\ \ add} \right) + T_{conv} \Rightarrow k \times \left(\underbrace{12n}_{bin\ add} \right) > k \times \left(\underbrace{\underbrace{8n + log_2\, n + 4}_{critical\ modulo\ add}}_{} \right) + \underbrace{32n + 21}_{conv}$$

Thus,

$$k > \frac{32n + 21}{4n - \log_2 n - 4} \tag{4.63}$$

This means that in order to achieve better timing performance of the RNS-based application, the addition should be repeated at least k times determined in equation (4.63).

In a similar manner, regarding multiplication based on the same moduli set, this operation should be repeated at least l times in order to achieve better timing performance of the RNS-based application.

$$l > \frac{32n + 21}{8n - 16} \tag{4.64}$$

End of example

According to equations (4.63) and (4.64), the minimum numbers of iterated additions and multiplications have been computed for different dynamic range requirements. Their values are shown in Tab. 4.20.

Tab. 4.20: The least numbers of iterated additions and multiplications required to achieve better timing performance of the RNS-based application that uses moduli set $\{2^n - 1, 2^n, 2^n + 1\}$

n	Dynamic range	Minimum number of iterated additions	Minimum number of iterated multiplications
4	12 bits	$k \geq 15$	$l \geq 10$
8	24 bits	$k \geq 12$	$l \geq 6$
11	33 bits	$k \geq 11$	$l \geq 6$
16	48 bits	$k \geq 10$	$l \geq 5$
22	66 bits	$k \geq 10$	$l \geq 5$

From Tab. 4.20, it is clear that the RNS speed advantage is mainly manifested in multiplication rather than addition. It is also obvious that as the dynamic range increases, the minimum number of repeated operations decreases. For medium dynamic range (12 bits), an RNS-based application that performs 10 repeated multiplications is theoretically faster than the binary-based one. Whereas, for very large dynamic range (66 bits), this number decreases

to only five repeated multiplications to achieve better timing performance than the binary based one.

However, this study is based on the theoretical point of view. Hence, in order to compare the theoretical results with the practical ones, two applications that perform different numbers of iterated additions and iterated multiplications, respectively, have been implemented on Virtex-4 XC4VSX25 FPGA. These applications are based on the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$.

The implementing results illustrating the maximum frequency and power consumption for different number of iterated additions/multiplications, for different dynamic ranges are detailed in Appendix in Tab. 6.6 - Tab. 6.13. The utilized components within these applications are based on the proposed designs stated in this thesis, including forward converter, reverse converter, modular adders and multipliers.

Since the proposed reverse converter could not be implemented for DRs larger than 48 bit on Virtex-4 XC4VSX25 FPGA, thus, the maximum DR that the proposed designs were implemented for was 48 bit instead of 66 bit.

In a similar manner as shown in Tab. 4.20, the least numbers of iterated additions/multiplications for different dynamic ranges, based on Virtex-4 implementation, are shown in Tab. 4.21.

Tab. 4.21: The least numbers of iterated additions and multiplications required to achieve better timing performance of the RNS based on Virtex-4 implementation

n	**Dynamic range**	**Minimum number of iterated additions**	**Minimum number of iterated multiplications**
4	12 bits	$k \geq 3$	$l \geq 2$
8	24 bits	$k \geq 6$	$l \geq 1$
11	33 bits	$k \geq 6$	$l \geq 1$
16	48 bits	$k \geq 9$	$l \geq 1$

The difference between the theoretical and implementation results is obvious. The implementation results are much better than the theoretical ones. Less numbers of iterated additions/multiplications are needed in order to achieve better timing performance of the RNS-based application. There are two reasons behind this improvement; the first one is the efficient reverse converter being used. Its structure is partially based on integrated block

RAMs, which can run at 500 MHz in the selected device [73]. This presents a considerable improvement in timing performance of the overall system. The second reason is the usage of the dedicated XtremeDSP slices built-in Virtex-4 FPGA. These DSP slices can run at 500 MHz and have small power consumption 2.3 mW/100 MHz per slice [74].

However, as illustrated in the appendix in Tab. 6.10 - Tab. 6.13, the application based on iterated multiplications shows power-saving feature, for averagely 10 iterated multiplications. Therefore, for the sake of a clear discussion, the implementation results (maximum frequency, hardware and power consumption) of both applications based on 10 iterated additions/multiplications are shown in this section in Tab. 4.22, Tab. 4.23 and Tab. 4.24, respectively.

Tab. 4.22: The maximum frequency of applications performing 10 iterated additions and 10 iterated multiplications using the RNS and BNS

Dynamic range	Application based on 10 additions			Application based on 10 multiplications		
	RNS	Binary	Speed improvement %	RNS	Binary	Speed improvement %
12 bits	193.9 MHz	123.9 MHz	56.5 %	134.6 MHz	22.7 MHz	493 %
24 bits	158.1 MHz	111.6 MHz	41.7 %	124.5 MHz	19.9 MHz	525.6 %
33 bits	150.5 MHz	105.9 MHz	42.1 %	121.6 MHz	12.8 MHz	850 %
48 bits	103.1 MHz	100.1 MHz	3 %	106.7 MHz	12.6 MHz	749.8 %

As shown in Tab. 4.22, it is obvious that the application based on iterated multiplications has much more impressive results than the one based on additions. The reason is the DSP48 units in FPGA. These DSP units are used in the case of application based on pure multiplications. Whereas, the one based on pure additions is implemented using LUTs only.

Moreover, the application that performs only multiplications for dynamic ranges 33 bit and 48 bits, RNS-based application becomes more power efficient for multiplications iterated for more than 6 times and 5 times, respectively. As shown in Appendix in Tab. 6.12 and Tab. 6.13, these savings gradually increase as number of iterated multiplications increases.

According to the application being under discussion (based on 10 iterated multiplications), compared to the binary-based application, the power consumption of the RNS-based one is reduced by 25.2% and 39.8% for dynamic ranges 33 bits and 48 bits, respectively, as shown in Tab. 4.23.

Tab. 4.23: Power consumption at 100 MHz running application performing 10 iterated multiplications using the RNS and BNS

Dynamic range	RNS	Binary	Power saving %
33 bits	506 mW	676 mW	25.2 %
48 bits	716 mW	1189 mW	39.8 %

This power efficiency feature is evident in RNS-based application for large and very large dynamic ranges (33 bits and 48 bits). The reason is the huge increment in the utilized 4 input look-up tables in binary-based application compared to the RNS-based one. Hardware requirements for implementing binary and RNS-based applications that perform 10 iterated multiplications are illustrated in Tab. 4.24.

Tab. 4.24: Hardware requirements for implementing applications performing 10 iterated multiplications using the RNS and BNS on Virtex-4 XC4VSX25 FPGA

Dynamic range	RNS			Binary		
	4-LUTs	DSP48s	RAMB16s	4-LUTs	DSP48s	RAMB16s
33 bits	610	23	3	3 301	15	0
48 bits	1 246	23	120	11 206	12	0

After implementing applications based on the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$, this study has been further extended to other moduli sets. The minimum numbers of iterated additions and multiplications required to achieve better timing performance of the RNS-based applications that use different moduli sets are shown in Tab. 4.25.

During this study, an unexpected issue has been observed. The three-moduli set $\{2^n - 1, 2^n, 2^{n+1} - 1\}$ [7] has the best results for different dynamic ranges. It requires the minimum number of iterated additions and multiplications in order to achieve better timing performance than binary applications. Another surprising issue is, that the five-moduli set $\{2^n, 2^{n/2} - 1, 2^{n/2} + 1, 2^n + 1, 2^{2n-1} - 1\}$ [17], contrary to all other sets, requires more additions and multiplications as the dynamic range increases.

It should be mentioned that both moduli sets $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$ [11]-I and $\{2^n, 2^{n/2} - 1, 2^{n/2} + 1, 2^n + 1, 2^{2n-1} - 1\}$ [17] can only be used with even values of n. Therefore, n was chosen in such a way to keep the acquired dynamic ranges as close as possible.

Thus, it is evident that using the RNS is more advantageous in applications that have large and very large dynamic ranges and contain multiplications rather than only additions.

Tab. 4.25: The least numbers of iterated additions and multiplications required to achieve better timing performance of the RNS-based application that uses moduli set $\{2^n - 1, 2^n, 2^n + 1\}$

Moduli set	Dynamic range	*n*	Min. add. #	Min. mul. #
$\{2^n - 1, 2^n, 2^n + 1\}$ [4]	12 bits	4	$k \geq 15$	$l \geq 10$
	24 bits	8	$k \geq 12$	$l \geq 6$
	33 bits	11	$k \geq 11$	$l \geq 6$
	66 bits	22	$k \geq 10$	$l \geq 5$
$\{2^n - 1, 2^n, 2^{n+1} - 1\}$ [7]	13 bits	4	$k \geq 9$	$l \geq 5$
	25 bits	8	$k \geq 7$	$l \geq 4$
	34 bits	11	$k \geq 6$	$l \geq 3$
	64 bits	22	$k \geq 5$	$l \geq 3$
$\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$ [11]-I	9 bits	2	$k \geq 11$	$l \geq 7$
	25 bits	6	$k \geq 9$	$l \geq 5$
	34 bits	8	$k \geq 9$	$l \geq 5$
	64 bits	16	$k \geq 8$	$l \geq 4$
$\{2^n - 1, 2^{2n}, 2^n + 1, 2^{2n} + 1\}$ [13]-II	12 bits	2	$k \geq 14$	$l \geq 9$
	24 bits	4	$k \geq 10$	$l \geq 6$
	36 bits	6	$k \geq 9$	$l \geq 5$
	66 bits	11	$k \geq 9$	$l \geq 4$
$\{2^n, 2^{n/2} - 1, 2^{n/2} + 1, 2^n + 1, 2^{2n-1} - 1\}$ [17]	9 bits	2	$k \geq 12$	$l \geq 6$
	19 bits	4	$k \geq 14$	$l \geq 7$
	39 bits	8	$k \geq 16$	$l \geq 8$
	69 bits	14	$k \geq 16$	$l \geq 8$

5 Conclusions

The main aim of this dissertation was designing RNS based building blocks for applications in the field of DSP applications (binary-to-residue converter, residue-to-binary converter, residue adder and residue multiplier). The achieved results and key outcomes are summarized in this Chapter.

Throughout this thesis, a general introduction into the RNS and its properties, including its basics, advantages and disadvantages, have been presented in Chapter 1. Chapter 2 includes a brief survey on the recent trends and achievements in all RNS areas. The applications where RNS can be useful have been presented too. Chapter 3 states the main aims and objectives of this dissertation thesis. Chapter 4 details the proposed work and dissertation results. The main RNS components have been proposed including a binary to residue converter, modular adders, subtractors, multipliers, a residue comparator, components for overflow and sign detection and correction and a residue to binary converter. Moreover, discussions on the recently suggested moduli sets, the most efficient ones and those that should not be used have been also presented in Sections 4.1 and 4.8.1, respectively. A comparison between binary and RNS-based applications and the considerations that should be taken into account before designing a DSP application based on the RNS are also stated in Sections 4.7 and 4.8.2. The efficiency of those proposed designs have been proven by illustrating the implementation results and comparing them with already published designs. The majority of the proposed designs can be implemented with any system that has any moduli set of the form $(2^k \pm 1)$. Hence, the proposed components can be implemented with any RNS system that uses any of the published moduli sets.

The proposed designs and outcomes of the doctoral thesis have been published in different national and international conferences and journals.

5.1 Final remarks

The final points of this thesis can be summarized as follows,

- The RNS-based applications should be used in fields that have dominant multiplications or at least a mix of multiplications and additions rather than only additions.
- The moduli set should be chosen in such a way that it contains as few moduli of the form $(2^k + 1)$ as possible, due to the complexity and delay caused by this channel. The most efficient set has been proven to be $\{2^{n+1} - 1, 2^n, 2^n - 1\}$.

- Contrary to the prevalent issue, the number of moduli within a set is not as important as it is widely known. Moduli number does not play a crucial role in enhancing the speed of the RNS system.
- Indeed, the form and magnitude of the largest modulo are the main concerns that should be taken into account. The width of the largest modulo should be at least less than half of the dynamic range‘s width. A number of published moduli sets that should not be used have been stated in Section 4.8.1.
- Enlarging the dynamic range is more efficient than using overflow detection units, since such components present a considerably long delay. Therefore, timing performance of the RNS-based application that uses these components is worse than the one that has a larger dynamic range and does not contain overflow detection units.
- Using the RNS in very large DRs results in so-called “super-efficient” applications compared to binary ones, (they provide considerably higher timing performance, reduced area and power consumption).
- Using the RNS provides a kind of low-level security, since the operands and results are presented using a totally different system.
- Choosing the type of forward and reverse converters should depend on the dynamic range (more than 15 bits, the proposed converters have been proven to be more efficient than pure memory ones).

Hence, the points mentioned in the objectives of this thesis have been met. This work has led to new sights in the field of the residue number system. This system has very attractive and promising features if used in applications that require large and very large dynamic ranges. Rather than providing reduced power consumption with compromising timing performance or vice versa, both aspects can be fully met simultaneously. Since these two terms have become the key interests in nowadays technology, the RNS is the promising means that provides the so-called “super-efficient” applications.

Bibliography

[1] OMONDI, A., PREMKUMAR, B. *Residue Number System: Theory and Implementation*. London: Imperial College Press. 2007. 312 pages. ISBN-13: 978-1860948664.

[2] MOHAN, P.V.A., *Residue Number System: Algorithms and Architectures*. Massachusetts: Springer, 2002. 272 pages. ISBN-13: 978-1402070310.

[3] WANG, W., SWAMY, M.N.S., AHMAD, M.O. *Moduli Selection in RNS for Efficient VLSI Implementation*. In Proceedings of the International Symposium on Circuits and Systems, 2003, p. IV-512 – IV-515. ISBN 0-7803-7761-3.

[4] PIESTRAK, S.J. *A High-Speed Realization of a Residue to Binary Number System Converter*. In IEEE Trans. on Circuits and Systems-II: Analog and Digital Signal Processing, 1995, vol. 42, p. 661 – 663. ISSN 1057-7130.

[5] NAVI, K., MOLAHOSSEINI, A.S., ESMAEILDOUST, M. *How to Teach Residue Number System to Computer Scientists and Engineers*. In IEEE Trans. on Education, 2011, vol. 54, p. 156 – 163. ISSN 0018-9359.

[6] WANG, W., SWAMY, M.N.S., AHMAD, M.O., WANG, Y. *A High-Speed Residue-to-Binary Converter for Three-Moduli (2^k, $2^k - 1$, $2^{k-1} - 1$) RNS and a Scheme for its VLSI Implementation*. In IEEE Trans. on Circuits and Systems-II: Analog and Digital Signal Processing, 2000, vol. 47, p. 1576 – 1581. ISSN 1057-7130.

[7] MOHAN, P.V.A. *RNS-to-Binary Converter for a New Three-Moduli Set {$2^{n+1} - 1$, 2^n, $2^n - 1$}*. In IEEE Trans. on Circuits and Systems-II: Express Briefs, 2007, vol. 54, p. 775 – 779. ISSN 1549-7747.

[8] MOLAHOSSEINI, A.S., NAVI, K., RAFSANJANI, M.K. *A New Residue to Binary Converter Based on Mixed-Radix Conversion*. In 3rd International Conference on Information and Communication Technologies: From Theory to Applications, 2008, p. 1 – 6. ISBN 978-1-4244-1751-3.

[9] WANG, W., SWAMY, M.N.S., AHMAD, M.O., WANG, Y. A Study of the Residue-to-Binary Converters for the Three-Moduli Sets. In IEEE Trans. on Circuits and Systems-I: Fundamental Theory and Applications, 2003, vol. 50, p. 235 – 243. ISSN 1057-7122.

[10] HARIRI, A., NAVI, K., RASTEGAR, R. *A New High Dynamic Range Moduli Set with Efficient Reverse Converter*. In Computers & Mathematics with Applications Journal, 2008, vol. 55, p. 660 – 668. ISSN 0898-1221.

[11] MOHAN, P.V.A., PREMKUMAR, A.B. *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, $2^n + 1$, $2^{n+1} + 1$}*. In IEEE Trans. on Circuits and Systems-I: Regular Papers, 2007, vol. 54, p. 1245 – 1254. ISSN 1549-8328.

[12] CAO, B., CHANG, C.H., SRIKANTHAN, T. An Efficient Reverse Converter for the 4-Moduli Set {$2^n - 1$, 2^n, $2^n + 1$, $2^{2n} + 1$} Based on the New Chinese Remainder Theorem. In IEEE Trans. on Circuits and Systems-I: Fundamental Theory and Applications, 2003, vol. 50, p. 1296 – 1303. ISSN 1057-7122.

[13] MOLAHOSSEINI, A.S., NAVI, K., DADKHAH, C., KAVEHEI, O., TIMARCHI, S. *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. In IEEE Trans. on Circuits and Systems-I: Regular Papers, 2010, vol. 57, p. 823 – 835. ISSN 1549-8328.

[14] HIASAT, A.A. *VLSI Implementation of New Arithmetic Residue to Binary Decoders*. In IEEE Trans. on VLSI Systems, 2005, vol. 13, p. 153 – 158. ISSN 1063-8210.

[15] ZHANG, W., SIY, P. *An Efficient Design of Residue to Binary Converter for Four Moduli Set ($2^n - 1$, $2^n + 1$, $2^{2n} - 2$, $2^{2n+1} - 3$) Based on New CRT-II*. In Information Sciences Journal, 2008, vol. 178, p. 264 – 279. ISSN 0020-0255.

[16] CAO, B., CHANG, C.H., SRIKANTHAN, T. *A Residue-to-Binary Converter for a New Five-Moduli Set*. In IEEE Trans. on Circuits and Systems-I: Regular Papers, 2007, vol. 54, p. 1041 – 1049. ISSN 1549-8328.

[17] MOLAHOSSEINI, A.S., DADKHAH, C., NAVI, K. *A New Five-Moduli Set for Efficient Hardware Implementation of the Reverse Converter*," In IEICE Electronics Express, 2009, vol. 6, p. 1006 – 1012. ISSN 1006-1012.

[18] MOLAHOSSEINI, A.S., TEYMOURI, F., NAVI, K. *A New Four-Modulus RNS to Binary Converter*. In Proc. of IEEE International Symposium on Circuits and Systems, 2010, p. 4161 – 4164. ISBN 978-1-4244-5308-5.

[19] SOUSA, L., ANTÃO, S. *MRC-Based RNS Reverse Converters for the Four-Moduli Sets {$2^n + 1$, $2^n - 1$, 2^n, $2^{2n+1} - 1$} and {$2^n + 1$, $2^n - 1$, 2^{2n}, $2^{2n+1} - 1$}*. In IEEE Trans. on Circuits and Systems II: Express Briefs, 2012, vol. 59, p. 244 – 248. ISSN 1549-7747.

[20] ALIABADIAN, R., ALIABADIAN, A., BOLHASANI, A., HOSSEINI, S.Z., GOLSORKHTABAR, A. *A Novel High Dynamic Range 4-Module Set {2^{2n+1}, $2^{2n} + 1$, $2^n + 1$, $2^n - 1$} with Efficient Reverse Converter and Review Improving Modular Multiplication's Dynamic Range with this Module Set*. In International Conference on Computer Communication and Informatics, 2012, p. 1 – 6. ISBN 978-1-4577-1580-8.

[21] PIESTRAK, S.J. *Design of Residue Generators and Multioperand Modular Adders Using Carry-Save Adders*. In IEEE Transactions on Computers, 1994, vol. 43, p. 68–77. ISSN 0018-9340.

[22] BAYOUMI, M., JULLIEN, G., MILLER, W. *A VLSI Implementation of Residue Adders*. In IEEE Transactions on Circuits and Systems, 1987, vol. 34, p. 284-288. ISSN 0098-4094.

[23] JULLIEN, G.A. *Residue Number Scaling and Other Operations Using ROM Arrays*. In IEEE Transactions on Computers, 1978, vol. C-27, p. 325-336. ISSN 0018-9340.

[24] BANERJI, D.K. *A Novel Implementation Method for Addition and Subtraction in Residue Number Systems*. In IEEE Transactions on Computers, 1974, vol. C-23, p. 106-109. ISSN 0018-9340.

[25] TAYLOR, F.J. *A VLSI Residue Arithmetic Multiplier*. In IEEE Transactions on Computers, 1982, col. C-31, p. 540-546. ISSN 0018-9340.

[26] BEUCHAT, J.L. *Some Modular Adders and Multipliers for Field Programmable Gate Arrays*. In IPDPS '03 Proceedings of the 17th International Symposium on Parallel and Distributed Processing, 2003, ISSN 1530-2075.

[27] VERGOS, H.T., EFSTATHIOU, C. *Efficient Modulo $2^n + 1$ Adder Architectures*. In Integration, the VLSI Journal, 2009, vol. 42, p. 149–157. ISSN 0167-9260.

[28] VERGOS, H.T., EFSTATHIOU, C., NIKOLOS, D. *Diminished-One Modulo $2^n + 1$ Adder Design*. In IEEE Transactions on Computers, 2002, vol. 51, p. 1389-1399. ISSN 0018-9340.

[29] EFSTATHIOU, C., VERGOS, H.T., NIKOLOS, D. *Fast Parallel Prefix Modulo $2^n + 1$ Adders*. In IEEE Transactions on Computers, 2004, vol. 53, p. 1211-1216. ISSN 0018-9340.

[30] KALAMPOUKAS, L. et al. *High-Speed Parallel-Prefix Modulo $2^n - 1$ Adders*. In IEEE Transactions on Computers, 2000, vol. 49, p. 673–679. ISSN 0018-9340.

[31] VASSALOS1, E., BAKALIS1, D., VERGOS, H.T. *Novel Modulo 2^n+1 Subtractors*. In 16th International Conference on Digital Signal Processing, 2009, p. 1-5. ISBN 978-1-4244-3297-4.

[32] TIMARCHI, S., NAVI, K., HOSSEINZADE, M. *New Design of RNS Subtractor for Modulo $2^n +1$*. In Information and Communication Technologies, 2006, vol. 2, p. 2803-2808. ISBN 0-7803-9521-2.

[33] NEDJAH, N., MOURELLE, L.M. *A Review of Modular Multiplication Methods and Respective Hardware Implementations*. In Informatica Journal, 2006, vol. 30, p. 111–129. ISSN 0350-5596.

[34] HIASAT, A.A., ZOHDY, H.S. *Design and Implementation of a Fast and Compact Residue-Based Semi-Custom VLSI Arithmetic Chip.* In Proceedings of the 37th Midwest Symposium on Circuits and Systems, 1994, vol. 1, p. 428–431. ISBN 0-7803-2428-5.

[35] HIASAT, A.A. *New Memoryless, Mod $(2^n - 1)$ Residue Multiplier*. In Electronics Letters, 1992, vol. 28, p. 314-315. ISSN 0013-5194.

[36] BAJARD, J.C., DIDIER, L.S., KORNERUP, P. *An RNS Montgomery Modular Multiplication Algorithm*. In IEEE Transactions on Computers, 1998, vol. 47, p. 766–776. ISSN 0018-9340.

[37] WANG, Z., JULLIEN, G.A., MILLER, W.C. *An Efficient Tree Architecture for Modulo $2^n + 1$ Multiplication*. In Journal of VLSI Signal Processing Systems - Special Issue on VLSI Arithmetic and Implementations. 1996, vol. 14, p. 241–248. ISSN 0922-5773.

[38] WANG, Z., JULLIEN, G.A., MILLER, W.C. *An Algorithm for Multiplication Modulo $(2^n - 1)$.* In IEEE 39th Midwest Symposium Circuits Systems, 1996, vol. 3, p. 1301–1304. ISBN 0-7803-3636-4.

[39] EFSTATHIOU, C., VERGOS, H.T. *Modified Booth 1's Complement and Modulo $2^n -1$ Multipliers*. In The 7th IEEE International Conference on Electronics, Circuits and Systems, 2000, vol. 2, p. 637-640. ISBN 0-7803-6542-9.

[40] SOUSA, L., CHAVES, R. *A Universal Architecture for Designing Efficient Modulo $2^n + 1$ Multipliers*. In IEEE Transactions on Circuits and Systems I: Regular Papers, 2005, vol. 52, p. 1166–1178. ISSN 1549-8328.

[41] VEROGS, H.T., EFSTATHIOU, C. *Design of Efficient Modulo $2^n + 1$ Multipliers*. In IET Computers & Digital Techniques, 2007, vol. 1, p. 49–57. ISSN 1751-8601.

[42] WRZYSZCZ, A., MILFORD, D. *A New Modulo $2^\alpha + 1$ Multiplier*. In IEEE International Conference on Computer Design: VLSI in Computers and Processors, 1993, p. 614–617. ISBN 0-8186-4230-0.

[43] ASKARZADEH, M., HOSSEINZADEH, M., NAVI, K. *A New Approach to Overflow Detection in Moduli Set $\{2^n - 3, 2^n - 1, 2^n + 1, 2^n + 3\}$*. In Second International Conference on Computer and Electrical Engineering, 2009, vol. 1, p. 439 – 442. ISBN 978-1-4244-5365-8.

[44] SHANG, M., JIANHAO, H., LIN, Z., XIANG, L. *An Efficient RNS Parity Checker for Moduli Set {$2^n - 1$, $2^n + 1$, $2^{2n} + 1$} and its Applications*. In Springer Journal of Science in China Series F: Information Sciences, 2008, vol. 51, p. 1563 – 1571. ISSN 1862-2836.

[45] DIMAURO, G., IMPEDOVO, S., PIRLO, G. *A New Technique for Fast Number Comparison in the Residue Number System*. In IEEE Transactions on Computers, 1993, vol. 42, p. 608 – 612. ISSN 0018-9340.

[46] WANG, Y., SONG, X., ABDOULHAMID, M. *A New Algorithm for RNS Magnitude Comparison Based on New Chinese Remainder Theorem II*. In Proceedings Ninth Great Lakes Symposium on VLSI, 1999, vol. 1, p. 362 – 365. ISSN 1066-1395.

[47] BI, S., GROSS, W.J. *Efficient Residue Comparison Algorithm for General Moduli Sets*. In 48th Midwest Symposium on Circuits and Systems, 2005, vol. 2, p. 1601 – 1604. ISBN 0-7803-9197-7.

[48] Sousa, L. *Efficient Method for Magnitude Comparison in RNS Based on Two Pairs of Conjugate Moduli*. In 18th IEEE Symposium on Computer Arithmetic, 2007, vol. 1, p. 240 – 250. ISSN 1063-6889.

[49] ZAREI, B., ASKARZADEH, M., DERAKHSHANFARD, N., HOSSEINZADEH, M. *A High-Speed Residue Number Comparator for the 3-Moduli Set {2^n-1, 2^n+1, 2^n+3}*. In International Signals Systems and Electronics, 2010, vol. 1, p. 1 – 4. ISBN 978-1-4244-6352-7.

[50] NANNARELLI, A., RE, M., CARDARILLI, G.C. *Tradeoffs between Residue Number System and Traditional FIR Filters*. In IEEE International Symposium on Circuits and Systems, 2001, vol. 2, p. 305 – 308. ISBN 0-7803-6685-9.

[51] CONWAY, R., NELSON, J. *Improved RNS FIR Filter Architectures*. In IEEE Transactions on Circuits and Systems II: Express Briefs, 2004, vol. 51, p. 26–28. ISSN 1549-7747.

[52] SHAHANA, T.K., JAMES, R.K., JOSE, B.R., JACOB, K.P. *Performance Analysis of FIR Digital Filter Design: RNS Versus Traditional*. In International Symposium on Communications and Information Technologies, 2007, vol.1, p. 1 – 5. ISBN 978-1-4244-0977-8.

[53] FREKING, W.L., PARHI, K.K. *Low-Power FIR Digital Filters Using Residue Arithmetic*. In Conference Record of the 31st Asilomar Conference on Signals, Systems & Computers, 1997, vol. 1, p. 739-743. ISBN 0-8186-8316-3. (LOW POWER)

[54] CARDARILLI, G.C., DEL RE, A., NANNARELLI, A., RE, M. *Low Power and Low Leakage Implementation of RNS FIR Filters*. In Conference Record of the Thirty-Ninth Asilomar Conference on Signals, Systems and Computers, 2005, vol. 1, p. 1620 – 1624. ISSN 1058-6393. (LOW POWER)

[55] CARDARILLI, G.C, NANNARELLI, A., RE, M. *Residue Number System for Low-Power DSP Applications*. In Conference Record of the Forty-First Asilomar Conference on Signals, Systems and Computers, 2007, vol. 1, p. 1412 – 1416. ISSN 1058-6393.

[56] RAMNARAYAN, R., TAYLOR, F. *Analysis of Errors in Residue Number System (RNS) Based IIR Digital Filters*. In IEEE International Conference on Acoustics, Speech and Signal Processing, 1982, vol. 7, p. 56 – 59.

[57] SODERSTRAND, M.A., SINHA, B. *A Pipelined Recursive Residue Number System Digital Filter*. In IEEE Transactions on Circuits and Systems, 1984, vol. 31, p. 415 – 417. ISSN 0098-4094.

[58] AMMAR, A., AL KABBANY, A., YOUSSEF, M., AMAM, A. *A Secure Image Coding Scheme Using Residue Number System*. In Proceedings of the Eighteenth National Radio Science Conference, 2001, vol. 2, p. 339 – 405. ISBN 977-5031-68-0.

[59] WANG, W., SWAMY, M.N.S., AHMAD, M.O. *RNS Application for Digital Image Processing*. In 4th IEEE international workshop on system-on-chip for real-time applications, 2004, vol. 1, p. 77–80. ISBN 0-7695-2182-7.

[60] TALESHMEKAEIL, D.K., MOUSAVI, A. *The Use of Residue Number System for Improving the Digital Image Processing*. In IEEE 10th International Conference on Signal Processing, 2010, vol. 1, p. 775–780. ISBN 978-1-4244-5897-4.

[61] TALESHMEKAEIL, D.K., MOHAMAMDZADEH, H., MOUSAVI, A. *Using Residue Number System for Edge Detection in Digital Images Processing*. In IEEE 3rd International Conference on Communication Software and Networks, 2011, vol. 1, p. 249–253. ISBN 978-1-61284-485-5.

[62] MOHARRAMI, S., TALESHMEKAEIL, D.K. *The Application of the Residue Number System in Digital Image Processing: Propose a Scheme of Filtering in Spatial Domain*. In Research Journal of applied science, 2012, vol. 7, p. 286–292. ISSN 1815-932X.

[63] YANG, L.L., HANZO, L. *Redundant Residue Number System Based Error Correction Codes*. In IEEE Vehicular Technology Conference, 2001, vol. 3, p. 1472–1476. ISBN 0-7803-7005-8.

[64] PONTARELLI, S., CARDARILLI, G.C., RE, M., SALSANO, A. *A Novel Error Detection and Correction Technique for RNS Based FIR Filters*. In IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems, 2008, vol. 1, p. 436 – 444. ISSN 1550-5774.

[65] YANG, L.L., HANZO, L. *A Residue Number System Based Parallel Communication Scheme Using Orthogonal Signaling .I*. In System outline. In IEEE Transactions on Vehicular Technology, 2002, vol. 51, p. 1534-1546. ISSN 0018-9545.

[66] YOUSSEF, M.I., EMAM, A.E., ABD ELGHANY M. *Direct Sequence Spread Spectrum Technique with Residue Number System*. In International Journal of Electrical & Electronics Engineering, 2009, vol. 3, p. 223-230. ISSN 2010-3972.

[67] BAJARD, J.C., IMBERT, L. *Brief Contributions: A Full RNS Implementation of RSA*. In IEEE Transactions on Computers, 2004, vol. 53, p. 769-774. ISSN 0018-9340.

[68] TIMARCHI, S., NAVI, K. *Improved Modulo 2^n+1 Adder Design*. In World Academy of Science Engineering and Technology, 2008, vol. 39, p. 577 – 584. ISSN 2010-3778.

[69] HAOHUAN, F., MENCER, O., LUK, W. *Optimizing Residue Arithmetic on FPGAs*. In International Conference on ICECE Technology, 2008, vol. 1, p. 41 – 48. ISBN 978-1-4244-2796-3.

[70] ZIMMERMANN, R. *Efficient VLSI Implementation of Modulo ($2^n \pm 1$) Addition and Multiplication*. In Proceedings of 14th IEEE Symposium on Computer Arithmetic, 1999, vol. 1, p. 158 – 167. ISBN 0-7695-0116-8.

[71] HIASAT, A., SWEIDAN, A. *Residue Number System to Binary Converter for the Moduli Set (2^{n-1}, 2^n-1, 2^n+1)*. In Journal of Systems Architecture: the EUROMICRO Journal, 2003, vol. 49, p. 53 – 58. ISSN 1383-7621.

[72] SMITH, S.W. *Digital Signal Processing: a Practical Guide for Engineers and Scientists*. USA: Newnes an Imprint of Elsiever.2003. 650 pages. ISBN-13:978-0-7506-7444-7.

[73] XILINX, *Virtex-4 FPGA User Guide*. 2008, 406 pages. [Online] Cited 2012-09-04. Available at http://www.xilinx.com/support/documentation/user_guides/ug070.pdf

[74] XILINX, *XtremeDSP for Virtex-4 FPGAs*. 2008, 121 pages. [Online] Cited 2012-09-11. Available at http://www.xilinx.com/support/documentation/user_guides/ug073.pdf

Author's publications

[75] YOUNES, D., STEFFAN, P. *A Comparative Study on Different Moduli Sets in Residue Number System*. In International Conference on Computer Systems and Industrial Informatics, Dubai, UAE, 2012, vol. 1, p. 1 – 6. ISBN 978-1-4673-5155-3.

[76] YOUNES, D., STEFFAN, P., *A Detailed Study on the Moduli Number Effect on RNS Timing Performance*. In Journal of Emerging Trends in Computing and Information Sciences, Islamabad, Pakistan, 2013, vol. 4, p. 85 – 93. ISSN 2079-8407.

[77] YOUNES, D., STEFFAN, P. *Novel Architectures of Modulo $2^n \pm 1$ Adders for Field Programmable Gate Array*. In Electronic Devices and Systems IMAPS CS International conference, Brno, Czech republic, 2011, vol. 1, p. 51 – 56. ISBN 978-80-214-4303- 7.

[78] YOUNES, D., STEFFAN, P. *New Structures of $2^n \pm 1$ Modular Adders for FPGAs*. In ElectroScope Journal, Czech Republic, 2011, vol. 5, p. 11 – 14. ISSN 1313-1842.

[79] YOUNES, D., STEFFAN, P. *Improved Design for Modulo $2^n + 1$ Adder*. In Electronic Devices and Systems IMAPS CS International Conference, Brno, Czech republic, 2010, vol. 1, p. 346 – 348. ISBN 978-80-214-4138- 5.

[80] YOUNES, D., STEFFAN, P. *Novel Modulo $2^n + 1$ Subtractor and Multiplier*. In the Sixth International Conference on Systems ICONS, St. Maarten, the Netherlands Antilles, 2011, vol. 1, p. 36 – 38. ISBN 978-1-61208-002- 4.

[81] YOUNES, D., STEFFAN, P. *Efficient Method for Overflow Detection and Correction in Residue Number System*. In Electronic Devices and Systems IMAPS CS International Conference, Brno, Czech republic, 2012, vol. 1, p. 183 – 188. ISBN 978-80-214-4539- 0.

[82] YOUNES, D., STEFFAN, P. *Universal Approaches for Overflow and Sign Detection in Residue Number System Based on $\{2^n - 1, 2^n, 2^n + 1\}$*. In the Eighth International Conference on Systems, Seville, Spain, 2013, vol. 1, p. 1 – 5. ISBN 978-1-61208-246- 2.

[83] YOUNES, D., STEFFAN, P. *FPGA Implementation of Residue-to-Binary Converters: A Comparison between New CRT-I and MRC Converters for the Moduli Set $(2^n - 1, 2^n, 2^n + 1)$*. In Electronics Journal, Sofia, Bulgaria, 2011, vol. 5, p. 11 – 14. ISSN 1313- 1842.

[84] YOUNES, D., STEFFAN, P. *Efficient Image Processing Application Using Residue Number System*. In 20th International Conference Mixed Design of Integrated Circuits and Systems, Gdynia, Poland, 2013. p. 468 – 472. ISBN 978-83-63578-00- 8.

[85] YOUNES, D., STEFFAN, P. *Fast and Power Reduced RNS-Based Image Filtering in Spatial Domain*. In ElectroScope Journal, Czech Republic, 2013. ISSN 1313-1842. Under review.

[86] YOUNES, D., STEFFAN, P. *Efficient Reverse Converter and Residue Comparator Based on a Novel Algorithm in RNS*. In IEICE Electronics Express Journal. ISSN 1349-2543. Under review.

6 Appendix

Tab. 6.1: Delay and hardware complexity of different components using unit gate model

Component	Delay (T)	Area (A)
NOT gate, circular shifting and bits rearrangement	ignored	ignored
OR, AND, NOR, NAND gates	1	1
XOR, XNOR gates	2	2
2:1 multiplexer	2	3
Half adder (HA)	2	3
Full adder (FA)	4	7
Carry propagate adder (CPA) of n bits	$4n$	$7n$
Carry propagate adder with end-around carry (CPA-EAC) of n bits	$8n$	$7n$
Carry save adder (CSA) of n bits	4	$7n$
Carry save adder with end-around carry (CSA-EAC) of n bits	4	$7n$
General modulo adder of n bits [5]	$8n + 3$	$17n + 1$
Modulo $(2^n - 1)$ adder (1^{st} complement adder) [1]	$8n$	$7n$
Modulo $(2^n + 1)$ adder [68]	$8n + 11$	$17n + 18$
Proposed modulo $(2^n + 1)$ adder [79]	$8n + \log_2 n + 4$	$\frac{35}{2}n + \log_2 n + 7$
Binary multiplier (array multiplier [1]) of n bits	$8n - 7$	$8n^2 - 11n$
General modulo multiplier (based on the product-partitioning method [1]) of n bits	$24n - 11$	$16n^2 - 5n + 1$
Modulo $(2^n - 1)$ multiplier	$16n - 7$	$8n^2 - 4n$
Modulo $(2^n + 1)$ multiplier [80]	$16n + 9$	$8n^2 + 22n + 15$

Tab. 6.2: Comparison between reverse converters, modular adders and modular multipliers for systems based on sets that provide DR = 3*n*

Moduli set	DR	*n* odd/ even	mod #	RC		Critical Channel	Modular adders		Modular multipliers	
				Delay	*Complexity*		*Delay*	*Complexity*	*Delay*	*Complexity*
$\{2^n - 1, 2^n, 2^n + 1\}$ [4]	3*n*	any	3	16*n* + 8	**31*n* + 13**	$(2^n + 1)$	8*n* + 11	38*n* + 18	16*n* + 12	$24n^2 + 7n + 15$
$\{2^{n-1} - 1, 2^n - 1, 2^n\}$ [6]	3*n*–1	any	3	24*n* – 2	54*n* – 45	$(2^n - 1)$	**8*n***	**21*n* – 7**	**16*n* – 7**	**$24n^2 - 35n + 12$**
$\{2^n - 1, 2^n, 2^{n+1} - 1\}$ [7]	3*n*+1	any	3	**8*n* + 30**	110*n* + 159	$(2^{n+1} - 1)$	8*n* + 8	21*n* + 7	16*n* + 9	$24n^2 - 27n + 4$

Tab. 6.3: Comparison between reverse converters, modular adders and modular multipliers for systems based on sets that provide DR = 4*n*

Moduli Set	DR	*n* odd/ even	mod #	RC		Critical Channel	Modular Adders		Modular Multipliers	
				Delay	*Complexity*		*Delay*	*Complexity*	*Delay*	*Complexity*
$\{2^n - 1, 2^n, 2^{2n+1} - 1\}$ [8]	4*n*+1	any	3	40*n* + 20	69*n* + 20	$(2^{2n+1} - 1)$	16*n*+ 8	**38*n* + 25**	32*n* + 9	$48n^2 + 9n + 4$
$\{2^n - 1, 2^n + 1, 2^{2n} + 1\}$ [9]	4*n*	any	3	**32*n* + 8**	**62*n* + 8**	$(2^{2n} + 1)$	16*n* + 11	58*n* + 36	32*n* + 12	$48n^2 + 62n + 15$
$\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$ [11]-I	4*n*+1	even	4	46*n* + 28	$15n + 5 + 7/2(n^2 - 3n - 4)$	$(2^n + 1)$	**8*n* + 11**	45*n* + 25	**16*n* + 12**	**$32n^2 + 19n + 19$**
$\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} + 1\}$ [11]-II	4*n*+1	odd	4	48*n* + 62	$7n^2 + 102n + 108$	$(2^{n+1} + 1)$	8*n* + 19	55*n* + 53	16*n* + 28	$32n^2 + 45n + 60$
$\{2^{n/2} - 1, 2^{n/2} + 1, 2^n + 1, 2^{2n+1} - 1\}$ [18]	4*n*+1	even	4	32*n* + 37	68*n* + 37	$(2^{2n+1} - 1)$	16*n* + 8	43*n* + 43	32*n* + 9	$44n^2 + 59n + 34$

Tab. 6.4: Comparison between reverse converters, modular adders and modular multipliers for systems based on sets that provide DR = 5*n*

Moduli Set	DR	*n* odd/ even	mod #	RC		Critical Channel	Modular Adders		Modular Multipliers	
				Delay	*Complexity*		*Delay*	*Complexity*	*Delay*	*Complexity*
$\{2^n, 2^{2n} - 1, 2^{2n} + 1\}$ [10]	5*n*	even	3	**32*n* + 4**	**44*n* + 8**	$(2^{2n} + 1)$	16*n* + 11	55*n* + 18	32*n* + 12	$72n^2 + 25n + 15$
$\{2^n - 1, 2^n, 2^n + 1, 2^{2n} + 1\}$ [12]	5*n*	any	4	32*n* + 12	95*n* + 39	$(2^{2n} + 1)$	16*n* + 11	72*n* + 36	32*n* + 12	$56n^2 + 51n + 30$
$\{2^n - 1, 2^n, 2^n + 1, 2^{2n+1} - 1\}$ [13]-I	5*n*+1	any	4	48*n* + 20	74*n* + 14	$(2^{2n+1} - 1)$	16*n* + 8	52*n* + 25	32*n* + 9	$56n^2 + 31n + 19$
$\{2^n - 1, 2^n, 2^n + 1, 2^n - 2^{(n+1)/2} + 1, 2^n + 2^{(n+1)/2} + 1\}$ [14]	5*n*	odd	5	32*n* + 20	184*n* – 9	$(2^n + 2^{(n+1)/2} + 1)$	**8*n* + 11**	72*n* + 37	24*n* + 13	$56n^2 + 29n + 28$
$\{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1, 2^{n+1} + 1\}$ [16]	5*n*	even	5	72*n* + 4*l* + 28	$7\times(5n^2 + 43n + m)/6 + 112n - 7$	$(2^{n+1} + 1)$	8*n* + 19	62*n* + 36	**16*n* + 28**	**$40n^2 + 25n + 72$**
$\{2^n, 2^{n/2} - 1, 2^{n/2} + 1, 2^n + 1, 2^{2n-1} - 1\}$ [17]	5*n*–1	even	5	52*n* + 4	97*n* + 11	$(2^{2n-1} - 1)$	16*n* – 8	**50*n* + 29**	32*n* – 23	$52n^2 - 20n + 42$

Tab. 6.5: Comparison between reverse converters, modular adders and modular multipliers for systems based on sets that provide DR = 6*n*

Moduli Set	DR	*n* odd/ even	mod #	RC		Critical Channel	Modular Adders		Modular Multipliers	
				Delay	*Complexity*		*Delay*	*Complexity*	*Delay*	*Complexity*
$\{2^n - 1, 2^n + 1, 2^{2n} - 2, 2^{2n+1} - 3\}$ [15]	6*n*+1	any	4	56*n* + 39	188*n* + 57	$(2^{2n+1} - 3)$	16*n* + 11	92*n* + 37	48*n* + 13	$144n^2 + 62n + 28$
$\{2^n - 1, 2^{2n}, 2^n + 1, 2^{2n} + 1\}$ [13]-II	6*n*	any	4	**32*n* + 12**	**88*n* + 24**	$(2^{2n} + 1)$	16*n* + 11	72*n* + 19	**28*n* + 12**	$80n^2 + 40n + 30$
$\{2^n + 1, 2^n - 1, 2^{2n}, 2^{2n+1} - 1\}$ [19]	6*n*+1	any	4	40*n*	98*n* + 7	$(2^{2n+1} - 1)$	**16*n* + 8**	**52*n* + 25**	32*n* + 9	**$80n^2 + 20n + 19$**
$\{2^{2n+1}, 2^{2n} + 1, 2^n + 1, 2^n - 1\}$ [20]	6*n*+1	any	4	**32*n* + 12**	88n + 28	$(2^{2n} + 1)$	16*n* + 11	72*n* + 34	**28*n* + 12**	$80n^2 + 72n + 27$

Tab. 6.6: The maximum frequency and power consumption of application performing a number of iterated additions using the RNS and BNS for DR = 12 bits (implemented on Virtex-4 XC4VSX25 FPGA)

Number of iterated adding operations	RNS-based		Binary-based		RNS improvement%	
	Max freq. [MHz]	PWR at 100 MHz [mW]	Max freq. [MHz]	PWR at 100 MHz [mW]	Freq. increment %	PWR reduction %
1	220.6	425	788	345	-	-
5	225.6	440	215.8	389	4.5%	-
10	193.9	454	123.9	395	56.5%	-
15	156.6	442	100.7	396	55.5%	-
20	141.5	460	82	399	72.6%	-

Tab. 6.7: The maximum frequency and power consumption of application performing a number of iterated additions using the RNS and BNS for DR = 24 bits (implemented on Virtex-4 XC4VSX25 FPGA)

Number of iterated adding operations	RNS-based		Binary-based		RNS improvement%	
	Max freq. [MHz]	PWR at 100 MHz [mW]	Max freq. [MHz]	PWR at 100 MHz [mW]	Freq. increment %	PWR reduction %
1	186.5	458	446.6	417	-	-
5	174.3	484	204	444	-	-
10	158.1	500	111.6	452	41.7%	-
15	147.6	534	95.6	455	54.4%	-
20	126.8	550	69.1	462	83.5%	-

Tab. 6.8: The maximum frequency and power consumption of application performing a number of iterated additions using the RNS and BNS for DR = 33 bits (implemented on Virtex-4 XC4VSX25 FPGA)

Number of iterated adding operations	RNS-based		Binary-based		RNS improvement%	
	Max freq. [MHz]	PWR at 100 MHz [mW]	Max freq. [MHz]	PWR at 100 MHz [mW]	Freq. increment %	PWR reduction %
1	161.5	496	316.3	443	-	-
5	166.5	517	173.7	484	-	-
10	150.5	568	105.9	493	42.1%	-
15	116.8	589	93.2	499	25.3%	-
20	107.7	614	66.7	504	61.5%	-

Tab. 6.9: The maximum frequency and power consumption of application performing a number of iterated additions using the RNS and BNS for DR = 48 bits (implemented on Virtex-4 XC4VSX25 FPGA)

Number of iterated adding operations	RNS-based		Binary-based		RNS improvement%	
	Max freq. [MHz]	PWR at 100 MHz [mW]	Max freq. [MHz]	PWR at 100 MHz [mW]	Freq. increment %	PWR reduction %
1	115.7	703	307.9	493	-	-
5	116.7	758	158.9	555	-	-
10	103.1	802	100.1	567	3%	-
15	114.9	833	87.8	575	30.9%	-
20	108.2	867	63.4	583	70.7%	-

Tab. 6.10: The maximum frequency and power consumption of application performing a number of iterated multiplications using the RNS and BNS for DR = 12 bits (implemented on Virtex-4 XC4VSX25 FPGA)

Number of iterated multiplying operations	RNS-based		Binary-based		RNS improvement%	
	Max freq. [MHz]	PWR at 100 MHz [mW]	Max freq. [MHz]	PWR at 100 MHz [mW]	Freq. increment %	PWR reduction %
1	141.4	410	146.8	340	-	-
5	137.4	417	43.5	343	215.9%	-
10	134.6	425	22.7	347	493%	-
15	134.1	435	15.4	348	770.8%	-
20	132.2	441	11.7	350	1029.9%	-

Tab. 6.11: The maximum frequency and power consumption of application performing a number of iterated multiplications using the RNS and BNS for DR = 24 bits (implemented on Virtex-4 XC4VSX25 FPGA)

Number of iterated multiplying operations	RNS-based		Binary-based		RNS improvement%	
	Max freq. [MHz]	PWR at 100 MHz [mW]	Max freq. [MHz]	PWR at 100 MHz [mW]	Freq. increment %	PWR reduction %
1	135.4	419	92.2	355	46.9%	-
5	125.9	437	35.9	360	250.7%	-
10	124.5	465	19.9	366	525.6%	-
15	121.3	476	13.7	380	785.4%	-
20	120.6	489	10.5	383	1048.6%	-

Tab. 6.12: The maximum frequency and power consumption of application performing a number of iterated multiplications using the RNS and BNS for DR = 33 bits (implemented on Virtex-4 XC4VSX25 FPGA)

Number of iterated multiplying operations	RNS-based		Binary-based		RNS improvement%	
	Max freq. [MHz]	PWR at 100 MHz [mW]	Max freq. [MHz]	PWR at 100 MHz [mW]	Freq. increment %	PWR reduction %
1	125.8	457	78.7	360	59.9%	-
5	122.9	486	26.4	362	365.5%	-
10	121.6	506	12.8	676	850%	25.2%
15	120.1	529	9.1	832	1219.8%	36.4%
20	119.8	545	7	1026	1611.4%	46.9%

Tab. 6.13: The maximum frequency and power consumption of application performing a number of iterated multiplications using the RNS and BNS for DR = 48 bits (implemented on Virtex-4 XC4VSX25 FPGA)

Number of iterated multiplying operations	RNS-based		Binary-based		RNS improvement%	
	Max freq. [MHz]	PWR at 100 MHz [mW]	Max freq. [MHz]	PWR at 100 MHz [mW]	Freq. increment %	PWR reduction %
1	114.7	663	54.7	411	109.7%	-
5	113.7	684	20.9	661	444%	3.5%
10	106.7	716	12.6	1189	749.8%	39.8%
15	114.1	757	8.9	1567	1182%	51.7%
20	105	777	Too large to be implemented on device	Too large to be implemented on device		