Imperial Journal of Interdisciplinary Research (IJIR)
Vol-2, Issue-4, 2016
ISSN: 2454-1362, http://www.onlinejournal.in

Optimizing the Precision of Digital Signal Processors Using Residue Number System

V. Bavya[1] & Ms R.Uthira Devi[2].
ME-Applied Electronics[1], Dept Of Electronics and Communication Engineering[2].
Sri Eshwar College of Engineering,Coimbatore[1,2].

Abstract— *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. This property of RNS is very helpful to reduce the complexity of calculation in many applications. In RNS, the arithmetic operations are spilt into smaller parallel operations which are independent of each other .It offers a promising future in VLSI because of its carry free operation in addition, subtraction and multiplication. Hence devices operating in this principle inherit property of high speed and low power consumption. Using this efficient moduli set, a guideline for a Reconfigurable processor is presented here that can process some predefined functions. As RNS minimizes the carry propagation, the scheme can be implemented in real time signal processing and other fields where high speed computations are required. Hence devices operating in this Principle inherit property of high speed and low power consumption . 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.*

1.INTRODUCTION

This 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,

$$X_i = X \bmod m_i = \langle x_i \rangle m_i ; \quad 0 \le x_i < m_i \qquad (1.1)$$

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,

$$X \xrightarrow{RNS} (\langle x_1 \rangle m_1 , \langle x_2 \rangle m_{2,\ldots,} \langle x_n \rangle m_n);$$

$$\text{GCD}(m_i, m_j) = 1; \qquad (1.2)$$

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.

$$M = \prod_{i=1}^{n} m_i \qquad (1.3)$$

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

$$0 \le X < M \qquad (1.4)$$

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

$$0 \le X < [M/2] \quad \text{for positive numbers}$$
$$[M/2] \le X < M \quad \text{for negative number} \qquad (1.5)$$

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 [7].

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.

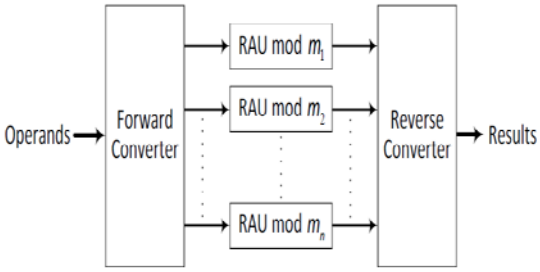


Fig. 1: The architecture of the residue number system (RNS)

The basic RNS processor's architecture is shown in Fig. 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).

2. SCHEDULING ALGORITHMS FOR PARALLELISM

The architecture of the reconfigurable RNS processor having a single number of unit devices is quite simple. But to increase the parallelism and decrease the processing time in real time mode, multiple unit devices can be used. The function can be broken into parts and can be arranged in such a way that the independent parts can be processed simultaneously. So to obtain the fastest processing design within the constrains specified, proper scheduling needs to be done, which basically arrange the behavioral operations to control the tasks. The scheduling algorithm will take the control and data flow graph as input and will generate the temporal ordering of individual operation, i.e. the states of the finite state machine. There are 3 scheduling algorithms [1] which are mostly used. They are as follows.

As Soon As Possible (ASAP) algorithm generates the ordering from the Data Flow Graph by a breadth-first search starting from the data sources to the sinks. It starts with the highest nodes in the Data Flow Graph that have no parents, and assigns time steps in increasing order as it proceeds downwards. This algorithm follows the simple rule that a successor node can execute only after its parent has executed. The advantage of ASAP is that it gives fastest schedule, as it requires least number of control steps and does not consider resource constraints.

As Late As Possible (ALAP) algorithm works very similar to the ALAP algorithm, except that it starts at the bottom of the Data Flow Graph and proceeds upwards. ALAP usually gives a bad solution, giving slowest possible schedule, which takes the maximum number of control steps. It does not necessarily reduce the number of functional units needed.

In ***Resource Constrained Scheduling (RC)*** algorithm, there is a constraint on the number of resources that can be used. One of the most popular methods is List-Based Scheduling, which is basically the generalization of ASAP scheduling, since it produces the same result in absence of resource constraints. It maintains a priority list of "ready" nodes. During each of the iteration, it tries to use up all resources in that state by scheduling operations in the head of the list. For conflicts, the operator with higher priority will be scheduled first.

3. RECONFIGURABLE ARCHITECTURE

The general architecture of a reconfigurable RNS processor is shown in Fig. 2 [5]. Given a moduli set hardware complexity depends on the functionalities of the RNS. Because of the

space issue, a simplified structure is shown using only three arithmetic operators. It contains

a. 2 nos. of Binary to RNS converters
b. 7 nos. of MUXs
c. Adder
d. subtractor
e. Multiplier
f. RNS to Binary converter [3][4]

Binary numbers are passed to the processor as inputs which first are converted to the RNS number. Here the selection of moduli is very much important because the proper selection of moduli optimizes the bit efficiency, area of the processor and time to process the particular function.

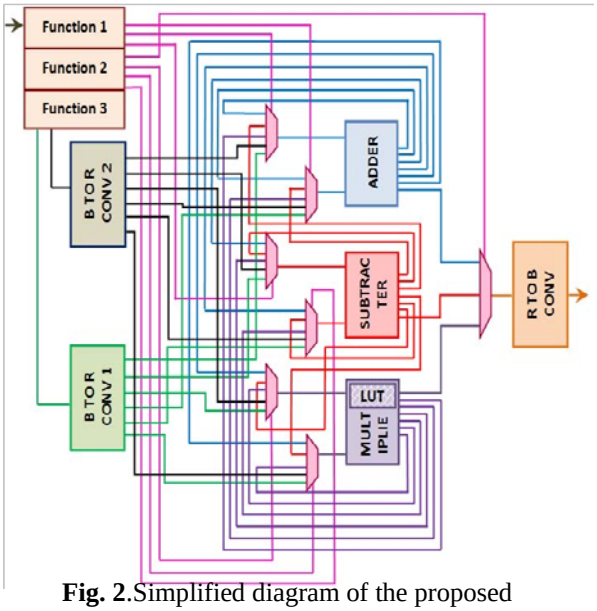


Fig. 2.Simplified diagram of the proposed Reconfigurable RNS Processor

After the conversion of the binary number to its corresponding residue representation, the arithmetic operations can be performed. As any binary number produces a set of RNS numbers depending upon the number of moduli used, m copies of arithmetic units (adder, subtractor, multiplier etc) are required to perform some arithmetic operation of a number when it is converted to RNS, where m is the number of moduli used in that scheme. As the residues can be independently operated, parallel arithmetic operations can be performed on the residue set.

Fig. 3 [5] shows the control and data flows between the various paths. The Programmable Controller can program the reconfigurable RNS Processor directly or Programmable Memory is used to store the bit stream. Programmable Controller is governed by the General purpose CPU.

In general, all modular arithmetic operations like Binary to RNS conversion or RNS addition, multiplication are implemented in chip by using two different methods [2]. One is the table look-ups, implemented by PLA. Second one is the Hybrid Methods, which is the combination of the legacy hardware, like full adders, with a table look-up, which can be used to convert the output of the legacy hardware to the correct residue format.

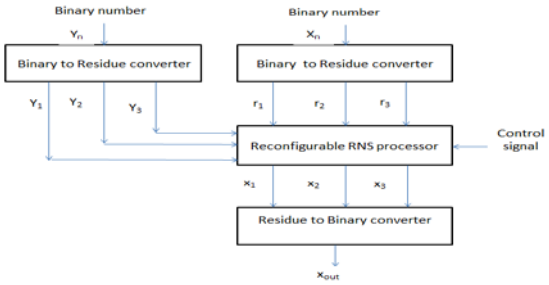


Fig 3.Control and Data Flow in the proposed Reconfigurable RNS Processor

when we are giving binary input to binary-to-residue converter, it gives residue output and vice versa .The output from the Yn binary to residue converter also given as the input to processor. Processor uses reconfigurable control signals for performing some arithmetic operations. The output of residues is given as the input to residue-to-binary converter.

3.1 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 [20].

In order to illustrate the forward conversion process, forward conversion equations corresponding to the special moduli set $\{2n - 1, 2n, 2n + 1\}$ will be stated [6], [7]. Actually, by modifying k, these equations can be applied with any modulo of the form $(2k \pm 1)$.

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

$$X=(b_{3n-1}b_{3n-2}\ldots.b_nb_{n-1}\ldots b_0)_2=B_12^{2n}+B_22^n+B_3 \quad (3.1)$$

where, b3n-1 … b0 are the binary digits (bits) of X. $B1$, $B2$, $B3$ are blocks, each of them contains n bits. The RNS representation of X according to the moduli set $\{2n - 1, 2n, 2n + 1\}$,

$$<x1>_{2n-1}=<X>_{2n-1}=<B_12^{2n}+B_22^n+B_3>_{2n-1}=<B_1+B_2+B_3>_{2n-1} \quad (3.2)$$

$$<x2>_{2n-1}=<X>_{2n}=<B_12^{2n}+B_22^n+B_3>_{2n}=<B_3>_{2n} \quad (3.3)$$

$$<x3>_{2n+1}=<X>_{2n+1}=<B_12^{2n}+B_22^n+B_3>_{2n+1}=<B_1-B_2+B_3>_{2n+1} \quad (3.4)$$

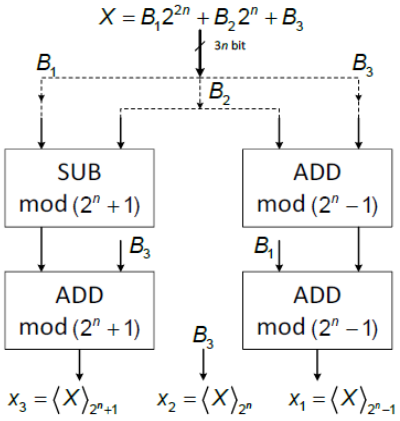


Fig. 4: RNS forward converter for the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$

Equations (3.2),(3.3) and (3.4) are extracted based on the following

$<2^n>_{2n-1}=<2^n-1+1>_{2n-1}=<1>_{2n-1}$

$<2^n>_{2n}=<0>_{2n}$

$<2^n>_{2n+1}=<2^n-1+1>_{2n+1}=<1>_{2n+1}$ (3. 5)

According to equations (3.2), (3.3) and (3.4), the general structure of the forward converter for the moduli set $\{2n - 1, 2n, 2n+1\}$is shown in Fig. 4 [1].

3.2 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,

$< x X x >^{-1}$; $0 \le x , x^{-1} < m$ (3. 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 ($x1$, $x2$,…, $x\ n$) by the following equation [1], [2],

$$X = \left\langle \sum_{i=1}^{n} \langle x_i N_i \rangle_{m_i} M_i \right\rangle_M \quad (3.7)$$

Where ,M= $m_1 x\ m_2 x \ldots x\ m_n$, $M_i = M/m_i$ and $N_i = <M_i^{-1}>_{m_i}$ is the multiplicative inverse of M_i relative to modulo mi.

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 [8], [9], [10], [13] and [16].

By using the MRC, a residue number ($x1$, $x2$,…, xn) can be converted back into its weighted equivalent X by,

$X=v_n \prod m_i + \ldots.. v_3 m_2 m_1 + v_2 m_1 + v_1$ (3. 8)

Where

$v_1 = x_1$ (3. 9)

$v_2 = <(x_2 - v_1) \times <m_1^{-1}>_{m_2}>_{m_2}$ (3. 10)

$v_3 = <(x_3 - v_1) \times <m_1^{-1}>_{m_3} - v_2) <m_2^{-1}>_{m_3}>_{m_3}$ (3. 11)

As illustrated in equation (3.8), 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 [10], [11]. 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 [18], [19].

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 ($x1$, $x2$,…, xn) can be converted back into its weighted equivalent X by,

$X = x_1 + m_1 x <k_1 (x_2 - x_1) + k_2 (x_3 - x_2) + \ldots.. k_{n-1} m_2 m_3 \ldots.. m_{n-1} (x_n - x_{n-1})>_{m2m3\ldots\ldots mn}$ (3. 12)

Where,

$<k_1 x m_1 > m_2 m_3 \ldots\ldots m_n = 1$ (3. 13)

$<k2 x m_1 x\ m_2 > m_3 \ldots.. m_n = 1$ (3. 14)

$<k_{n-1} x m_1 x m_2 x \ldots\ldots x m_{n-1} > m_n = 1$ (3. 15)

As can be noticed in equation (3.12), the final modular adder is reduced by one modulo. This can bring a great benefit when the first modulo is of the $2k$ form, and the multiplication of the rest moduli is of the $(2k - 1)$ form. Such reverse converters are reported in [12], [14].

The new CRT-II even further reduces the size of the final modular adder. A residue number ($x1$, $x2$,…, xn) can be converted back into its weighted equivalent X by the new CRT-II by,

$X = Z + m_1 m_2 <k_1 (y - z)>_{m3m4}$

$Z = x_1 + m_1 <k_2 (x_2 - x_1)>_{m2}$

$Y = x_3 + m_3 <k_3 (x_4 - x_3)>_{m4}$ (3. 16)

where,

$\langle k_1 m_1 m_2 \rangle_{m3m4}=1, \langle k_2 m_1 \rangle_{m2}=1, \langle k_3 m_3 \rangle_{m4}=1$

This algorithm is very efficient. It is used with sets that have three moduli. Reverse converters based on the new CRT-II are presented in [15], [17].

3.3 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 decrease-delay and increase area applications.

$X=(x_1,x_2,.....,x_n)$ $Y=(y_1,y_2,.....,y_n)$

$Z=X^{o}Y \equiv (\langle x_1{}^{o}y_1 \rangle_{m1}, \langle x_2{}^{o}y_2 \rangle_{m2} \cdots\cdots \langle x_n{}^{o}y_n \rangle_{mn}$; $^{o} \equiv (+,-,x)$

$Z=(z_1,z_2,.....,z_n)$

(3.17)

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 [6], [7], [20]. The first approach is realized by using ROMs [21]. 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 [22]. A RAU based on the third approach, that uses both memory and combinational circuits, is presented in [23].

4. DESIGNING A RECONFIGURABLE PROCESSOR

In the reconfigurable architecture, there is no fixed path between the device units, but the path can be changed depending upon the requirements. MUX are used before the inputs of the device units that act as the switch determining a specific path with respect to some particular select condition. For an example, suppose there are x number of adders, y number of Subtractor and z number of multiplier in the processor. Also we are considering that the chip is accepting k number of inputs. So in general, for all the arithmetic unit having 2 inputs, the MUX in front of the inputs must be having of (x inputs coming from the outputs of adders + y inputs coming from the outputs of subtractors + z inputs coming from the outputs of multipliers + k external inputs). So the MUX must have $[\log_2(x + y + z + k)]$ select lines. In our example (Fig. 1), for simplicity, we have taken x = y = z = 1, k = 2. So we can use 5 x 1 MUXes having 3 select lines before all the arithmetic devices in general. As the output coming from all the units are fed to the inputs of all the unit devices in general, it is possible to have any combination of the arithmetic operations computed by the processor.

In our work, the aim is to design a RNS processor which can be reconfigured dynamically to compute some predetermined functions. For this, the unit operations need to be analyzed and sequenced in terms of the inputs and arithmetic operations. As the arithmetic operations are depicted in terms of the select conditions on the MUX, the inputs as well as the select conditions need to be stored using a LUT. The bit sequences are stored in the LUT blockwise , each block has some particular address. When the address is given for some function, these inputs and select conditions are passed to the input MUXes.

5. IMPLEMENTATION USING FPGA

The proposed scheme is implemented using a VirtexE board (XC5V600E). Verilog codes are generated corresponding to the Reconfigurable RNS Processor and are synthesized and simulated using Xilinx.

Most of the types of equations containing the combination of arithmetic operations like addition, subtraction and multiplication can be implemented using the proposed scheme. Some examples are given here to illustrate the scheme.

Let the Function1 = (X + Y) × Z. When the address of Function1 is given to the LUT, the corresponding block is activated. It will first supply the 2 binary numbers, X and Y, X to the BtoR Conv1 and Y to the BtoR Conv2. Let m = 3, the number of moduli in the moduli set. So BtoR Conv1 and BtoR Conv2 will generate (*x1, x2, x3*) and *(y1, y2, y3)* respectively. Therefore, SM1 = 000 and SM2 = 001 so that *(x1, x2, x3)* and *(y1, y2, y3)* can propagate to the inputs of the adder. So TEMP1 = *(x1 + y1, x2 + y2, x3 + y3*). In the next step Z must be supplied to the BtoR Conv2 and SM5 = 101 and SM6 = 001 so that *(x1 + y1, x2 + y2, x3 + y3*) and *(z1, z2, z3)* can propagate to the inputs of the multiplier. So TEMP2 = *((x1 + y1) × z1, (x2 + y2) × z2, (x3 + y3) × z3*). Now set SM7 = 000 and it is enabled so that the output of the Multiplier passes to the RtoB Conv to have the final output of Function1 = (X + Y) × Z in binary.

Let the Function2 = X ^ Y. X and Y are fed to the BtoR Conv1 and BtoR Conv2 respectively to generate *(x1, x2, x3)* and *(y1, y2, y3)* respectively.

Suppose POWER is the function unit (not shown in the Fig.). POWER actually uses MULTIPLIER internally. When the select inputs of the corresponding MUXes are given, *(x1, x2, x3)* are multiplied by itself *(y1, y2, y3)* times using the same procedure shown before.Two moduli set namely

$$S_{M1} = (2^n, 2^n+1 , 2^n-1)$$
$$S_{M2} = (2^n, 2^n-1 , 2^{n-1}-1)$$

The result can be illustrated graphically in Fig. 5 and 6.From this fig. it is observed that the area,power and delay of the reconfigurable RNS processor varies as the two of moduli set.

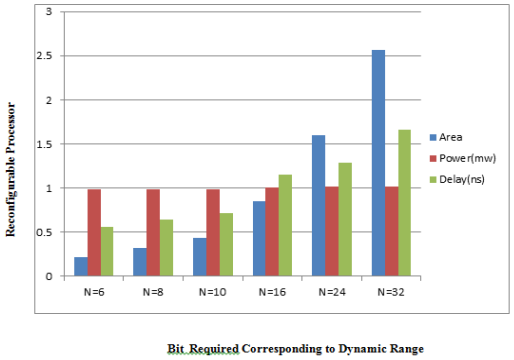


Fig 5:Graphical representation of moduli set *(2^n,2^n-1,2^n+1)*

Table 1:Area,power and Delay of moduli set *(2^n,2^n + 1,2^n - 1)*

S.no	N	Area	Power(mW)	Delay(ns)
1	6	0.2196	0.9873	0.5557
2	8	0.3241	0.9873	0.6373
3	10	0.4389	0.9873	0.7100
4	16	0.8527	1.0104	1.1550
5	24	1.6027	1.0138	1.2845
6	32	2.5618	1.0138	1.6562

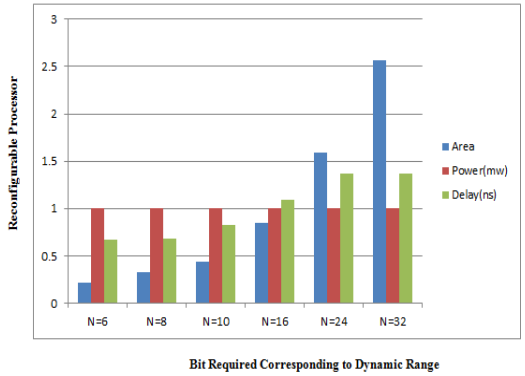


Fig 6:Graphical representation of moduli set *(2^n,2^n-1,2^{n-1}-1)*

Table 2:Area,power and Delay of moduli set *(2^n,2^n - 1,2^{n-} -1)*

S.no	N	Area	Power(mw)	Delay(ns)
1	6	0.2205	1	0.6670
2	8	0.3240	1	0.6785
3	10	0.4414	1	0.8273
4	16	0.8532	1	1.0870
5	24	1.5918	1	1.3700
6	32	2.5688	1	1.3700

6. THE OPERATION PERFORMED BY MODULI SET

6.1 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 [20], is defined by,

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

(6. 1) Assuming that the width of modulo *m* is *n* bits, the structure of the general modular adder is shown in Fig. 7 [20]. It consists of two *n*-bit adders, an *OR* gate and a multiplexer.

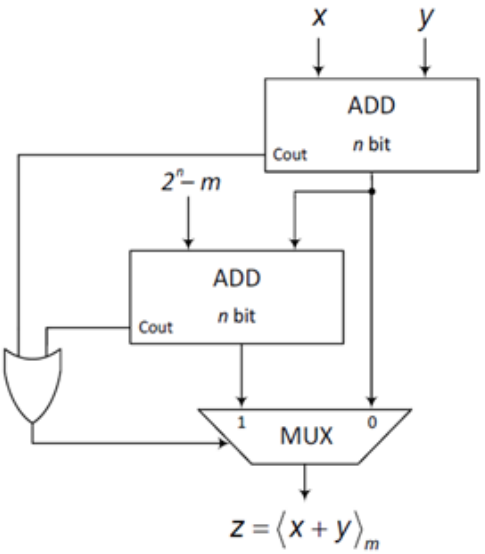


Fig 7: The structure of general modulo adder

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.

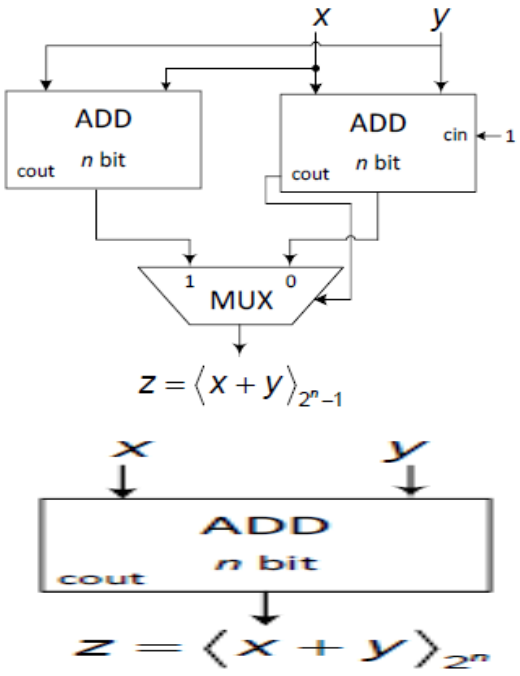
A modulo (2^n) adder can be simply realized using an *n*-bit binary adder, with ignored carry-out. A modulo (2^n – 1) adder, can also be simply realized

using an *n*-bit binary adder with EAC (end around carry) [24].

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 [25], [26]. 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 [27]. A quite interest publication [24] 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 [26], [27] and [28].

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. 8 [6], [24]. These structures have been utilized during the study on different moduli sets.



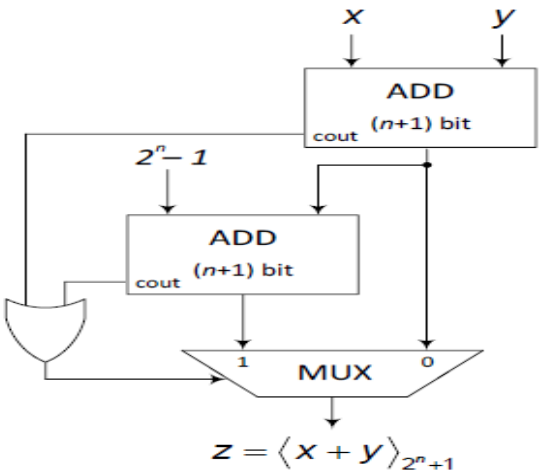


Fig 8: General structures of modular adders based on the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$

6.2 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 [6], [7]. 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,

$$\left\langle \bar{y} \right\rangle_m = \left\langle -y \right\rangle_m = \left\langle m - y \right\rangle_m \Rightarrow \left\langle x - y \right\rangle_m = \left\langle x + \bar{y} \right\rangle_m \quad (6.2)$$

Assuming the width of modulo *m* is *n* bits and according to equation (6.2), 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. 9 [6], [7].

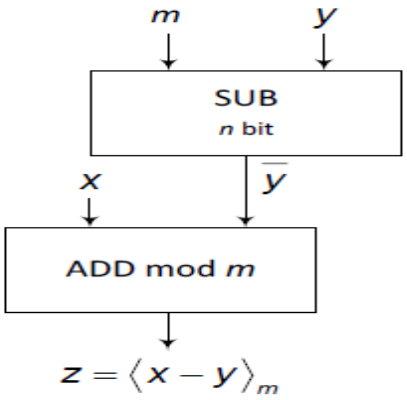


Fig 9: The structure of general modular subtractor

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 [29], [30]. 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.

6.3 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 [31]; 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 [31]. 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 [32], [33]. An example of this method is shown in Fig. 10. The structure of this multiplier is based on the product-partitioning approach presented in [6].

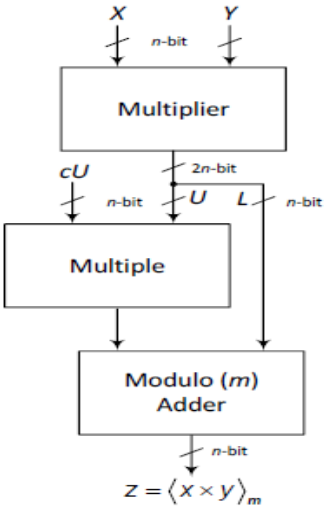


Fig 10: The structure of general modulo multiplier

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 [34]. 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 $(2^n - 1, 2^n + 1)$ [33] . Two RNS multipliers for moduli $(2^n - 1, 2^n + 1)$ based on modified Booth were published in [33], [27]. 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 [28]. This multiplier is an improved design of [27], whose architecture is almost exclusively composed of full and half adders.

7. CONCLUSION

The techniques is for RNS to binary conversion ,binary to RNS conversion ,multiplication ,addition And subtraction can be Implemented easily by using this Proposed project .This principles described can be applied to other of two related moduli sets as well .We have also shown that power efficiency of the proposed scheme is better than all other scheme . In future we will be working on how these parameters bit efficiency ,how complexity and time can be optimized for reconfigurable RNS processor.

REFERENCES

[1] J. P. Hayes, "Computer Architecture and Organization", *Mcgraw -Hill* 2004

[2]V. K. Paliouras, and T. K. Stouraitis, "A low-complexity combinatorial RNS multiplier", *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 48, issue. 7, pp. 675-683, 2001.

[3]W. Wang M.N.S. Swamy, M. O. Ahmad and Y. Wang, "A study of the residue-to-binary converters for the three-moduli sets", *IEEE Transactions on Circuits and Systems I*, vol. 50, issue. 2, pp. 235 – 243, 2003

[4]W. Wang M.N.S. Swamy, and M.O. Ahmad, "An area-time-efficient residue-to-binary converter", *Proceedings of the IEEE Midwest Circuits and Systems Conference, Lansing, MI, USA*, vol. 2, pp. 904- 907, 2000.

[5]Chaitali Biswas Dutta, Partha Garai, Amitabha Sinha, "Design of A Reconfigurable DSP Processor with Bit Efficient Residue Number System", *International Journal of VLSI Design & Communication Systems*, vol. 3, issue 5, pp. 175-189, 2012

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

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

[8] 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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] CAO, B., CHANG, C.H., SRIKANTHAN, T. An Efficient Reverse Converter for the 4-Moduli Set {2*n* – 1, 2*n*, 2*n* + 1, 22*n* + 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.

[15] MOLAHOSSEINI, A.S., NAVI, K., DADKHAH, C., KAVEHEI, O., TIMARCHI, S. *Efficient Reverse Converter Designs for the New 4-Moduli Sets {2n – 1, 2n, 2n + 1, 22n+1 – 1} and {2n – 1, 2n + 1, 22n, 22n + 1} Based on New CRTs*. In IEEE Trans. on Circuits and Systems-I: Regular Papers, 2010, vol. 57, p. 823 – 835. ISSN 1549-8328.

[16] 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.

[17] ZHANG, W., SIY, P. *An Efficient Design of Residue to Binary Converter for Four Moduli Set (2n – 1, 2n + 1, 22n – 2, 22n+1 – 3) Based on New CRT-II*. In Information Sciences Journal, 2008, vol. 178, p. 264 – 279. ISSN 0020-0255.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

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

[34] 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.