Efficient Method for Magnitude Comparison in RNS Based on Two Pairs of Conjugate Moduli

Leonel Sousa
Electrical and Computer Engineering Department
INESC-ID/IST, TULisbon
R. Alves Redol, 9, 1000-029, Lisboa, Portugal
leonel.sousa@inesc-id.pt

Abstract

The non-positional nature of Residue Number Systems (RNS) is very useful to achieve carry free arithmetic. However it makes the comparison of numbers more difficult than in the traditional weighted number systems: there is no any efficient general method for magnitude comparison in RNS. Moreover, magnitude comparison for RNS that rely on pairs of conjugate moduli, which are not relatively prime moduli sets recently proposed because of the large dynamic ranges and the simplicity of the arithmetic units, is a new unsolved problem. In this paper an efficient method and a VLSI architecture is proposed for magnitude comparison in RNS based on sets formed by two pairs of conjugate moduli. This proposed method is much more efficient than the other known ones and is the only one valid for moduli sets not formed by relatively prime integers. The method has been applied to design a very fast Sum-of-Absolute Differences (SAD) unit for motion estimation in video sequences that performs the function entirely within the RNS channels. Experimental results show that this new SAD unit, implemented in the internal memory blocks of the xc2vp50-7 FPGA, is capable of achieving the high throughput required to perform real-time motion estimation in high resolution images.

1 Introduction

RNS have been applied to speedup linear processing, namely to achieve high-speed and low-power VLSI implementations for the multiply-accumulate operation, typically employed in linear signal and image processing [10]. However, the non-positional nature of the RNS prevents its usage to implement the division and, in general, non-linear processing. Magnitude comparison, which is a fundamental operation to support this type of processing is difficult to implement in RNS, being an important topic of research in the last few years [6], [2] and [14].

The traditional techniques for magnitude comparison in RNS use the Chinese Remainder Theorem (CRT) or the Mixed Radix Conversion (MRC) to convert representation of the numbers from the residues to a positional code [10]. However both these techniques are inefficient, because CRT requires modulo M operations (where M is the range of the number system) and MRC is a slow sequential method. A different technique for comparing the magnitude of numbers in residue representation, originally proposed by Akushskii, Burcev and Park, uses the concept of "core function" and applies a descendent and lift scheme to determine the "critical core" values. An improved version of this technique has been proposed several years later, avoiding the iterative procedure by introducing a redundant modulus [6]. Another proposed approach for magnitude number comparison in RNS is based on the "diagonal function", defined as the sum of suitable quotients for estimating its magnitude order [2]. More recently, a new algorithm based on the New Chinese Reminder Theorems [13] has also been proposed to compare the magnitude of numbers in RNS [14]. By applying the new CRT II, it reduces the CRT modulo operation size to $\sqrt{M}$.

In this paper a new algorithm is proposed for comparing the magnitude of numbers in RNS, based on the new CRT III [13]. This new theorem extends the application of RNS to the case where the moduli set is not formed by pairwise relatively prime moduli. Thus it leads to the proposal of a new class of more efficient RNS based on multi-moduli sets that rely on pairs of conjugate moduli [8]. The proposed algorithm takes advantage of both the characteristics of the two pairs of conjugate moduli sets ($\{2^n-1, 2^n+1, 2^{n+1}-1, 2^{n+1}+1\}$) and also of elementary properties relating the parity of integer numbers when added or subtracted.

The proposed algorithm is important for two main reasons. On one hand it is the first algorithm to be proposed for

this important class of moduli sets,. On the other hand, it is the first algorithm leading to VLSI architectures with practical interest for real applications that require comparing the magnitude of numbers, exhibiting even better performance than the ones for weighted number systems. This practical interest is illustrated by implementing an efficient RNS unit for computing the minimum Sum of Absolute Differences (SAD), with application to video motion estimation, which is, by far, the most time consuming and compute intensive task in video coding.

The paper is organized as follows. The adopted class of moduli sets is presented and its characteristics are discussed in the next section. Section 3 proposes a new algorithm for comparing the magnitude in RNS and a VLSI architecture for implementing it. Section 4 applies the proposed VLSI architecture for magnitude comparison to design a video motion estimator. Finally, section 5 concludes this paper.

2 The adopted Class of Moduli Sets

A class of sets (S) composed by pairs of conjugate moduli ($2^{n_i} \pm 1$) has been recently proposed to achieve faster and more efficient RNS processing [8]:

$$\{m_1, m_1^*, \cdots, m_k, m_k^*\} = \{2^{n_1}-1, 2^{n_1}+1, \cdots, 2^{n_k}-1, 2^{n_k}+1\} \ .$$

Although S does not form a set of pairwise relatively prime moduli, for any $k > 1$, it was proved that this limitation can be removed by translating the set S into the set S':

$$S' = \left\{ \frac{m_1}{c_1}, \frac{m_1^*}{c_1^*}, \cdots, \frac{m_k^*}{c_k^*} \right\} \ ,$$

where $\frac{m_1}{c_1}, \frac{m_1^*}{c_1^*}, \cdots, \frac{m_k^*}{c_k^*}$ are relative prime. This class of RNS result in hardware efficient two-level implementations for the weighted-to-RNS and the RNS-to-weighted converters [8, 9], by applying the CRT III [13].

Two pairs of conjugate moduli form a four elements set with a large dynamic range defined by the Least Common Multiple (LCM) of the moduli:

$$M = LCM(2^{n_1}-1, 2^{n_1}+1, 2^{n_2}-1, 2^{n_2}+1) \ .$$

The value of n_2 is usually fixed to $n_2 = n_1 + 1$, in order to achieve a 4-moduli RNS not only completely balanced but also with interesting numerical properties. Hereupon, n_1 will be represented simply by n and the adopted moduli set is represented as $\{2^n - 1, 2^n + 1, 2^{n+1} - 1, 2^{n+1} + 1\}$. The Greatest Common Divisor (GCD) of the product of conjugate pairs can be easily computed by applying the Euclidian algorithm:

$$GCD(2^{2n}-1, 2^{2n+2}-1) = 3 \Rightarrow M = \frac{(2^{2n}-1)(2^{2n+2}-1)}{3} \quad (1)$$

An important property of this moduli sets is that M is an odd number for any integer n. This property, which will be used on this paper, can be easily proved ($\langle x \rangle_y$ denotes the operation $x \bmod y$):

$$\langle M \rangle_2 = \left\langle \frac{\overbrace{(2^{2n}-1)^2}^{odd}}{3} + \overbrace{2^{2n} \times (2^{2n}-1)}^{even} \right\rangle_2 = \langle 1+0 \rangle_2 = 1 \ . \quad (2)$$

By using the following properties, $\left\langle 2^k \right\rangle_{2^k-1} = 1$ and $\left\langle 2^k \right\rangle_{2^k+1} = -1$, for any integer k, the residues X_1 and X_2 of an integer X can be easily computed in the second level (L2) of the binary-to-RNS converter:

$$\begin{aligned} X_1 &= \langle X \rangle_{2^{2n}-1} = \langle X[2n-1:0] + \\ &+ X[4n-1:2n] + X[4n:4n] \rangle_{2^{2n}-1} \quad (3) \\ X_2 &= \langle X \rangle_{2^{2(n+1)}-1} = \langle X[2(n+1)-1:0] + \\ &+ X[4n:2(n+1)] \rangle_{2^{2(n+1)}-1} \ , \quad (4) \end{aligned}$$

where X[k:l] represents the bits k to l of the integer X.

In the first level (L1) of the binary-to-RNS converter, the four residues, x_1, x_1^*, x_2 and x_2^*, are computed by using the following equations:

$$\begin{aligned} x_1 &= \langle X_1[n-1:0] + X_1[2n-1:n] \rangle_{2^n-1} \quad (5) \\ x_1^* &= \langle X_1[n-1:0] - X_1[2n-1:n] \rangle_{2^n+1} \quad (6) \\ x_2 &= \langle X_2[n:0] + X_2[2n:n+1] \rangle_{2^{n+1}-1} \quad (7) \\ x_2^* &= \langle X_2[n:0] - X_2[2n:n+1] \rangle_{2^{n+1}+1} \ . \quad (8) \end{aligned}$$

A single modulo $2^{2n}-1$ (3:2) Carry-Save Adder (CSA) and a modulo $2^{2n}-1$ Carry-Propagate Adder (CPA) can be used to compute (3), and similar units can be used for computing (4). Four CPAs are required for L1, to the 4 moduli, to compute in parallel the residues according to (5) to (8).

The RNS-to-binary conversion can be computed by applying CRT III [13]. For sets with two moduli $\{m_1, m_2\}$ the number X can be computed by using the following equation:

$$X = X_1 + m_1 \left\langle (\frac{m_1}{d})^{-1} \frac{X_2 - X_1}{d} \right\rangle_{m_2/d} \ , \quad (9)$$

where $\left\langle \theta^{-1} \right\rangle_\phi$ denotes the multiplicative inverse of θ modulo ϕ and d represents the $GCD(m_1, m_2)$. The value of m_1 should be greater than m_2 , in order to obtain the modulo operation for the minimum divisor. It can be noticed that for the case that $d = 1$, (9) yields the traditional 2-channel Mixed Radix Conversion (MRC) equation.

In the first level of the converter for the adopted moduli set (d=1) there are two converters operating in parallel:

$$\begin{aligned} X_1 &= x_1^* + (2^n+1) \left\langle (2^n+1)^{-1}(x_1 - x_1^*) \right\rangle_{2^n-1} \\ &= x_1^* + (2^n+1) \left\langle 2^{n-1}(x_1 - x_1^*) \right\rangle_{2^n-1} \quad (10) \\ X_2 &= x_2^* + (2^{n+1}+1) \left\langle (2^{n+1}+1)^{-1}(x_2 - x_2^*) \right\rangle_{2^{n+1}-1} \\ &= x_2^* + (2^{n+1}+1) \left\langle 2^n(x_2 - x_2^*) \right\rangle_{2^{n+1}-1} \ . \quad (11) \end{aligned}$$

In the second level of this converter, $d = 3$ and the X value is computed as:

$$X = X_2 + (2^{2(n+1)} - 1)\left\langle (\frac{2^{2(n+1)}-1}{3})^{-1}\frac{X_1 - X_2}{3}\right\rangle_{\frac{2^{2n}-1}{3}} . \quad (12)$$

3 Proposed method for comparing magnitude in RNS

Two unsigned integer numbers A and B can be compared by subtracting their values ($C = A - B$) and by comparing the results against zero. Depending on the relative values of the operands, C assumes a magnitude that results from one of the following two expressions:

$$C = \begin{cases} A - B, & \text{for } A \geq B \quad (13a) \\ M + A - B, & \text{for } A < B \quad . \quad (13b) \end{cases}$$

The proposed algorithm for number comparison in RNS is based on the parity, which indicates whether an integer number is odd or even.

Axiom 1. *The subtraction of two numbers with the same parity leads to an even number and the subtraction of two numbers with different parities leads to an odd number.*

Based on the mathematical axiom 1, (13a), and (13b), the following prepositions can be stated for a moduli set for which M is an odd number, like the adopted one:

Preposition 1. *$A \geq B$ iff: i) A and B have the same parity and C is an even number* **or** *ii) A and B have different parities but C is an odd number.*

Proof. $A \geq B$ corresponds to the condition (13a). Thus, axiom 1 states that C must be an even number when A and B have the same parity and odd in the other cases. Let us suppose, by assumption, that A can be smaller than B when C is an even number and A and B have the same parity. Then, from (13b), $C = A - B + M$ would become an odd number (M is odd), which contradicts the initial assumption. On the other hand, supposing that A can be smaller than B when C is an odd number and A and B have different parities it is observed from (13a) that C becomes an even number. This contradicts the assumption that C can be odd in this case and proves that $A \geq B$ if and only if preposition 1 is verified. □

Preposition 2. *$A < B$ iff: i) A and B have the same parity and C is an odd number* **or** *ii) A and B have different parities but C is an even number.*

Proof. Preposition 2 can be easily proved by contradiction has it was made for Preposition 1. □

In [5] it has been already presented the idea of performing parity checking for comparing the magnitude in RNS. However, since there is no simple method to identify the parity of a number represented in RNS, it proposes the usage of parity tables with minimum size of $M \times 1$-bit.

The algorithm now proposed to compare the magnitude in RNS is also based on Prepositions 1 and 2, but makes use of properties of the pairs of conjugate moduli sets to efficiently identify the parity of a number directly from the residues. By considering n any integer number, we derive a theorem to identify the parity of a RNS number X ($\langle X \rangle_2$).

Lemma 1. $\left\langle \frac{2^{2n+1}+1}{3} \right\rangle_{\frac{2^{2n}-1}{3}} = \left\langle \frac{2^{2(n+1)}-1}{3} \right\rangle_{\frac{2^{2n}-1}{3}} = 1$.

Proof. $\left\langle \frac{2^{2n+1}+1}{3} \right\rangle_{\frac{2^{2n}-1}{3}} = \left\langle \frac{2\times(2^{2n}-1)}{3} + 1 \right\rangle_{\frac{2^{2n}-1}{3}} = 1$

$\left\langle \frac{2^{2(n+1)}-1}{3} \right\rangle_{\frac{2^{2n}-1}{3}} = \left\langle \frac{4\times(2^{2n}-1)}{3} + 1 \right\rangle_{\frac{2^{2n}-1}{3}} = 1$ □

Lemma 2. $\left\langle (\frac{2^{2(n+1)}-1}{3})^{-1} \right\rangle_{\frac{2^{2n}-1}{3}} = \frac{2^{2n+1}+1}{3}$.

Proof. By applying the two expressions in lemma 1:
$\left\langle \frac{2^{2n+1}+1}{3} \times \frac{2^{2(n+1)}-1}{3} \right\rangle_{\frac{2^{2n}-1}{3}} = \left\langle \frac{2^{2(n+1)}-1}{3} \right\rangle_{\frac{2^{2n}-1}{3}} = 1$ □

Theorem 1. *Given an integer X in the range* $[0, \frac{(2^{2n}-1)\times(2^{2(n+1)}-1)}{3}[$ *and the moduli set* $\{2^n - 1, 2^n + 1, 2^{n+1} - 1, 2^{n+1} + 1\}$*, the parity of the number X can be computed by using the following simple equation:*

$$\langle X \rangle_2 = \left\langle \langle X_2 \rangle_2 \oplus \left\langle \langle X_1 - X_2 \rangle_{2^{2n}-1} \right\rangle_2 \right\rangle_2 , \quad (14)$$

where $\oplus$ denotes the XOR operation and X_1 and X_2 are obtained by computing (10) *and* (11)*, herein presented again for completeness:*

$$X_1 = x_1^* + (2^n + 1) \times \left\langle 2^{n-1}(x_1 - x_1^*) \right\rangle_{2^n-1} \quad (15)$$

$$X_2 = x_2^* + (2^{n+1} + 1) \times \langle 2^n(x_2 - x_2^*) \rangle_{2^{n+1}-1} \quad (16)$$

where x_1, x_1^*, x_2 and x_2^* represent the residues of the channels $2^n - 1$, $2^n + 1$, $2^{n+1} - 1$ and $2^{n+1} + 1$, respectively.

Proof. Applying lemmas 2 and 1 in (12):

$$X = X_2 + (2^{2(n+1)} - 1)\left\langle \frac{2^{2n+1}+1}{3} \times \frac{X_1 - X_2}{3} \right\rangle_{\frac{2^{2n}-1}{3}} \quad (17)$$

$$X = X_2 + (2^{2(n+1)} - 1)\left\langle \frac{X_1 - X_2}{3} \right\rangle_{\frac{2^{2n}-1}{3}} . \quad (18)$$

To derive a simple expression for computing $\frac{X1-X2}{3}$, we start from the definition of a residue:

$$X = k_1 \times (2^{2n}-1) + X_1, \quad (19)$$
$$\text{with} \quad k_1 = \left\lfloor \frac{X}{2^{2n}-1} \right\rfloor < \frac{2^{2\times(n+1)}-1}{3}$$
$$X = k_2 \times (2^{2(n+1)}-1) + X_2, \quad (20)$$
$$\text{with} \quad k_2 = \left\lfloor \frac{X}{2^{2(n+1)}-1} \right\rfloor < \frac{2^{2n}-1}{3} \ .$$

Subtracting (21) from (20):

$$X_1 - X_2 = (2^{2n}-1) \times (4 \times k_2 - k_1) + 3 \times k_2 \ . \quad (21)$$

Since $k_2 < \frac{2^{2n}-1}{3}$:

$$\left\langle \frac{X_1 - X_2}{3} \right\rangle_{\frac{2^{2n}-1}{3}} = \left\langle \frac{(2^{2n}-1)}{3} \times (4 \times k_2 - k_1) + \right.$$
$$\left. + \ k_2 \right\rangle_{\frac{2^{2n}-1}{3}} = k_2 \quad (22)$$
$$\langle X_1 - X_2 \rangle_{2^{2n}-1} = \left\langle (2^{2n}-1) \times (4 \times k_2 - k_1) + \right.$$
$$\left. + \ 3 \times k_2 \right\rangle_{2^{2n}-1} = 3 \times k_2 \ . \quad (23)$$

It can be concluded that $\left\langle \frac{X_1-X_2}{3} \right\rangle_{\frac{2^{2n}-1}{3}}$ has the same parity that $\langle X_1 - X_2 \rangle_{2^{2n}-1}$. Therefore, to simplify the identification of the parity of X, (18) can be written as:

$$X = X_2 + (2^{2(n+1)}-1) \times k_2 \ . \quad (24)$$

Since $X_2 < 2^{2(n+1)} - 1$ ("|" denotes the concatenation operator):

$$\begin{aligned} X &= [X_2 | (2^{2(n+1)} \times k_2)] - k_2 \\ \langle X \rangle_2 &= \langle \langle X_2 \rangle_2 - \langle k_2 \rangle_2 \rangle_2 \\ \langle X \rangle_2 &= \left\langle \langle X_2 \rangle_2 \oplus \left\langle \langle X_1 - X_2 \rangle_{2^{2n}-1} \right\rangle_2 \right\rangle_2 \ , \quad (25) \end{aligned}$$

which ends the proof of theorem 1. □

Corollary 1. *For an integer X in the reduced range* $[0, 2^{2n}-1]$ *and the moduli set* $\{2^n-1, 2^n+1, 2^{n+1}-1, 2^{n+1}+1\}$*, the parity of the number X can be computed by using the the simple expression:*

$$\langle X \rangle_2 = \langle X_2 \rangle_2 \ . \quad (26)$$

Proof. $X_1 = X_2$ when X ∈ [0,$2^{2n}-1$[in (14). For X= $2^{2n}-1$, $\langle X_1 \rangle_{2^{2n}-1} = \langle X_2 \rangle_{2^{2n}-1} = 0$ and (14) is also reduced to (26). □

Based on the previous equations, Algorithm 1 is proposed to efficiently compare the magnitude of two RNS numbers (A, B), represented by the residues (a_1, a_1^*, a_2, a_2^*) and (b_1, b_1^*, b_2, b_2^*), respectively. The parity of numbers A and B is not known. The sequence of the algorithm is expressed by the sequential numbering of the lines, where the computation in each line can be performed in parallel .

Algorithm 1 Comparison of the numbers A, B represented in RNS $(a_1, a_1^*, a_2, a_2^*, b_1, b_1^*, b_2, b_2^*)$.

1: $c_1 = \langle a_1 - b_1 \rangle_{2^n-1}$; $c_1^* = \langle a_1^* - b_1^* \rangle_{2^n+1}$;
   $c_2 = \langle a_2 - b_2 \rangle_{2^{n+1}-1}$; $c_2^* = \langle a_2^* - b_2^* ; \rangle_{2^{n+1}+1}$;
2: (A_1, A_2) = 1st-level-converter(a_1, a_1^*, a_2, a_2^*); {(15) and (16)}
   (B_1, B_2) = 1st-level-converter(b_1, b_1^*, b_2, b_2^*); {(15) and (16)}
   (C_1, C_2) = 1st-level-converter(c_1, c_1^*, c_2, c_2^*); {(15) and (16)}
3: $\overline{P_A} = LSB(\langle A_1 - A_2 \rangle_{2^{2n}-1}) \oplus LSB(A2)$; {'1' if X even}
   $\overline{P_B} = LSB(\langle B_1 - B_2 \rangle_{2^{2n}-1}) \oplus LSB(B2)$;
   $\overline{P_C} = LSB(\langle C_1 - C_2 \rangle_{2^{2n}-1}) \oplus LSB(C2)$;
4: **if** $P_A \oplus P_B \oplus P_C =$ '1' **then**
5: $A \geq B$ is TRUE;
6: **else**
7: $A < B$ is TRUE;
8: **end if**

Table 1. Performance of the algorithms.

Algorithm	Operation size
[6]	$\approx 4 \times M$
[2]	modulo($\approx 2^{3n+2}$)
[14]	modulo(2^{2n-1})
proposed	modulo($2^{n+1}-1$)

In line 1 the residues of C can be computed by subtracting in parallel the residues of A and B. All operations are performed in the respective modulo. In line 2, the residues of A, B and C in the first level are computed by applying (10) and (11). All the residues, for each channel and for the different numbers A, B, C, can be computed in parallel. In line 3, the parity of the three numbers are computed by using theorem 1, or alternatively by applying corollary 1 when A and $B \in [0, 2^{2n}-1]$. The logic value of each variable P_X becomes one when the corresponding number is even. Finally, depending on these parities, prepositions 1 and 2 are used to compare the magnitude of A and B: if the cardinal of numbers with even parity is odd (one ore three), it means that $A \geq B$, otherwise $A < B$.

Table 1 assesses the performance of the proposed algorithm with other known comparison algorithms for RNS sets composed by relatively prime moduli. The "operation size" in this table corresponds to the largest modulo operation or the largest integer operated in the algorithms. For the proposed algorithm, which is specific to the adopted class of moduli sets, this figure is only $(2^{n+1}-1)$, an order of magnitude smaller than the "operation size" of the best amongst the other algorithms. Moreover the total number of operations is drastically reduced to the point where the cost of comparing the magnitude of two numbers represented in RNS is lower than the cost of comparing binary numbers.

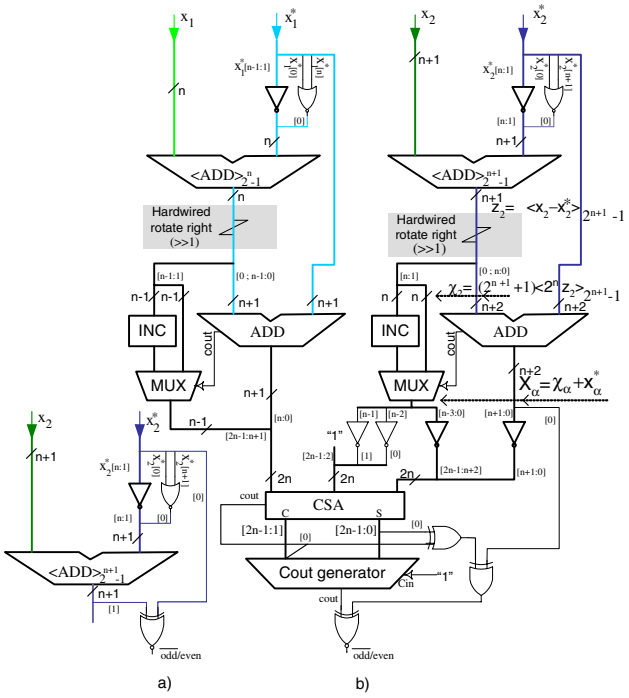

Figure 1. Architecture for parity detection: a) L1: range $[0, 2^{2n} - 1]$; b) L1,2: full range.

The VLSI implementation of the algorithm relies on very simple digital circuits (see fig. 1). The first level of the RNS-to-weighted conversion ((15) and (16)), can be performed in parallel for the two channels. A single 1's complement adder is needed for each computation $\langle (x_1 - x_1^*) \rangle_{2^n-1}$ and $\langle (x_2 - x_2^*) \rangle_{2^{n+1}-1}$. Scaling the result by 2^{n-1} (or 2^n) is performed by rotating it one bit to the right and then multiplying the result by $2^n + 1$ (or $2^{n+1} + 1$) corresponds to concatenating the result with itself. Apparently a $2n$-bit binary adder is required to add x_1^* (and x_2^*) but the hardware can be simplified, as depicted in fig. 1, to a $n + 1$-bit adder and a $n + 1$-bit incrementer. Finally, a simplified 1's complement adder can be used to compute the Less Significant Bit (LSB) of $\langle (X_1 - X_2) \rangle_{2^{2n}-1}$, based on a $2n$-bit CSA and a carry out generator circuit. It is worth noticing that rotation and concatenation operations can be implemented hardwired, thus requiring no further hardware resources. The proposed VLSI architectures presented in fig. 1 were described in VHDL and its overall functionality was thoroughly simulated. Next section applies the proposed architecture to motion estimation in video sequences.

4 Motion estimator in RNS

Block-Matching Motion Estimation (BME) searches for the "best matching block" between the current and a reference frame, according to a search algorithm and a distance metric. The most frequently used employed distance metric is the Sum of Absolute Differences (SAD). The search algorithm can vary from the optimal Full Search (FS) to sub-optimal fast search algorithms [4]. As depicted in Algorithm 2 for blocks with the size $N \times N$, to compute the SAD for a current block x, located at (i, j), and a candidate block $\hat{x}$, with relative coordinates (u, v) in the search area of the reference frame, three different stages can be considered: *i*) absolute difference (ABS) calculation; *ii*) accumulation of absolute differences (SAD variable) and *iii*) determination of SAD_{min}, which is performed only once, out of the two nested loops.

Algorithm 2 BME using SAD distortion measure for the block x with the original coordinates (i,j)

```
1: for k=0 to (N-1) do
2:    for l=0 to (N-1) do
3:       ABS(u,v,k,l) = |x(i+k,j+l) − x̂(u+i+k,v+j+l)|;
4:       SAD(u,v) = SAD(u,v) + ABS(u,v,k,l);
5:    end for
6: end for
7: if SAD(u,v) < SAD_min then
8:    SAD_min = SAD(u,v) ;
9:    Motionvector = (u,v) ;
10: end if
```

Typically video coding standards use $N = 16$ and a 2^p intensity levels with $p = 8$, while the H.264/AVC standard supports variable block size [7]. A block diagram of a typical architecture for computing Algorithm 2 is presented in fig. 2. In the first top stage of this architecture, absolute differences are computed in parallel for 16 pixels of the current block and of a candidate block ($|x - \hat{x}|$). Then these differences are applied to an adder tree and a final adder is required to accumulate the sequences of 16 pixels that compose a block. Finally the SAD_min is determined by comparing the actual value of the `SAD_min` with any new computed value. The `SAD\_min` register is initially set the maximum possible value (`SAD_IV=M-1`).

Optimized and efficient arithmetic structures and units have been proposed to improve each of the stages of the architecture presented in fig. 2. Some examples are an improved architecture for determining the minimum SAD [1] and techniques for enhancing the architectures at the three processing stages [11]. Nevertheless, p-bit CPAs are required and a $(2 \times \log_2 N + p)$-bit CPA (or at least its carry generation part). Moreover, for example, 6 levels of CSAs are required if N=16 differences are computes in parallel and have to summed.

Let us design a SAD architecture for RNS based on the proposed method to directly compare the magnitude of numbers from their residues. Beside the comparison operations, required in the first and last stages of the algorithm for computing the minimum SAD, only modulo $2^n \pm 1$ ad-

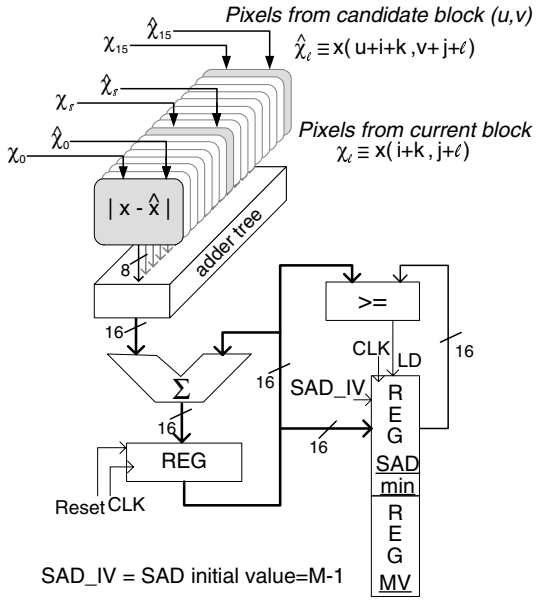

Figure 2. Block diagram of the traditional architecture for SAD calculation and minimum value determination.

ditions and subtractions have to be calculated.

To set up the value of n, the maximum value of the SAD, $N^2 \times 2^p$, has to be within the dynamic range (M) associated to the moduli set ($\{2^n - 1, 2^n + 1, 2^{n+1} - 1, 2^{n+1} + 1\}$). Since $2^{4n} < M$, for $n \geq 2$, the following expression defines the minimum value of n required for computing the SAD:

$$n \geq \lceil \frac{2 \times \log_2 N + p}{4} \rceil \quad . \tag{27}$$

For the typical values N=16 and p=8, n can be equal to 4 leading to the moduli set: $\{15, 17, 31, 33\}$.

Fig. 3 presents a block diagram of the proposed new architecture for motion estimation based on RNS arithmetic. In the first top stage, the values of the pixels are converted to residues and the absolute value of the differences are computed in the RNS domain (see fig. 4). For $p = 2 \times n$, the minimum and maximum input values are 0 and $2^{2n} - 1$, so the binary-to-RNS converter corresponds just to four modulo adders operating in parallel to directly compute (5)-(8). Differences are also calculated in parallel to the multiple residues and both $x_i - \hat{x}_i$ and $\hat{x}_i - x_i$ are computed in order to obtain the absolute value. The absolute value is achieved by directly using the parity of x_i and $\hat{x}_i$ (LSB of input numbers) and by computing the parity of $x_i - \hat{x}_i$. Given that $|x_i - \hat{x}_i| \leq 2^{2n} - 1$, a "parity checker L1" simplified architecture (see fig. 1a) is applied in fig. 4. Four modulo adder trees are used to calculate in parallel the sum of the absolute differences for the four channels. These modulo adder trees can be built by applying modified CSAs trees or by using modulo $2^i - 1$ and modulo $2^i + 1$ CPAs [15]. The final bottom stage in fig. 3 accumulates the computed values to obtain the SAD for a block. Finally, to determine the SAD_min for each block, two "Parity Checker L1,2" units (see fig. 1b) are employed to identify the parity of the computed SAD and the parity of the difference between the computed SAD and the actual SAD_min. The parity of the SAD_min is initially set to '1' (corresponding to SAD_IV=M-1 in fig. 3) and this is updated whenever the value of SAD_min is updated, which occurs whenever the number of parity bits with the value '1' is odd (see Algorithm 1).

It is worth to notice that the largest modulo operation in the proposed architecture is $2^6 + 1$ while for the original architecture in fig. 2 it is necessary to add and to compare 16-bit binary numbers. Since the efficiency of modulo $2^i \pm 1$ adders is similar to the 2^i adders, it can be concluded that the proposed architecture (fig. 3) is much more efficient than the original architecture (fig. 2) to achieve an Application Specific Integrated Circuit (ASIC) for computing the SAD and its minimum value. Both architectures were described using both behavioral and fully structural parameterizable IEEE VHDL and

In the meanwhile, we implemented a SAD unit in a FPGA by exploiting the modular and local processing of the proposed architecture. The arithmetic units were directly mapped on Look-Up-Tables (LUT) by employing the internal memory of actual FPGAs–such implementation is not possible for the binary architectures, because a single 16-bit adder requires 8 GB of memory. The minimum amount of memory required for implementing the proposed architecture directly on LUTs is 274,304 B (approximately 268 kB). This amount of memory can be considerably decreased if some of the tables are replaced by logic circuits: e.g. the computation of the symmetrical to obtain the absolute value takes about 50% of the total amount of memory required. Since the internal memory blocks (BRAMs) in a FPGA are synchronous and have registered outputs, a pipeline structure is automatically established. The SAD unit with the architecture presented in fig 3 was directly implemented in an Xilinx VirtexII Pro (xc2vp50-7) using the ISE (8.2) tools also from Xilinx, without any optimization effort. This device provides 232 dual-port BRAMS, with a capacity of 16 kb each and operating a a maximum clock frequency of 350 MHz, which means a total memory capacity of 464 kB. All the logic circuits were directly implemented through LUTs on the BRAMS, except the $2 \times n$-bit CSA with 3 inputs of the "Parity Checker L1,2" unit (see fig. 1b) that is implemented with logic gates. The dual-ports in BRAMs were exploited to implement twice, in parallel, the same function in a single LUT.

Table 2 presents the implementation results after Place

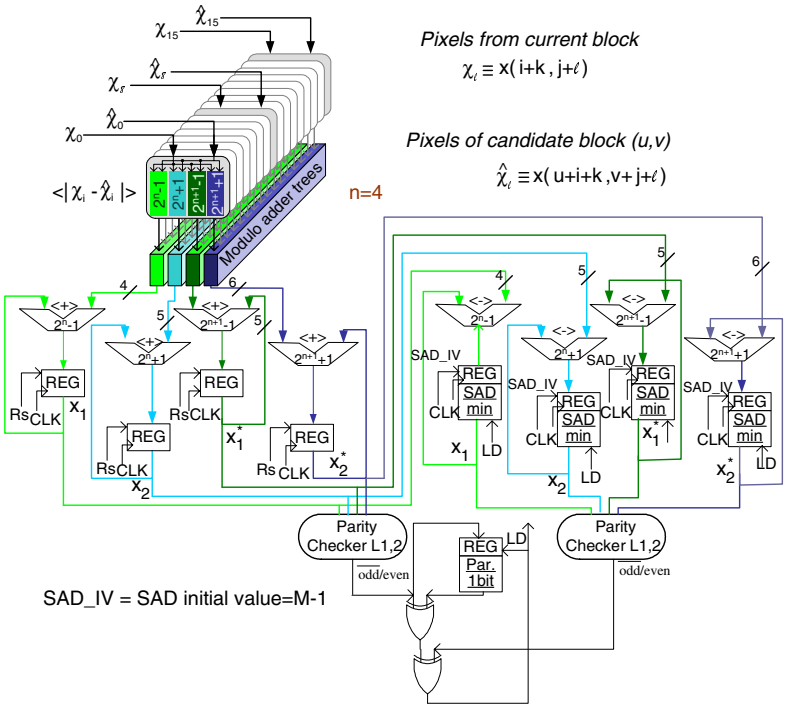

Figure 3. Block Diagram of the proposed architecture for SAD calculation and minimum value determination.

Table 2. Implementation results of the SAD computation and minimum calculation.

Slices (% total)	BRAMs (% total)	Freq. MHz	Latency Cycles	Throughput Blocks/s
246 (1%)	211 (90%)	254	12	1.5×10^7

and Route. The achieved throughput is very high, leading to a processing rate above 1.5×10^7 Blocks/s ($N = 16$) for a maximum operating frequency of around 250 MHz. This means that real time motion estimation is achieved even when the most demanding conditions are considered. A frame rate of about 40 frames/s is achieved when 4CIF high resolution images (704×576) are processed and the exhaustive search procedure is adopted on 16×16 search areas. The output latency is 12 clock cycles, 5 of them are spent to sum the 16 absolute values and to accumulate the result. The optimized SAD architecture for binary representation proposed in [12] led to the fastest known implementation of a SAD unit in a FPGA [3], also a VirtexII Pro FPGA. The maximum clock frequency for this optimized implementation is higher than our implementation, around 300 MHz versus 250 MHz, but the latency is also higher, 17 versus 12 clock cycles. So our implementation, which can be considerably optimized, already allows a performance near the ones obtained with the best SAD units implemented in FPGAs.

Due to the direct mapping of a LUT in a BRAM, the amount of used memory is about 50% higher than the minimum amount of required memory and corresponds to 90% of the total number of FPGA internal memory blocks. As referred above, this amount of memory can be reduced by using the internal memory more efficiently or/and by implementing parts of the circuits with logic gates.

5 Conclusions

This paper proposes a new efficient method for magnitude comparison in RNS based on two pairs of conjugate moduli ($\{2^n - 1, 2^n + 1, 2^{n+1} - 1, 2^{n+1} + 1\}$). This new method relies on characteristics of the moduli set and on elementary properties that relates the parity of integer numbers when they are subtracted. Not only this method is the first to compare the magnitude of numbers in RNS based on sets of non relatively prime moduli, but it is also the first leading to VLSI architectures with practical interest for comparing the magnitude of numbers in residue representation. These RNS proposed architectures are even more efficient than the weighted number systems counterpart.

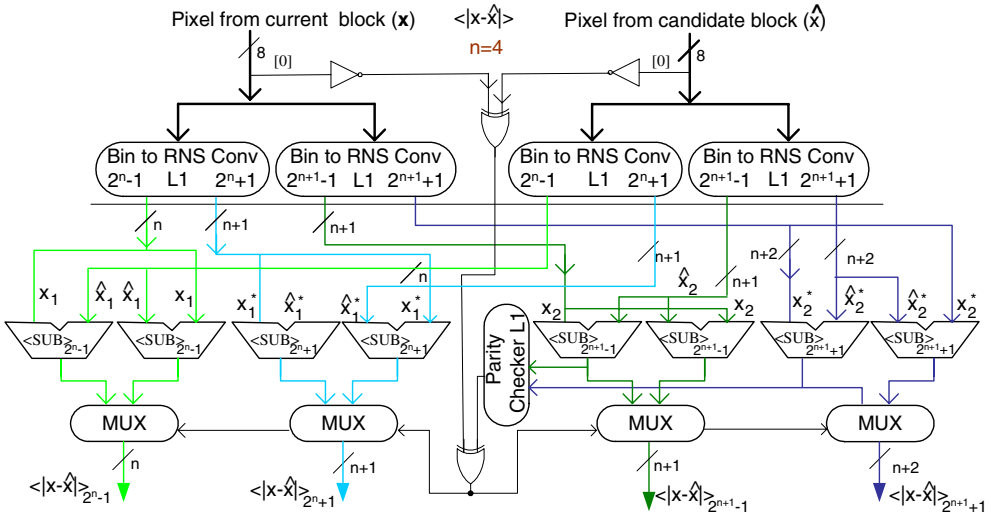

Figure 4. Architecture proposed for computing the absolute difference in RNS.

The practical interest of the proposed comparison method was illustrated by applying it for computing the minimum SAD, which is by far the most time consuming and compute intensive task in video coding. A very efficient minimum SAD computation unit is obtained in RNS based on two pairs of conjugate moduli just by implementing the RNS arithmetic units through look-up-tables. Experimental results show that, by directly mapping these look-up-tables into on-chip memory blocks of a Xilinx xc2vp50-7 FPGA, a throughout of more than 1.5×10^7 Blocks/s is achieved (16×16 pixels per block). This throughput allows to code in real time high resolution 4CIF video sequence with the optimal full search block matching motion estimation and 16×16 search areas.

References

[1] T. Dias, N. Roma, and L. Sousa. Low power distance measurement unit for real-time hardware motion estimators. In *Int. workshop on Power and Timing Modeling, Optimization and Simulation - PATMOS'2006*, volume 4148 of *LNCS*, pages 247–255. Springer Berlin, September 2006.

[2] G. Dimauro, S. Impedovo, and G. Pirlo. A new technique for fast number comparison in residue number system. *IEEE Trans. on Comp.*, pages 608–612, 1993.

[3] G. Kuzmanov, G. N. Gaydadjiev, and S. Vassiliadis. The Virtex II Pro MOLEN processor. In *Proc. of the International Workshop on Computer Systems: Architectures, Modelling, and Simulation (SAMOS)*, pages 192–202, July 2004.

[4] Y. Liu and S. Oraintara. Complexity comparison of fast block-matching motion estimation algorithms. In *Proc. of IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, volume III, pages 341–344, 2004.

[5] M. Lu. *Arithmetic and Logic in Computer Systems*. John Wiley & Sons, 2004.

[6] D. D. Miller, R. E. Altschul, J. R. King, and J. N. Polky. Analysis of the residue class core function of Akushskii, Burcev and Pak. In *Residue number System Arithmetic: Modern Applications in Digital Signal Processing*, pages 390–401. IEEE Press, New York, 1986.

[7] I. Richardson. *H.264 and MPEG-4 Video Compression: Video Coding for Next Generation Multimedia*. John Wiley, 2004.

[8] A. Skavantzos and M. Abdallah. Implementation issues of the two-level residue number system with pairs of conjugate moduli. *IEEE Trans. on Signal Processing*, 47(3):826–838, 1999.

[9] A. Skavantzos and Y. Wang. Application of new chinese remainder theorems to RNS with two pairs of conjugate moduli. In *Proc. of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, pages 165–168, 1999.

[10] N. S. Szabo and R. I. Tanaka. *Residue Arithmetic and its Applications to Computer Technology*. McGraw-Hill, 1967.

[11] J. Vanne, E. Aho, T. Hamalainen, and K. Kuusilinna. A high-performance sum of absolute difference implementation for motion estimation. *IEEE Trans. on Circuits and Systems for Video Technology*, 16(7):876–883, July 2006.

[12] S. Vassiliadis, E. Hakkennes, S. Wong, and G. Pechanek. The sum-absolute-difference motion estimation accelerator. In *Proc. of the Euromicro Symposium on Digital Systems Design*, pages 559–566, 1998.

[13] Y. Wang. New chinese remainder theorems. In *Proc. of Thirty-Second Asilomar Conference on Signals, Systems & Computers,*, volume 1, pages 165–171, 1998.

[14] Y. Wang, X. Song, and M. Aboulhamid. A new algorithm for RNS magnitude comparison based on new chinese remainder theorem II. In *Proc. of the Ninth Great Lakes Symposium on VLSI p. 362*, pages 362–365, 1999.

[15] R. Zimmermann. Efficient VLSI implementation of modulo $(2^n \pm 1)$ additi on and multiplication. In *Proc. 14th IEEE Symposium on Computer Arithmetic*, Australia, April 1999.