An improved fast mode decision algorithm for VLSI architecture implementation

J. CHARLES RAJESH KUMAR[a,*], T. VANCHINATHAN[b] P. SUDHARSAN[c]
[a]*Lecturer, Department of Electrical & Computer Engineering, Effat University, Kingdom of Saudi Arabia*
[b]*Director – AXIIP Semiconductor (P) Ltd.*
[c]*Director – AXIIP Semiconductor (P) Ltd.*

For better video quality in the H.264/AVC video coding technology, motion estimation has massive growth due to improvements in searching algorithms and improved significantly in compression efficiency and complexity, specifically in area, power and throughput. In this paper, an efficient sum of absolute difference (SAD) tree and its hardware architecture have proposed in Residue Number System (RNS) based moduli and implements the full search variable block size motion estimation (FSVBSME). The main advantage is that for performing carry free addition operation, residue number system is being considered as a non weighted number system to binary number system, RNS is mostly suitable for image compression techniques and loss of image quality is very less. In hardware implementation, it occupies less area and takes less execution time for output result. This proposed architecture is capable of achieving the less hardware cost and logical elements, high throughput required to perform real time motion estimation. Experimental results show that synthesized with TSMC 180nm CMOS, the proposed design occupies 12.9k logic gates at 352MHZ and consumes 19mW power to encode 1920X1088 HDTV video frames at 30 frames per second.

(Received April 9, 2015; accepted May 7, 2015)

Keywords: SAD RNS adder, SAD comparator, Mode decision

1. Introduction

Motion Estimation (ME) is a vital part of most motion-compensated video coding standards [1]. It is a process for estimating motion vectors (MV) that transform from reference frame to the current frame in a video sequence coding. FSVBSME is a temporal redundancy elimination technique between two or more consecutive frames for video compression. H.264/AVC is the standard video coding developed by the ITU-T. ME [2-3] is mostly based on a block-matching [4-8] technique is playing a major role in H.264/AVC by using the temporal redundancy between consecutive successive frames. In H.264, a video frame split by using macro blocks (MB) of 16x16 size in a FSVBSME approach. So, FSVBSME architecture for the H.264/AVC have been proposed [9-10]. In arithmetic systems, the speed is limited by making the logic decisions and the extent to which the low order numeric significance decisions can affect higher significance results. This issue is described by the addition operation, by which a low-order carry can have a rippling effect on a sum. RNS have been applied to achieve high-speed and low-power VLSI implementations, typically utilized in signal and image processing. To convert representation of the numbers from the residues to a positional, The conventional magnitude comparison systems in RNS [11] utilize the Chinese Remainder Theorem (CRT) and the Mixed Radix Conversion (MRC). However, both these methods are inefficient, the main reason is that the CRT requires modulo M (number system range) operations. MRC is a slow sequential technique. Recently, a New Chinese Reminder Theorem [12] was proposed to analyze the magnitude of the number in RNS. In this paper, the proposed algorithm takes advantage of the characteristic of the conjugate moduli set $(2^n - 2^k - 1)$ offers better performance of delay, area and speed. The new modulo adder could be isolated into four units, such as, 1. Preprocessing unit, 2. Prefix computation unit, 3. Carries correction unit, and 4. Sum computation unit. In the proposed scheme, To obtain the final carries required in the sum computation module, the carry information of A+B+T could be calculated by prefix computation unit. So that the proposed modulo $(2^n - 2^k - 1)$ adder can get the best delay performance. The proposed algorithm has two main reasons. It is the best algorithm leading to VLSI architecture with the real time applications for better performance in weighted number systems. And it is described by implementing an efficient RNS for computing the minimum Sum of Absolute Differences (SAD), with more time consuming video motion estimation application.

The paper is classified as follows. Section II discussed about SAD adder tree. Section III discussed about Residue Number System and its modular addition procedure and RNS modulo $(2^n - 2^k - 1)$ adder. Section IV describes the motion estimation using RNS. Finally, section V concludes this paper.

2. SAD adder tree

In the 16 SAD architecture, each one is in charge of the SAD computation of one primitive 4x4 sub-block in

parallel. There are 16 absolute differences computed and then the 16 absolute values are fed into the adder unit to complete a 4x4SAD. Adding the 16 absolute differences to obtain one 4x4SAD is implemented by employing multilevel 3-2 compressors, as shown in Fig. 1 (a). Fig. 1 (b) shows the structure of the 3-2 compressor, where the k binary inputs of a,b and c bit values a_0-a_k, b0-b_k, and c_0-c_k respectively. And the depth of input value is 8 bit, so k equals to 8. The output sum_0-sum_k stand for each of the summer bit of the input three binary bits, and $carry_0$-$carry_k$ stands for each of the carry bit.

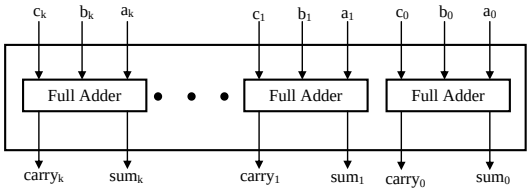

Fig. 1. (a) Structure of 3-2 compressor.

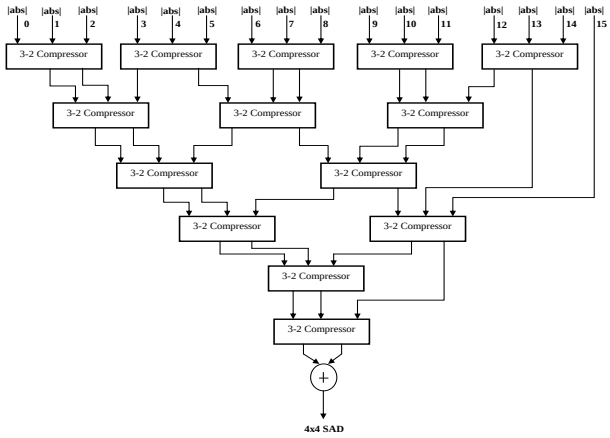

Fig. 1. (b) Structure of adder unit.

One 16×16 MB is partitioned into 16 4x4 sub-block, denoted as C0-C15,as shown in Fig. 2 (a). During the processing procedure, eight 8×4 SADs and 4x8 SADs can be first obtained simultaneously, and then four 8×8 SADs can be produced at the same time, then two 16x8 SADs and 8x16 SADs be synchronously achieved subsequently, and finally the 16×16 SAD can be obtained, shown in Fig. 2 (b). All of the 41 SADs should be stored in the registers for the reuse of the following unit. The generation of the whole 41 SADs of one MB can be implemented within 6 cycles.

C0	C1	C2	C3
C4	C5	C6	C7
C8	C9	C10	C11
C12	C13	C14	C15

Fig. 2. (a) Block Pattern of H.264.

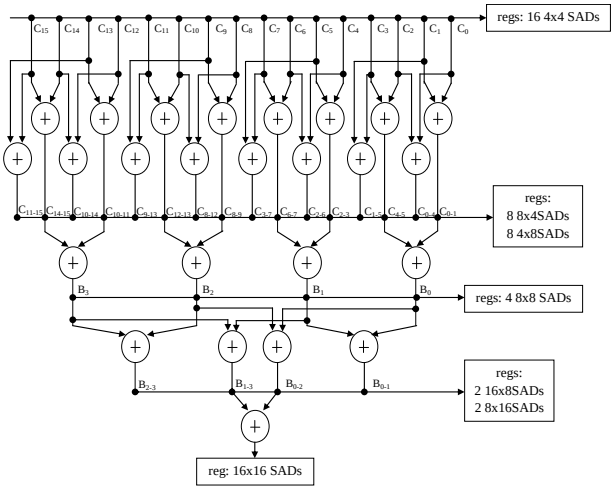

Fig. 2. (b) Structure of Adder Array.

Mostly, the processing systems concerned about the speed of arithmetic. In arithmetic systems, the speed is limited by the way of building block that makes logic decisions and the extent to which decisions of least numeric significance can affect results of most significance. This problem is best designed by the addition operation, in which a lower-order carry can have a rippling effect on a sum. The adder tree array is a carry effective. In the adder tree array, the addition operation effect the carries on digits of most significance. For every addition operation, it facilitates the realization of less speed and more power. The most significance sum value waits for the carry values for execution result. Hence, the adder tree consumes more power and less speed.

3. Residue number system and its modular addition procedure

The advantage of the RNS adder tree is that the absence of carry propagation between its arithmetic units, and for every addition operation, it is no need wait for carry values. Hence, It facilitates the realization of more speed, less power arithmetic. The Residue Number System is defined as co-prime modular radix groups $\{m_1,\ m_2,\ \dots,\ m_N\}$, where N is greater than 1, $GCD(m_i, m_j) = 1, i \neq j, i, j = 1,2,\dots,N$, and $GCD(m_i, m_j)$ is the greatest common divisor of $m_i and\ m_j$. By residues respect to the modulus $m_i$of the integer X in [0,M) can be represented as $(x_1,\ x_2,\ \dots,\ x_N)$, where $x_i = \langle X \rangle_{m_i}$, $M = \prod_{i=1}^{N} m_i, i = 1,2,\dots,N$. In the range of [0,M), the integers A,B, and C can be represented RNS numbers as $(a_1,\ a_2,\ \dots,\ a_N), (b_1,\ b_2,\ \dots,\ b_N)$ and $(c_1,\ c_2,\ \dots,\ c_N)$ respectively. According to Guassian modular algorithms, $C_i = (a_i\ \Delta\ b_i)_{m_i}$, where Δ represents addition operation, subtraction operation and multiplication operation.

For the range of [0,M) integers A and B, modulo m addition is defined as

$$C = \langle A+B\rangle_m = \begin{cases} A+B & A+B < m \\ A+B-m & A+B \geq m \end{cases} \quad (1)$$

If $C = \langle A+B\rangle_m$ and the modular adder bit width is n-bit, where $n = \lceil \log_2 m \rceil$. So that,

$$C = \begin{cases} A+B & A+B\ +T < 2^n \\ \langle A+B+T\rangle_{2^n} & A+B+T \geq 2^n \end{cases} \quad (2)$$

Here the correction $T = 2^n - m$. The basic rule in most modular adder design is that if A+B+T carry bit is 1, the result of modular addition is n LSBs of A+B+T, otherwise, the result is A+B. Then, Parallel prefix addition operation is extensively accepted in binary adder design. Present sum and carry bits can be calculated with the previous carries and inputs. The prefix computation is calculated as

$$(g_i, p_i) = (a_i b_i, a_i \oplus b_i) \quad (3)$$

Where carry generation bit g_i and carry propagation bit p_i. In the sum computational unit, the carries c_i from the prefix computation unit and the partial sum of the pre-processing unit are utilized to calculate the final sum bits s_i,

$$s_i = p_i \oplus c_i \ \ i = 0,1,\dots,n-1. \quad (4)$$

RNS MODULO $2^n - 2^k - 1$ ADDER

The modulo $(2^n - 2^k - 1)$adder is divided into four different modules, is shown in Fig. 3.

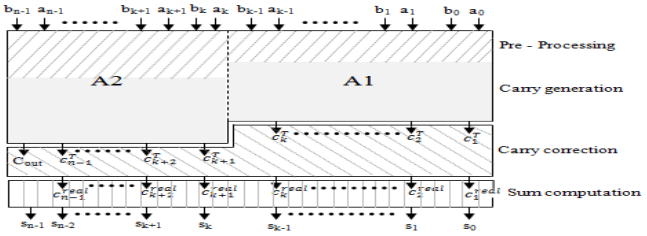

Fig. 3. The proposed modulo $2^n - 2^k - 1$ adder structure.

1. Pre-processing Unit: The pre-processing unit is used to generate the carry generation g_i and carry propagation p_i bits of A+B+T.

$$T = 2^{\lceil \log_2 2^n - 2^k - 1 \rceil} - m = 2^k + 1 \quad (5)$$

Actually, A1 is utilized for lower k-bits and A2 is utilized for higher n-k bits addition,and it can be performed in the computation of A+B+T. Here, the binary representation of A, B and T are $(a_{n-1} \dots a_k\ a_{k-1}\ \dots a_1 a_0)$, $(b_{n-1} \dots b_k\ b_{k-1}\ \dots b_1 b_0)$ and "$\underbrace{000\dots001}_{(n-k)bit}\ \underbrace{00\dots001}_{k\ bit}$" respectively. The operation of the adder A1 and A2 can be represented as

$$\begin{cases} S_{A_1} = a_{k-1} \dots a_0 + b_{k-1} \dots b_0 + T_{A_1} \\ S_{A_2} = a_{k-1} \dots a_0 + b_{k-1} \dots b_0 + T_{A_2} + C_{A_1} \end{cases} \quad (6)$$

Where $T_{A_1} = \underbrace{00\dots001}_{k\ bit}$, $T_{A_2} = \underbrace{000\dots001}_{(n-k)bit}$ and C_{A_1} is the carry out bit of A1 adder.

In Fig. 1, A1 can be declared as *k*-bit adder in the lowest carry-in bit for zero e every bit value except the LSB value. the carry generation and carry propagation bits are

$$\begin{cases} (g_0, p_0) = \left(a_0 + b_0, \overline{a_0 \oplus b_0}\right) & i = 0 \\ (g_i, p_i) = (a_i b_i, a_i \oplus b_i) & i = 1,2,\dots,k-1 \end{cases} \quad (7)$$

The three inputs of adder A_2 are $a_{n-1} \dots a_k, b_{n-1} \dots b_k$and T_{A_2}in binary. The adder A2 depends on both the constant T_{A_2} and the A_1carry out bit of C_{A_1}. For adder A_2, Simple carry save adder (SCSA) reduces the number of inputs form three to two. When $i = k, k+1, \dots, n-1$, for a_i and b_i; the second stage of g_i and p_i in SCSA and the final outputs of pre-processing unit is defined as

$$(g_i', p_i') = (a_i b_i, a_i \oplus b_i) \quad (8)$$

$$\begin{cases} (g_k, p_k) = \left(p_k', \overline{p_k'}\right) & i = k \\ (g_i, p_i) = (p_i' g_{i-1}', p_i' \oplus g_{i-1}') & i = k+1, \dots, n-1 \end{cases} \quad (9)$$

From Eq. 6 and 8, the computational prefix is obtained. And carry out of SCSA is defined from the A+B+T carry-out bit.

$$C_{SCSA} = a_{n-1} b_{n-1} = g_{n-1}' \quad (10)$$

4. Carry generation unit

The pre-processing unit is with the carry-generation and carry-propagation bits, the carries $c_i^T\ (i = 1, 2, \dots, n)$ of A+B+T can be obtained. To determine the carry out bit of A+B+T, C_{SCSA} is combined with the prefix tree carry-out bit.

$$\begin{aligned} C_{out} &= C_{SCSA} + c_n^T = C_{SCSA} + G_{n-1:0} \\ &= C_{SCSA} + G_{n-1:l} + P_{n-1:l} G_{l-1:0} \\ &= C_{SCSA} + G_{n-1:l} + P_{n-1:l}\, c_l^T \end{aligned} \quad (11)$$

Where $0 < l \leq n-1$.

5. Carry correction unit

The real carries c_i^{real} of the carry correction unit is used for each bit in the final sum computation. For modulo

$2^n - 2^k - 1$ adder, T is $2^k + 1$ represented as $\underbrace{000\ldots001}_{(n-k)bit}\ \underbrace{00\ldots001}_{k\ bit}$ in binary. The A+B+T computation, $S_A = A + B + \underbrace{000\ldots001}_{(n)bit}$ and $S_B = S_A + \underbrace{000\ldots001}_{(n-k)bit}\ \underbrace{00\ldots001}_{k\ bit}$.

Carry correction of A_1

The binary representation of T is $\underbrace{000\ldots001}_{(n-k)bit}\ \underbrace{00\ldots001}_{k\ bit}$, c_i^T can be regarded as the carry bits of (A+B+T-1) + c_{in} and c_{in} is 1. The first correction output $c_{i+1}^{c_1}$ result is

$$c_{i+1}^{c_1} = \overline{c_{out}}\, c_{i+1}^{T-1} + c_{out} c_{i+1}^{T} = \overline{c_{out}}\overline{P_{i:0}} c_{i+1}^{T} + c_{out} c_{i+1}^{T} = c_{i+1}^{T}\left(c_{out} + \overline{P_{i:0}}\right) \quad (12)$$

Carry correction of A_2

If c_{out} is zero, c_i^{real} is equal to the carry of A + B + T − 1 − 2^k, else the c_i^{real} is equal to A + B + T carry. i.e, $c_i^{real} = c_i^{c_1}$. The inputs of adder A_2 are $p'_{n-1} \ldots p'_{k+1} p'_k$ and $g'_{n-2} \ldots g'_{k+1} g'_k 1$. the carry-in bit $c_k^{c_1}$ is the A_1 carry-out bit. The two inputs additions of A_2 are $p'_{n-1} \ldots p'_{k+1} p'_k$ and $g'_{n-2} \ldots g'_{k+1} g'_k c_k^{c_1}$ with the carry-in bit value 1.

$$\begin{cases} p'_{n-1} \ldots p'_{k+1} p'_k + g'_{n-2} \ldots g'_{k+1} g'_k 1 + \underbrace{00\ldots c_k^{c_1}}_{(n-k)\ bit} & a) \\ p'_{n-1} \ldots p'_{k+1} p'_k + g'_{n-2} \ldots g'_{k+1} g'_k c_k^{c_1} + \underbrace{00\ldots1}_{(n-k)\ bit} & (b) \end{cases} \quad (13)$$

6. Sum computation unit:

The partial sum bits of A + B and A + B + T for final sum computation are defined by the correction carry. If c_{out} is zero, the correction carry c_i^{real} is the A + B carry bit. Else, A + B + T carry bit.

$$\begin{cases} p_0^0 = \overline{p_0},\ p_0^1 = p_0 & i = 0 \\ p_k^0 = \overline{p_k},\ p_k^1 = p_k & i = k \\ p_i^0 = p_i^1 = p_i & i = 1, \ldots, k-1, k+1, \ldots, n-1 \end{cases} \quad (14)$$

Where p_i^0 and p_i^1 are the partial sum bits of A + B and A + B + T respectively, (i = 0, 1, … , n − 1). Hence

$$s_0 = \overline{c_{out}}\, p_0^0 + c_{out}\, p_0^1 = \overline{c_{out}\, p_0} + c_{out}\, p_0 = c_{out} \oplus \overline{p_0} \quad (15)$$

$$s_k = c_k^{real} \oplus (\overline{c_{out}}\, p_k^0 + c_{out}\, p_k^1) = c_k^{real} \oplus (\overline{c_{out}\, p_k} + c_{out}\, p_k) = c_k^{real} \oplus c_{out} \oplus \overline{p_k} \quad (16)$$

When $i = 1, \ldots, k-1, k+1, \ldots, n-1$

$$s_i = c_i^{real} \oplus p_i \quad (17)$$

The final sum bits are

$$s_i = \begin{cases} c_{out} \oplus \overline{p_k} & \text{i} = 0 \\ c_k^{real} \oplus c_{out} \oplus \overline{p_k} & i = k \\ c_i^{real} \oplus p_i & i = 1, \ldots, k-1, k+1, \ldots, n-1 \end{cases} \quad (18)$$

Here $c_i^{real}, c_{out} \oplus \overline{p_k}$ can be obtained at the same time. Hence, no extra delay can be occurred.

RNS architecture

The VLSI implementation of modulo $2^n - 2^k - 1$ adder architecture is shown in Fig. 5. And describing the shapes of the modulo $2^n - 2^k - 1$ adder in Fig. 4.

The "white circle" pattern in Fig. 4 is the pre-processing unit and it is utilized to generate carry generation and carry propagation bits for prefix calculation. And the fixed input "1" at the 1st and the 4th positions, the " triangle " and " white square box" patterns are utilized for this special condition. These patterns computations considered by eq (7), (8) and (9).

The prefix computation unit is defined in the "black circle" pattern. This pattern consists of one OR gate and one AND gate. It is used for carry generation path. However, To compute the propagation bits, the "gray circle" pattern of prefix tree final stage is not required.

The "triangle black circle" pattern is computing the c_{out} in Fig. 4. From eq (10), c_{out} can get after an OR gate.The "pentagon box", "gray square box" and "square cross box"patterns used for the correction of carry unit.

The "cross circle" and "cross nibble circle" patterns are performing the final sum computation. The cross circle pattern performs the XOR operation and "cross nibble circle" pattern performs the XOR operation with one of it's input is inverted. Because, in eq (17) $c_{out} \oplus \overline{p_k}$ computation can be performed.

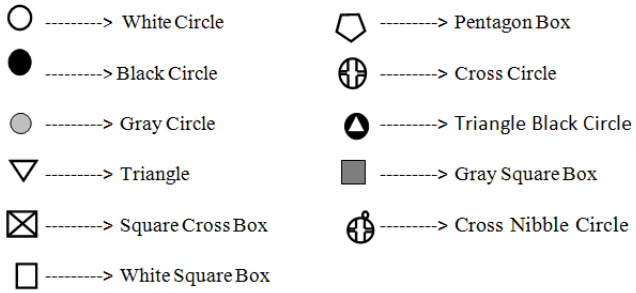

Fig. 4. Describing the shapes of modulo $2^n - 2^k - 1$.

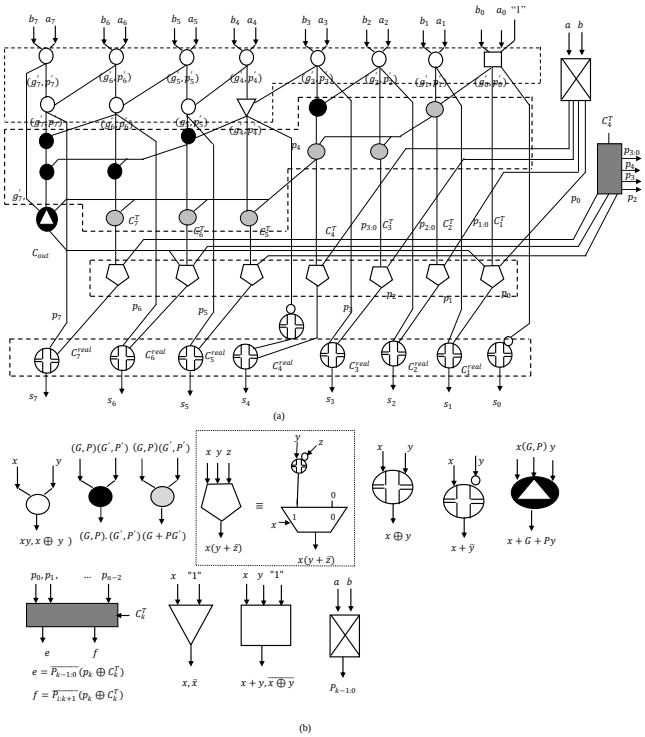

Fig. 4. Implementation architecture of Modulo $2^n - 2^k - 1$ *adder (n=8, k=4).*

7. Motion esimation using RNS

FSVBSME (BME) searches the best matching block between the current frame and a reference frame. The most frequently utilized technique to find the distance is SAD. The search algorithm can be varied from optimal FS to sub-optimal fast search algorithms.

In H.264/AVC, frame of a video is split into macro blocks of 16x16 size. Each macro block is segmented into different sub-blocks, shown in Fig. 5. Motion estimation is conceded 7 different modules, mode 1 has a 16x16 macro block, mode 2 has two 16x8 sub-blocks, mode 3 has two 8x16 sub-blocks, and mode 4 has four 8x8 sub-blocks. Then, each block of 8x8 size is split into sub-blocks. Mode 5 has two 8x4 sub-blocks, mode 6 has two 4x8 sub-blocks, and mode 7 has four 4x4 sub-blocks. The total possible partitions are 41. In FSVBSME, to calculate the minimum SAD (SADmin) of 4x4 block and achieve all the SADmin from all these 41 modes slicing.

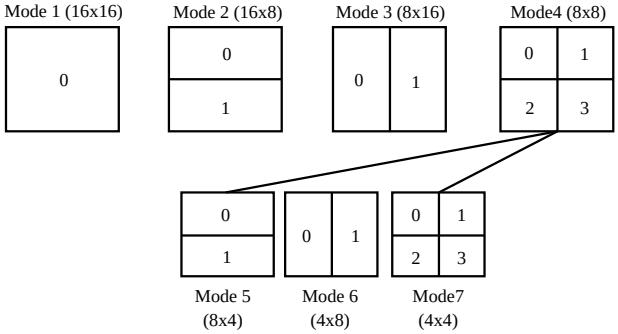

Fig. 5. The different sub-block of partitions and its positions.

In H.264/AVC video coding, motion estimation is mainly concentrating the temporal redundancy between successive frames. For the H.264 FSVBSME implementation, the proposed architecture has the advantages of low latency and high throughput. Fig. 6 shows the block diagram of the proposed architecture, which contains the external memory, memory controller, current and reference frames, RNS adder tree, RNS SAD array, SAD comparator and mode decision.

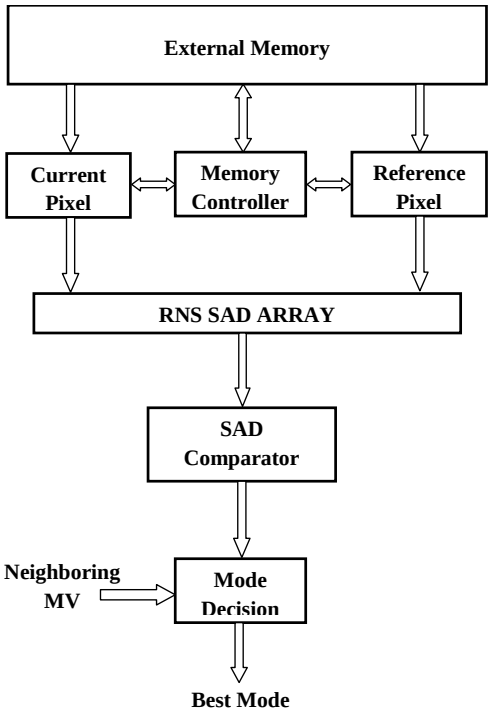

Fig. 6. The proposed architecture of motion estimation using RNS.

From Fig. 6, first of all, the memory controller reads the current pixel and reference pixel from the external memory by a system bus, and send to the Current pixel and reference pixel values to be stored; Secondly, the current pixel and reference pixel values are responsible to send the data to RNS adder arrays to calculate SAD and

array of RNS adder, shown in Fig. 7. When all the searching points are looped over, the cost value (SAD) of the final 41 sub-blocks can be achieved. Then, send the costs of these 41 parallel sub-blocks to Comparator SAD Tree Array to calculate the module information, such as the optimal position cost and the motion vector. Finally, the mode decision decides the best motion vector, by finding the position of the present cost values and previous neighboring motion vector position.

RNS SAD Array

To measure the similarity between the two image window blocks, the sum of absolute differences (SAD) algorithm is being evaluated for image comparison and object recognition in digital image processing. It works by taking the absolute difference between each pixel in the current frame and corresponding reference frame in the image window block. It is widely utilized for stereo vision, the generation of disparity maps for stereo images, optical flow, motion estimation for video compression. In Fig. 7, the SAD is computed the 16 differences from the current and reference pixels in the 4x4 window size image.

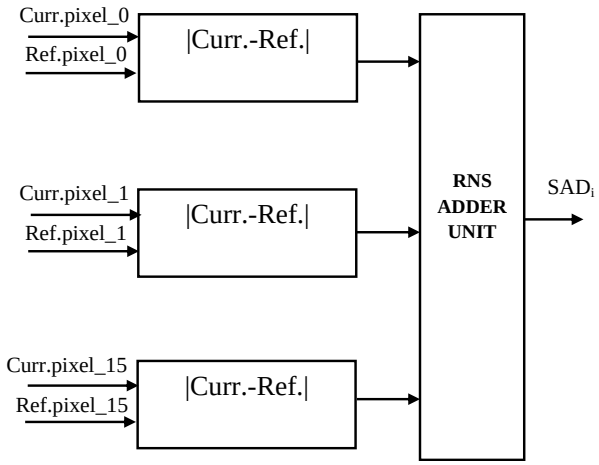

Fig. 7. Sum of Absolute Difference (SAD) Unit.

Each macro block is split into seven sub-blocks. First, the block will be calculated the 4x4 window size block. In the 4x4 window. In Fig.8, the RNS adder tree is done the carry free addition operation in the parallel process. So, the execution of the RNS adder tree process speed is increased. One 16x16 MB is partitioned into 16 4x4 sub-block, denoted as C_0-C_{15},as shown in Fig. 9. During the processing procedure, eight 8x4 SADs and 4x8 SADs can be first obtained simultaneously, and then four 8x8 SADs can be produced at the same time, then two 16x8 SADs and 8x16 SADs be synchronously achieved subsequently, and finally the 16x16 SAD can be obtained. All of the 41 SADs should be stored in the registers for the reuse of the following unit. These 41 SADs of one MB can be implemented in 4 cycles.

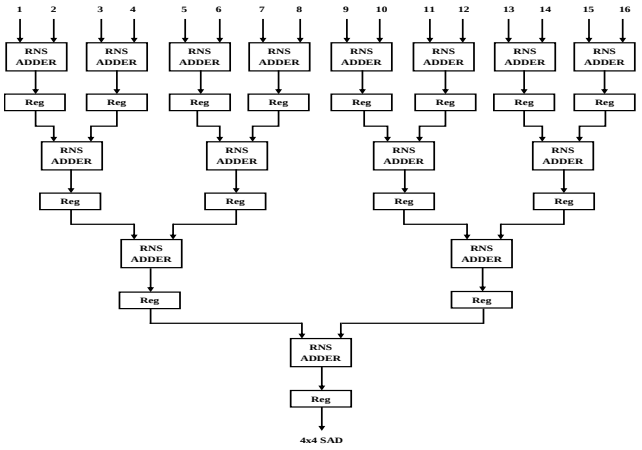

Fig. 8. RNS Adder Tree.

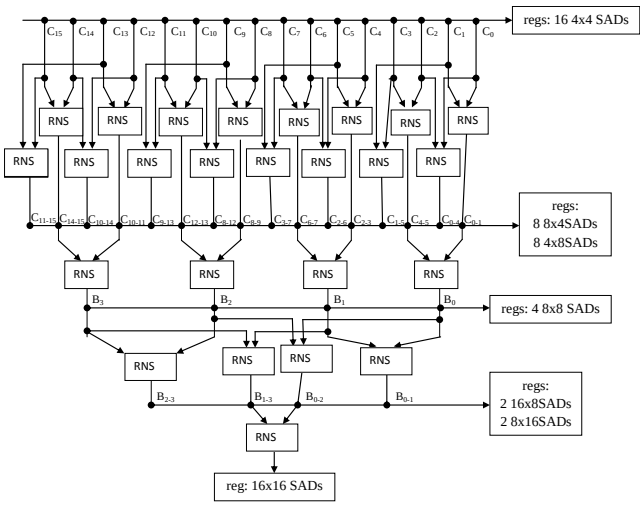

Fig. 9. RNS SAD Tree.

SAD comparator

The SAD comparator compares each 16x16 SAD of the selected object window, and then compares the finest SAD value. In Fig. 9, The SAD comparator first accumulates the output of SAD 16x16 present and previous SAD 16x16 stored values, and then it compares to conclude the minimum SAD value by using the comparator.

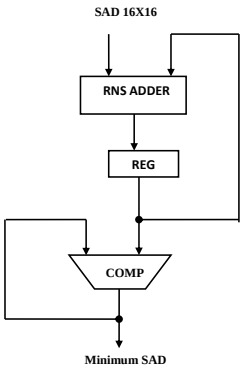

Fig. 9. SAD comparator.

Mode decision

In mode decision, the motion vector (MV) declares that the position of the minimum SAD in the x and y directions. Here, x and y coordinates are denoted as row and column directions respectively.

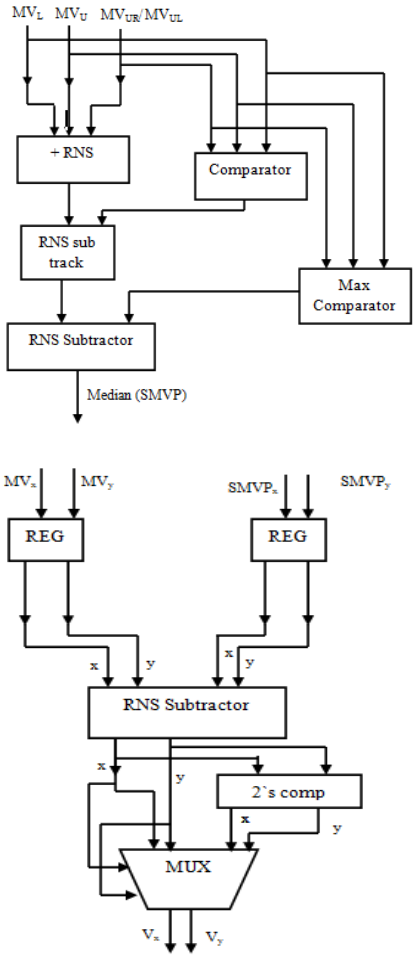

Fig. 10. Motion Vector Difference.

The MV_U, MV_L,MV_{UR},MV_{UL} are the Upper motion vector, Lower motion vector, Upper right motion vector and Lower left of the motion vectors respectively. These four MVs are located at the corresponding motion vector's position. From both MV_{UR}, MV_{UL}, the position of motion vector can be selected Either MV_{UR} or MV_{UL}. In Fig. 10 (a), A Single motion vector predictor (SMVP) can be predicted by the neighbor motion vector value. In Fig. 10 (b), the motion vector difference computes the difference between current MV position and SMVP positions. In Fig. 11(a) (b), the Best mode can be predicted from Both the corresponding SAD16x16 output and the motion vector difference values;. Here, λ_{motion} is common to both the corresponding SADs for sub-blocks of other sizes and each sub-block. It can be computed in the 16 4x4 input SAD trees.

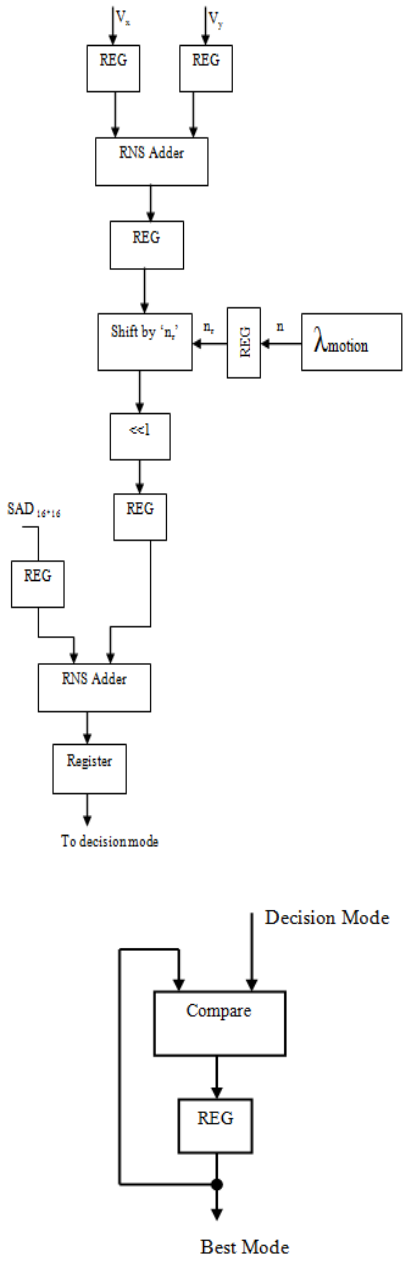

Fig. 11. Mode Decision Module.

8. Discussion and result

This architecture is implemented with TSMC CMOS 180nm technology. The characteristics and the performance of proposed algorithm based on the FSVBSME. Table 1 describes the comparison between the proposed algorithm with the existing algorithm techniques. In Ref. [5], After logic synthesis using SMIC 130nm standard cell library, the integer ME architecture allows 36k logic gates with the processing of 1280x720 (720HD) at 38fps under a clock frequency of 300MHz with full search block matching algorithm (FS-BMA) in a search range [-32, +32]. In Ref. [7], The Integer Motion Estimation processor chip was designed in the UMC 180nm technology, the result in a circuit with 32.3k logic gates. And a clock frequency of 300MHz can be estimated

with a processing capacity for HDTV (1920x1088 @30fps) and a search range of 32x32. In Ref. [10], a novel VLSI design was designed with TSMC CMOS 0.18μm technology, the result in the architecture occupies 15.8k gates at the frequency of 200MHz, which can constantly reduce about 66% of the RD related computation with a negligible quality loss. It is expected to be utilized in the hardware module in a real-time HDTV (1920×1088p) H.264 encoder. Proposed Method results show that synthesized with TSMC 180nm CMOS, the proposed design occupies 12.9k logic gates at 352MHZ and consumes 79mW power to encode 1920X1080 HDTV video frames at 30 frames per second.

Table 1. The Differences Between The Proposed and Existing Algorithm Techniques.

Reference	Technology	Search Range	Logic Gate Count	Frequency	Throughput	Power
Ref. [5]	SMIC 130nm	65 × 65	36k	300MHZ	1280×720@38fps	-
Ref. [7]	UMC 180nm	32 × 32	32.3k	300MHZ	1920×1088@30fps	115mW
Ref. [10]	TSMC 180nm CMOS	32 × 32	15.8k	200MHZ	1920×1088p@30fps	-
PROPOSED	**TSMC 180nm CMOS**	**32 × 32**	**12.9k**	**352MHZ**	**1920×1088p@30fps**	**79mW**

10. Conclusion

In this paper, the proposed algorithm implemented the fast mode decision and its architecture in RNS module for real time applications of motion estimation for video compression. By finding the ways of representing numbers, the best way is that reduce the sequential effect of carries on digits of most significance, which is a carry free arithmetic. The advantage of the RNS adder tree is that the absence of carry propagation between its arithmetic units, and for every addition operation. This carry-free arithmetic represents that the way of approaching on the speed at which addition can be performed. And it is no need to wait for carry values. Hence, It facilitates the realization of high-speed, low-power arithmetic. This proposed architecture is achieved that the less logical elements, high throughput required to perform real time motion estimation. In the results the proposed method is synthesized with TSMC 180nm CMOS, and occupies 12.9k logic gates at 352MHZ and consumes 79mW power to encode 1920X1088 HDTV video frames at 30 frames per second in search range of 32x32.

"Compliance with Ethical Standards"

1. Disclosure of potential conflicts of interest - No conflict of Interest.
2. Research involving Human Participants and/or Animals –NA
3. Informed consent – NA

References

[1] Rec. H.264/ISO/IEC 11496-10, "Advanced Video Coding ", Final Committee Draft, Document JVT-E022, 2002.
[2] Yeu-Shen-Jehng, Liang-Gee-Chen, Tzi-Dar Chiueh, IEEE transaction on signal processing, **41**(2), (1993).
[3] Swee Yeow Yap, John V. McCanny, IEEE computer society, 2003.
[4] Swee Yeow Yap, John V. McCanny, IEEE Transactions On Circuits And Systems, **51**(7), 384 (2004).
[5] Meihua Gu, Ningmei Yu, Lei Zhu, Wenhua Jia, Journal of Computational Information Systems, **7**(4), 1310 (2011).
[6] Jun Sung Park, Hyo Jung Song, World Academy of Science, Engineering and Technology, **13**, 637 (2008).
[7] G. A. Ruiz, J. A.Michell, Elsevier Signal Processing: Image Communication, **26**, 289 (2011).
[8] Haibing Yin, InTech chapter. 8, Advanced Video Coding for Next-Generation Multimedia Services, 157 (2012).
[9] Chuan-Yu Cho, Shiang-Yang Huang, Jenq-Neng Hwang, Jia-Shung Wang, IEEE International Conference on Image Processing, **3**, 1016 (2005).
[10] Shen Li, Xianghui Wei, Takeshi Ikenaga, Satoshi Goto, GLSVLSI, 20-24, (2007).
[11] N. S. Szabo, R. I. Tanaka, McGraw-Hill, 1967.
[12] G. Dimauro, S. Impedovo, G. Pirlo, IEEE Transaction on Computers, **42**(5), 608 (1993).

*Corresponding author: research4charles@gmail.com