Signal Processing: *Image Communication* 26 (2011) 289–303

Contents lists available at ScienceDirect

Signal Processing: *Image Communication*

journal homepage: www.elsevier.com/locate/image

An efficient VLSI processor chip for variable block size integer motion estimation in H.264/AVC

G.A. Ruiz *, J.A. Michell

Dpto. de Electrónica y Computadores, Facultad de Ciencias, Universidad de Cantabria, Avda. de Los Castros s/n, 39005 Santander, Spain

ARTICLE INFO

Article history:
Received 23 July 2010
Accepted 13 April 2011
Available online 28 April 2011

Keywords:
H.264/AVC
Full search motion estimation
Variable block size motion estimation (VBSME)
VLSI architecture

ABSTRACT

Motion estimation (ME) is the most critical component of a video coding standard. H.264/AVC adopts the variable block size motion estimation (VBSME) to obtain excellent coding efficiency, but the high computational complexity makes design difficult. This paper presents an effective processor chip for integer motion estimation (IME) in H264/AVC based on the full-search block-matching algorithm (FSBMA). It uses architecture with a configurable 2D systolic array to obtain a high data reuse of search area. This systolic array supports a three-direction scan format in which only one row of pixels is changed between the two adjacent subblocks, thus reducing the memory accesses and saving clock cycles. A computing array of 64 PEs calculates the SAD of basic 4×4 subblocks and a modified Lagrangian cost is used as matching criterion to find the best 41 variable-size blocks by means of a tree pipeline parallel architecture. Finally, a mode decision module uses serial data flow to find the best mode by comparing the total minimum Lagrangian costs. The IME processor chip was designed in UMC 0.18 μm technology resulting in a circuit with only 32.3 k gates and 6 RAMs (total 59kBits on-chip memory). In typical working conditions (25 °C, 1.8 V), a clock frequency of 300 MHz can be estimated with a processing capacity for HDTV (1920×1088 @ 30 fps) and a search range of 32×32.

© 2011 Elsevier B.V. All rights reserved.

1. Introduction

H.264/AVC, developed by the Joint Video Team (JVT), is nowadays drawing considerable attention because of its network friendliness and the high video quality achieved with both low and high bit rates [1]. Compared with previous video standards, H.264/AVC can provide up to 50% coding efficiency for different bit rates and video definitions [2]. Fig. 1 presents a simplified block diagram of the H.264 encoder with the following main blocks: motion estimation (ME), motion compensation (MC), intra prediction, forward transform (FT), forward quantization (FQ), inverse quantization (IQ), inverse transform (IT), entropy coding and de-blocking filter. Initially, most of the work done on H.264 was oriented toward its software implementation [3]. However, in recent years the contributions to the VLSI hardware implementation of the H.264 have increased greatly in order to enable the implementation of fast architectures for real-time video applications [4–6].

ME based on a block-matching strategy is the most important part of H.264/AVC in exploiting the temporal redundancy between successive frames but it is also the most time consuming part in the coding framework. It requires large amounts of computation and accounts for 60–90% of encoding time. In H.264, a video frame is first split using macroblocks (MB) of size 16×16 in a VBSME approach [1]. This approach provides a better estimation of small and irregular motion fields and allows a better adaptation of the motion boundaries to object boundaries; all with the aim of

* Corresponding author. Fax: +34 942 201402.
E-mail address: ruizrg@unican.es (G.A. Ruiz).

0923-5965/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.image.2011.04.006

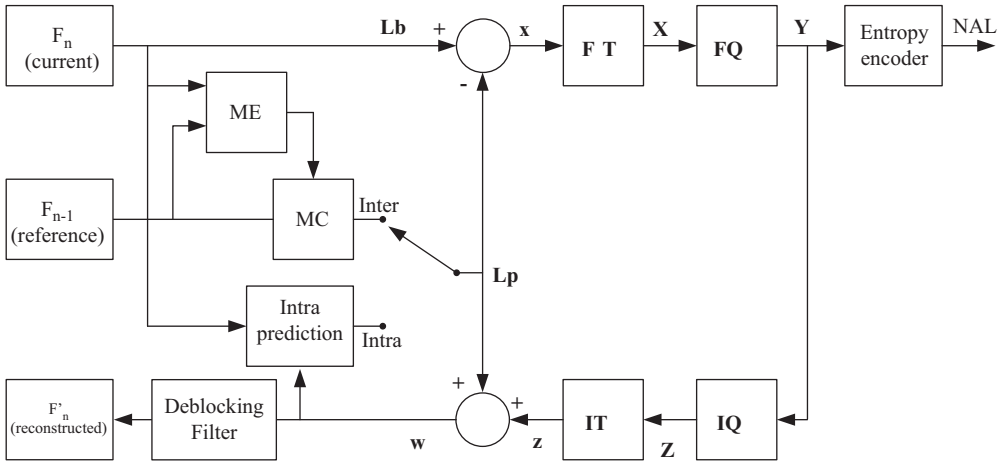

Fig. 1. Diagram of the H.264 encoder.

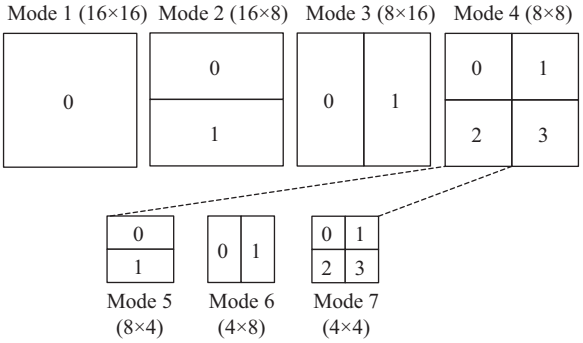

Fig. 2. Subblock partitions.

reducing the number of bits required for coding prediction errors. In VBSME, each MB may be segmented into subblocks of different sizes, as illustrated in Fig. 2. ME is carried out in 7 different modes: one 16×16 MB (Mode 1), two 16×8 subblocks (Mode 2), two 8×16 subblocks (Mode 3) and four 8×8 subblocks (Mode 4). In turn, each 8×8 subblock is also split up into two 8×4 subblocks (Mode 5), two 4×8 subblocks (Mode 6) and four 4×4 subblocks (Mode 7). The total number of possible partitions is 41. ME refines the best candidate for each subblock's hierarchy in two phases: Integer Motion Estimation (IME) and Fractional Motion Estimation (FME). IME finds the best integer motion vector (MV) for all 41 variable-size blocks. FME refines those MVs in quarter-pixel precision using a 6-tap filter and a MV-bit-rate estimation. A pipeline architecture is the best solution to implement IME and FME [4,6,7].

There are many proposed IME algorithms and architectures based on the criteria of search, matching simplification, bit-width reduction, memory access or area/power, among others, where the system designer can choose the best trade-off for a specific application [8,9]. FSBMA in the VBSME guarantees the best results by performing exhaustive matching of all candidate blocks. Most hardware video encoder designs adopt FSBMA in IME due to its excellent quality and data flow regularity, although it requires the maximum computation complexity and memory bandwidth. Systolic arrays [10,11] composed of locally connected processing elements (PEs) has been considered to be the best option to implement FSBMAs for many reasons: pipelined data flow via the local connections does not require any control overhead; reuse of data for each PE by propagating through the array significantly reduces the memory bandwidth; and high clock frequencies and high processing speeds are achieved due to the small load capacities. 1D systolic arrays are attractive in low-end products because of their low hardware cost and desirable processing capability, but they are high in cost in terms of latency and efficiency [12–14]; Ref. [15] presents a 2D architecture based on 1D to attain low latency and high throughput. Indeed, many VLSI implementations propose 2D systolic arrays to be more suitable for high-end real-time usage. References which adopt the complete FSBMA include the 2D architecture with a simple regular control in [16], the novel memory-access with minimum off-chip memory bandwidth in [19], the high-performance reconfigurable architecture to support a scan format for a high data reuse within the search area in [20], the bit serial architecture in [22] and the high throughput design in [39]. Modifications of the FSBMA to reduce either hardware or computing time, at the cost of introducing some video quality loss, can be found in the soft algorithm to simplify the predicted MV and the early termination of motion search used in [21], the multi-resolution IME algorithm presented in [23], the adaptive size in the search area depending on the degree of motion activity in [24,25], the modified algorithm to reduce hardware based on data dependency of motion vector prediction, pixel truncation and subsample proposed in [18], the IP with coarse and fine searches in [26] and the inter-candidate 4-parallel data reuse scheme with 16 2D PE-arrays in [27].

Finally, other fast ME algorithms, which are alternatives to FSBMA, have been developed with the aim of reducing the computational complexity and establishing a trade-off between efficiency and image quality [28,29]. However, this reduction is often at the cost of losses in terms of visual quality and/or irregularities in data flow. This means that these algorithms lead to irregular memory access and difficulty in data reuse as well as

introducing losses in terms of peak signal-to-noise ratio (PSNR). Often the irregular local search patterns can easily lead to a minimum local (suboptimal result) as opposed to a global minimum local by FSBMA. Examples of fast ME algorithms based on decimation of checking points are the three-step search [30], hexagon search [31] and diamond search [32]. FSBMA architectures are typically implemented with systolic mesh-connected arrays which are not suitable for the fast ME algorithms because of their unpredictable data flow and hardly parallelizable sequential control. Although the fast ME implementations achieve good efficiency over the FSBMA architectures, they are too rigid for a broad range of applications [33].

This paper presents a high-performance VLSI processor chip for IME based on the full-search VBSME algorithm for H.264/AVC video coding standard. The configurable 2D systolic array architecture supports a three-direction scan format for a high data reuse of the search area, an array of 16×4 PEs compute the SAD of basic 4×4 subblocks and a modified Lagragian cost is used as matching criterion to find the best 41 variable-size blocks by means of tree pipeline parallel architecture. The mode decision module selects the best mode and best MVs by comparing the total minimum Lagrangian costs. A prototype of the IME processor chip was designed in UMC 0.18 μm technology using a standard cell methodology. It achieves enough processing capacity for HDTV (1920×1088 @ 30 fps) with a search range of 32×32 in a high-performance circuit operating at 300 MHz and occupying reduced area. The remainder of this paper is organized as follows. Section 2 describes the proposed Lagrangian cost to be implemented in a parallel architecture. Section 3 presents a detailed description of the IME architecture including its main modules such as the processing unit, motion estimation, computation of the Lagrangian cost and motion decision unit. The results and comparisons of VLSI chip processor implementation are listed in Section 4. Finally, the conclusions are stated in Section 5.

2. Proposed Lagrangian cost in IME

The reference software JM of H.264, which is available on-line at [3], basically supports two methods to carry out a mode decision in terms of the cost calculation criteria: Motion Vector (MV) cost and Rate Distortion Optimization (RDO) cost. RDO cost is mainly used for selection prediction mode and MV cost in ME. RDO involves forward integer transform, quantization, dequantization or scaling, inverse integer transform and entropy coding. As a result, most real-time hardware encoders do not implement RDO because of its high computational complexity [4–6]. However, in order to approach this problem, alternative non-optimal coding methods have recently been proposed [34–36].

ME is optimized using the Lagragian method as the best approach for bit allocation. This method finds, for each one of the 41 subblocks belonging to a MB, the MV that minimizes the matching error within the search area using the cost function ($J_{M\times N}$) defined as

$$J_{M\times N}=SAD_{M\times N}+\lambda_{\text{motion}}R(|\mathbf{mv}-\mathbf{p}|)=SAD_{M\times N}+\lambda_{\text{motion}}R(|mv_x-p_x|+|mv_y-p_y|) \quad (1)$$

where M and $N\in\{4,8,16\}$ are the vertical and horizontal dimensions of the specific subblock, $\mathbf{mv}=(mv_x, mv_y)$ is the MV, $\mathbf{p}=(p_x, p_y)$ is the predicted MV, and λ_{motion} is the Lagrange multiplier. The $SAD_{M\times N}$ or Sum of Absolute Distortion is given for **mv** by

$$SAD_{M\times N}=\sum_{i=1}^{M}\sum_{j=1}^{N}\left|Y_{i,j}-X_{i-mv_x,j-mv_y}\right| \quad (2)$$

where **Y** is the reference video signal and **X** is the current video signal. In Eq. (1), λ_{motion} is the Lagrangian multiplier and $R(|\mathbf{mv}-\mathbf{p}|)$ represents the bits used to encode the motion information, both usually obtained from a look-up table. Here, $\mathbf{mv}-\mathbf{p}$ is the MV displacement (or the MV prediction error) between the true MV, **mv**, found as a result of the search algorithm and the predicted MV, **p**, fixed by the MVs of some previously encoded neighboring blocks of the current block. To calculate **p**, the MV of the neighbor blocks must be available or sufficiently estimated, as they not only depend on neighboring MBs, but also on previous subblocks within a MB [1].

For the sake of clarity, Fig. 3a graphically shows an example of the definition of the different parameters used to compute the MV cost ($J_{M\times N}$) of an 8×8 subblock. Fig. 3b illustrates the definition of **p** for an 8×8 subblock, labeled $\mathbf{p}_{8\times8}$. In this case, the $\mathbf{p}_{8\times8}$ is computed from the

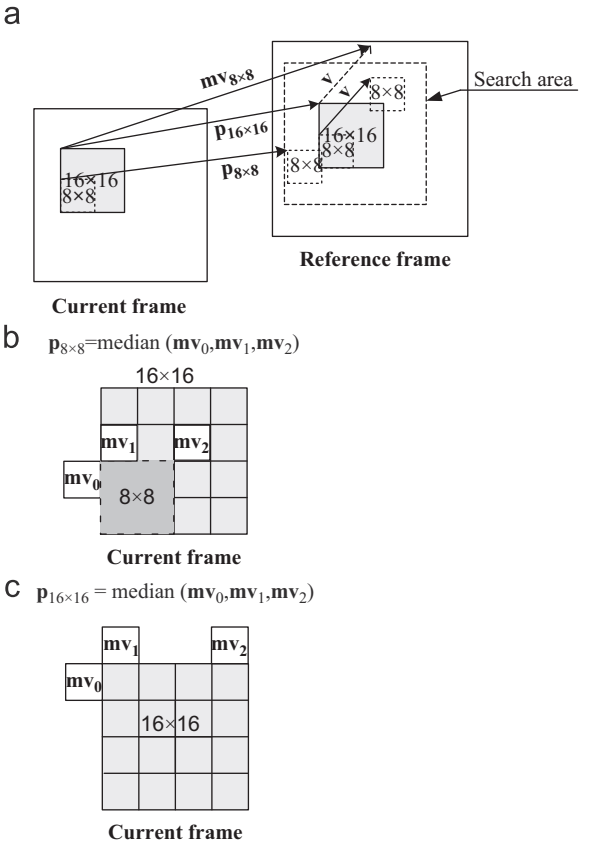

Fig. 3. MV definition for an 8×8 subblock: (a) graphic representation, (b) definition of $\mathbf{p}_{8\times8}$, (c) definition of $\mathbf{p}_{16\times16}$.

median of left ($\mathbf{mv}_0$), top ($\mathbf{mv}_1$) and top-right ($\mathbf{mv}_2$); $\mathbf{mv}_0$ belongs to a previous 16×16 MB, and $\mathbf{mv}_1$ and $\mathbf{mv}_2$ belong to the 4×4 subblocks which have just been processed in the same MB. The prediction $\mathbf{p}_{16\times16}$ is common to every subblock and represents an offset of 16×16 MB defined using the MB boundary's MVs as is shown in Fig. 3c. The local vector $\mathbf{v}$ represents the motion of the subblock within the search region. The IME algorithm in the H.264/AVC searches for the value of $\mathbf{mv}=\mathbf{v}+\mathbf{p}_{16\times16}$ that minimizes Eq. (1), although the vector $\mathbf{mv}-\mathbf{p}_{8\times8}$ will finally be coded.

The computation complexity of H.264/AVC for computing Eq. (1) applied to each 41 subblocks belonging to a MB leads to a hardware design where two main problems must be taken into account. First, the data dependency of the predicted MV in the VBSME and Lagrangian cost both make parallel computation prohibitive. This means that the $\mathbf{p}$ vectors of adjacent subblocks are not available in a parallel scheme; in the *JM* software, each subblock is computed sequentially according to a predefined order. Second, in order to avoid huge numbers of memory accesses, architectures with a high degree of parallelism and reused data must be the aim for the implementation of fast architectures for real-time video.

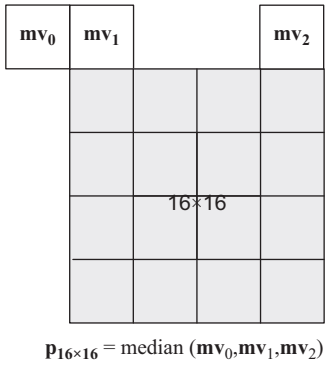

Fig. 4. Proposed definition of $\mathbf{p}_{16\times16}$ for a MB.

The IME is not specified in the H.264 standard. This gives developers some flexibility in choosing an IME design. In many low hardware architectures, SAD is used as the only matching criterion to avoid computing the Lagragian cost, but it leads to a non-optimum IME scheme. Indeed, we propose a modification of Lagragian cost with the aim of making available parallel computation with a slight penalty in the coding quality. This modification consists in performing a simplification of Eq. (1) giving a proposed Lagragian cost for a $M\times N$ subblock defined as

$$J_{M\times N}=SAD_{M\times N}+\lambda_{\text{motion}}R(|v_x|+|v_y|) \tag{3}$$

Here, the predicted MVs of subblocks are not necessary; this means $\mathbf{p}=\mathbf{p}_{16\times16}$ for all cases meaning that $\mathbf{mv}-\mathbf{p}=\mathbf{v}$. Only a common offset ($\mathbf{p}_{16\times16}$) for a MB is considered which is computed from the top-right, top and top-left MVs of the upper boundary MBs as is shown in Fig. 4, instead of the original definition described in Fig. 3c. This modified MV prediction, similar to that proposed in [18], enables data to be fetched from the nearest right *MB* without finishing the processing of the current MB. As a

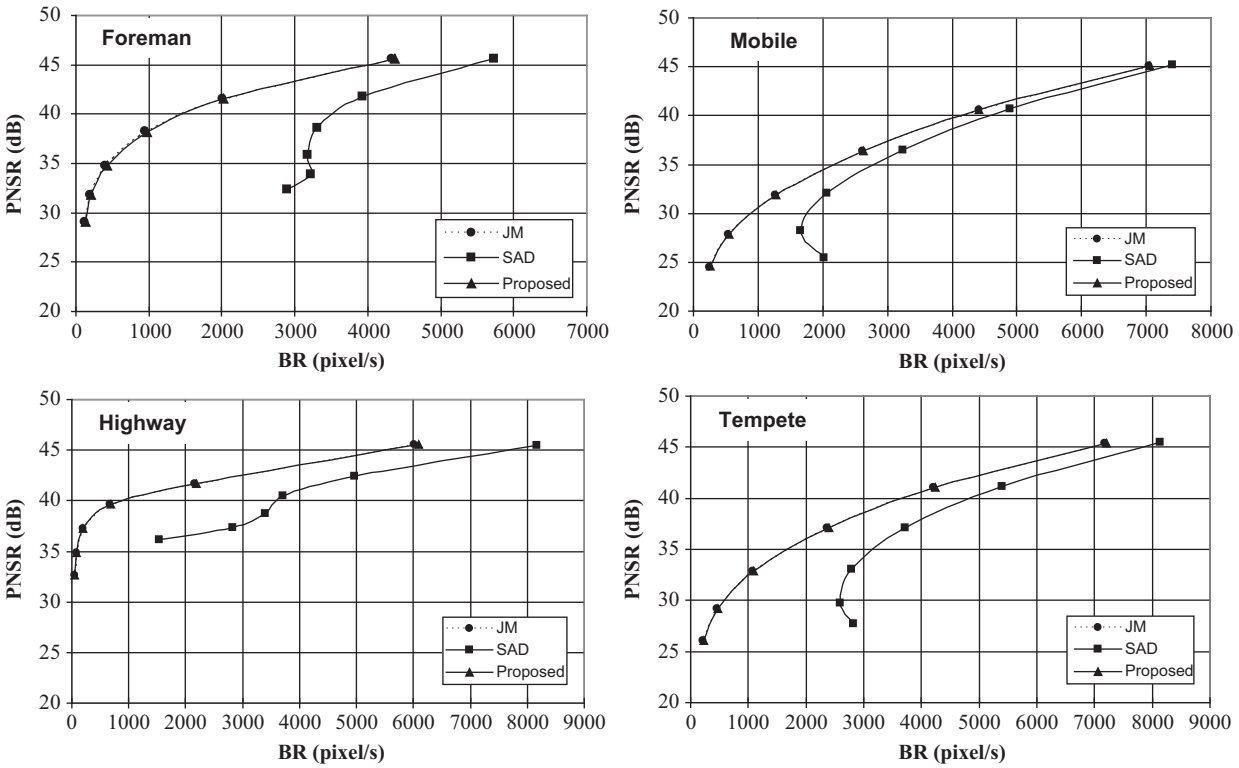

Fig. 5. Rate-distortion curves of various standard sequences.

Table 1
Performance comparison of motion estimation approaches.

	QP					BDPSNR (dB)	BDBR (%)
	15	**20**	**25**	**30**	**35**		
Foreman							
JM							
PSNR	45.66	41.60	38.23	34.79	31.82	−0.139	3.04
BR	4333.74	2001.71	950.84	404.44	198.3		
Proposed							
PSNR	45.65	41.59	38.22	34.78	31.79		
BR	4365.77	2025.73	972.39	423.58	212.77		
Mobile							
JM							
PSNR	45.14	40.59	36.40	31.89	27.88	−0.040	0.60
BR	7045.54	4406.98	2614.87	1268.92	543.93		
Proposed							
PSNR	45.14	40.59	36.40	31.89	27.89		
BR	7057.93	4420.34	2629.27	1278.88	553.51		
Highway							
JM							
PSNR	45.53	41.68	39.66	37.24	34.9	−0.040	1.62
BR	6016.18	2162.6	671.91	198.17	87.29		
Proposed							
PSNR	45.53	41.67	39.66	37.24	34.9		
BR	6099.57	2182.27	683.93	202.35	88.62		
Tempete							
JM							
PSNR	45.40	41.09	37.11	32.87	29.21	−0.065	1.12
BR	7173.18	4207.25	2371.46	1082.71	461.52		
Proposed							
PSNR	45.40	41.08	37.11	32.87	29.21		
BR	7202.34	4231.26	2395.78	1098.2	473.75		
Average						−0.07	1.6

result, this overlapping in the processing of two adjacent MB saves clock cycles as it is unnecessary to spend time in fetching data to the external memories. Therefore, the vector $\mathbf{v}$ used as a parameter in the computation of the term $R(|v_x|+|v_y|)$ in Eq. (3) only depends on local movement of a subblock in the search. This enables the independent computation of this term in parallel for all 41 different subblocks of the VBSME. After finishing the IME, the coded vector will be $\mathbf{v}+\mathbf{p}_{16\times16}-\mathbf{p}_{8\times8}$ because the prediction vectors $\mathbf{p}_{16\times16}$ and $\mathbf{p}_{8\times8}$ are now available as all the subblocks have been processed.

The JM reference software's C code [3] has been rewritten to analyze the effect of the proposed Lagrangian cost based on Eq. (3) in the video quality. These modifications have been tested on several very popular video sequences in video standardization (Foreman, Mobile, Highway, Tempete, available from http://trace.eas.asu.edu/yuv/index.html) with different texture characteristics. This analysis has been performed with RDO disabled, baseline mode, group of pictures (GOP) IPPP with an intra period of 20, search range ±16 with a full-search IME scheme, Hadamard transform off and rate control disabled. Fig. 5 shows the rate-distortion curves of these sequences for three IME schemes: Original (JM), only SAD and the proposed Lagrangian cost. SAD obtains the worst results as both RDO and MV cost are disabled. This means that the IME implementation based on SAD when it is used as the only cost calculation criteria provides non-optimum results. On the contrary, in the proposed simplification of Lagragian cost, not only does the parallel computation in an MB become feasible, but the quality is also maintained with slight differences with respect to JM. For the sake of clarify, Table 1 shows the numerical results comparing the JM and the proposed method. The average loss in video quality can be estimated to cause a reduction of 0.01–0.02 dB for the PSNR and an increase of roughly 1–2% for the BR. For a further numerical comparison, these results are contrasted in terms of the Bjontegaard delta PSNR (BDPSNR) and the Bjontegaard delta bit rate (BDBR) [40], which give the average bit rate and PSNR metric over several QP values of two RD curves. As a result, the degradation of its coding efficiency is very negligible: it is only -0.07 dB and 1.6%, respectively. This means that the proposed method is very efficient to reduce the computational complexity without any meaningful degradation.

3. Hardware architecture of the IME

Most FSBMA algorithms exploit the large amounts of overlapped data among the adjacent blocks. The proposed architecture for inter-prediction uses a 2-D systolic array to compute the SAD of 4×4 blocks. To optimize that overlapped data, the search area adopts a scan format scheme that supports three scan directions, unlike other traditional 2-D systolic array with one scan direction.

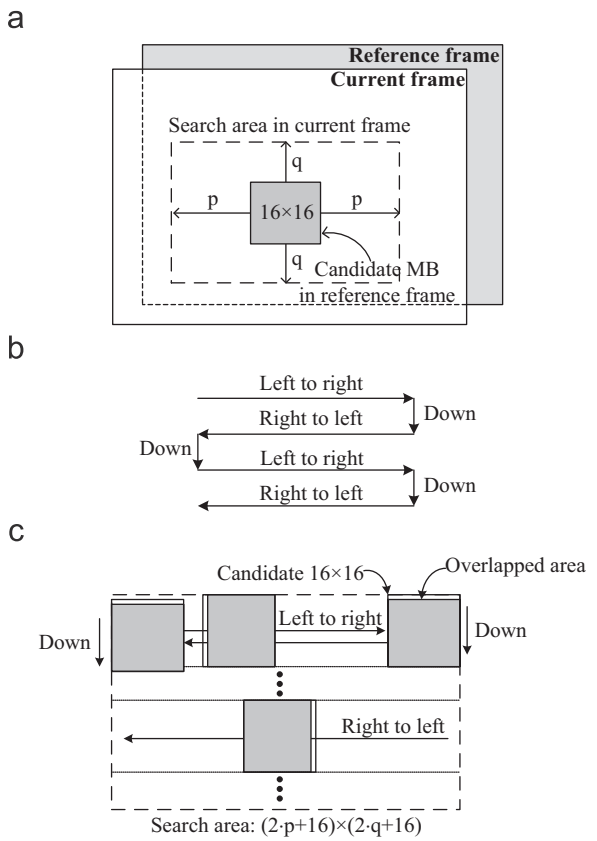

Fig. 6. Searching scheme: (a) search range, (b) scan direction, (c) scan scheme of the search area.

This reused data saves roughly 30% of the memory access cycles achieving high throughput rate, low memory bandwidth and high hardware utilization.

In the proposed search scheme, the candidate 16×16 MB in the reference frame covers the search range $[\pm p, \pm q]$ in the current frame as shown in Fig. 6a. This search range is equivalent to a search area of $(2p+16) \times (2q+16)$ pixels, which is processed according to the scan order scheme shown in Fig. 6b. In a more detailed description of Fig. 6c, the odd row is scanned from left to right and the following even row from right to left. At the end of these rows, the search direction is changed and a down operation of one row is performed. As a result, this 2-D reconfigurable architecture enables the greatest reuse of overlapping data by supporting a three direction scan scheme in order to reduce the number of memory accesses to a minimum.

Based on this search scheme, Fig. 7 shows the proposed architecture for computing the IME of H.264. This architecture makes a trade-off between the processing time and the hardware utilization to achieve sufficient capacity to encode the high-resolution real-time video stream for HDTV at a low area cost. Two external memories contain data in an 8-bit format corresponding to the reference frame and the current frame. Initially, external data are read through a 4-pixel input data port and stored in two local memories: RAM1, which is made up of four double-port memories of 400×32 bits, stores the search area, and RAM2 with a size of 64×32 bits stores the current MB. Since the 4×4 is the smallest block in the MB partitions, the processing unit computes 16 SADs in parallel of 4×4 subblocks belonging to the MB by means of two 2D registers – REGS of 16×20 pixels and REGC 16×16 pixels – and 64 processing elements (PE). The search range is configurable to 8×8, 16×16, 32×32 and 64×64 pixels and fixes the number of clock cycles of that process; other geometries in the search area are also easily available. These SADs are input into the motion estimation module to obtain the 41 MVs and their corresponding Lagragian cost from those that minimize Eq. (3). Here, the SADs of larger subblocks are merged in a hierarchical pattern by adding up subblocks of lower sizes. Finally, the mode decision module selects the best mode and best MVs which are stored in RAM3 with a size of 180×32 bits for use in the MV prediction to compute the prediction vector $\mathbf{p}_{16 \times 16}$ according to scheme described in Section 2. These vectors are used in the addressing module to generate the memory addressing to fetch all data in the external memories for the next MB.

3.1. Processing unit

The processing unit is based on a 2D systolic array combined with 64 PEs to compute in parallel 16 SADs of 4×4 subblocks by means of a data-reuse scheme in order to achieve high throughput rate, low memory bandwidth and efficient hardware utilization at high operating frequency. The 2-D systolic array scheme is popular in many ME implementations due to its regular and simple structure because it can reuse data from regular flow to decrease memory access by parallel processing and pipelining. In the proposed processing unit, the systolic array is made up of two registers: REGS of 16×20 pixels and REGC of 16×16 pixels. REGS stores the search area and REGC the current MB. REGS contains a MB to be processed along with the one stored in REGC; in REGS an extra 16×4 column is added to adopt a three-scan format allowing both to save extra clock cycles in loading data and to reduce memory accesses.

Functionally, REGS and REGC are divided into elementary 4×4 subblocks where each subblock inside shifts data from up to down. REGS is a configurable register with four modes: right-to-left shift, left-to-right shift, rotation and down. These modes are easily implemented through multiplexers at the input of each subblock. The right-to-left shift (Fig. 8a) performs a shifting operation in the horizontal direction by inputting data, which are fetched from RAM1, from left to right. On the contrary, the right-to-left shift (Fig. 8b) does the same operation but in the opposite direction from left to right. Rotation (Fig. 8c) means a cyclic shifting operation in the same subblock. Finally, the down operation moves a horizontal line of pixels up from the subblock below into the one above when the MB moves into the row below. On the contrary, REGC only performs rotation operations synchronized with REGS using a "clock gating" strategy to reduce hardware and save power.

In order to support the three-scan directions of the search process in an area of $(2p+16) \times (2q+16)$ pixels, the processing unit follows a data flow scheme shown in Fig. 9. After initializing the registers which takes 16 clock

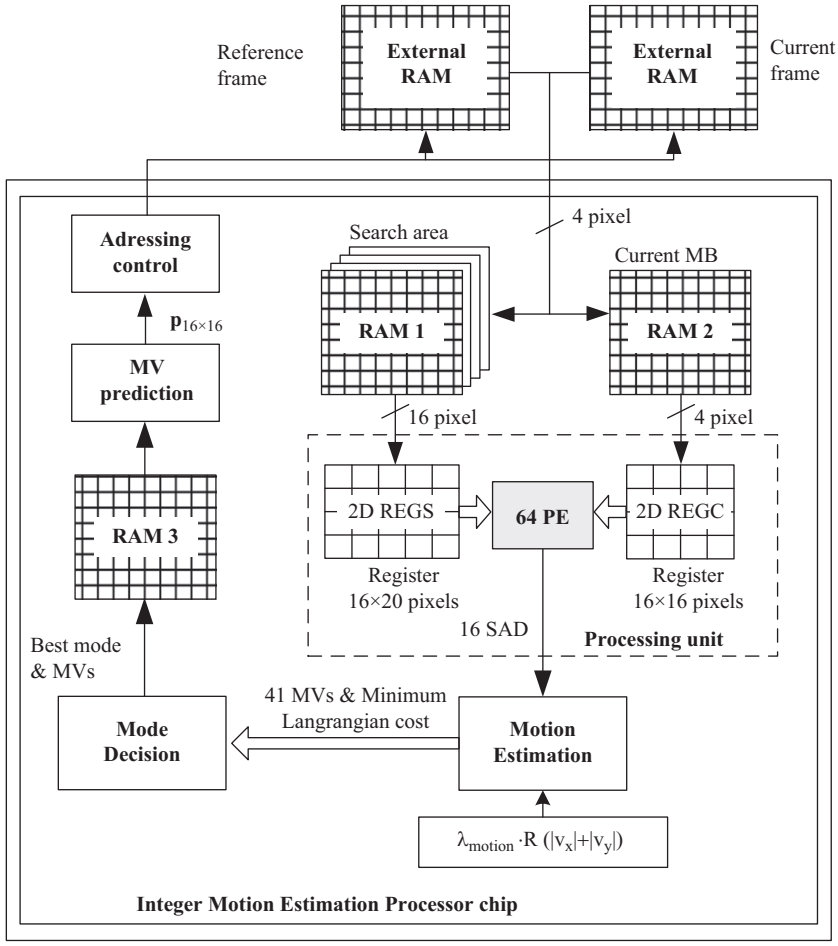

Fig. 7. Architecture of the IME processor chip.

cycles, the cyclic process starts with a right-to-left shifting for $2q+1-4$ clock cycles. Once the down (1 clock cycle) and the rotation (4 clock cycles) operation are finished, the left-to-right shifting takes another $2q+1-4$ clock cycles repeating the whole process after another down and rotation operation. The processing of a row takes $2q+2$ clock cycles and the processing of the whole search area takes $(2q+2)\times(2p+1)+16$ clock cycles.

For the sake of clarity, Fig. 10 shows the timing diagram corresponding to the first 4×4 subblock as a simple example of the different operations carried out in the processing unit. Each subblock relies on 4 processing elements (PE) connected to this subblock and to the nearest subblock to the right. PE0 processes all the pixels of REGS belonging to the first row, PE1 processes pixels of the second row where 3 pixels correspond to the subblock and 1 pixel to the right subblock, PE2 processes 2 pixels of the third row and 2 pixels to the right subblock, and finally PE3 processes 1 pixel of the fourth row and 3 pixels to the right subblock. This use of the one right shifting position in the connection of each PE enables the emulation of one moving pixel in the current MB of the search area. In this timing diagram, the process starts by initializing the REGS. After 16 clock cycles and following the data flow specified in Fig. 9, the four PEs sequentially compute the SAD each 4 clock cycles with 1 delayed clock cycle among them. Only the down operation has an idle clock cycle where no SAD is computed. In REGS, there are 4 PEs for each 16 left subblocks – which total 64 PEs – generating in parallel 16 SADS to be processed in the motion estimation unit.

Each PE, whose architecture is shown in Fig. 11, computes the SAD of a 4×4 subblock by processing 4 input pixels from the REGS and REGC. A PE is composed of an absolute difference module, an adder tree and a final adder-accumulator. The absolute difference module implements the absolute difference operation [6,37] expressed as

$$\sum_{j=0}^{3}|Y_{i,j}-X_{i,j}| = \sum_{j=0}^{3}\begin{cases}\overline{Y_{i,j}+\overline{X}_{i,j}} & \text{if } Y_{i,j}\le X_{i,j}\\ Y_{i,j}+\overline{X}_{i,j}+1 & \text{if } Y_{i,j}>X_{i,j}\end{cases} \quad (4)$$

where $Y_{i,j}$ represents a reference pixel stored in REGS and $X_{i,j}$ denotes a current pixel stored in REGC. To implement Eq. (4), a first level of adders computes $Y_{i,j}+\overline{X}_{i,j}$ and the most significant bits of the output $\{S_3, S_2, S_1, S_0\}$ are used to decide whether to invert the output through a bit-XOR or not. However, 1 must be added when $Y_{i,j}>X_{i,j}$, which is equivalent to having the corresponding output S_j at 1. In order to reduce hardware, the addition of $\{S_3, S_2, S_1, S_0\}$ is

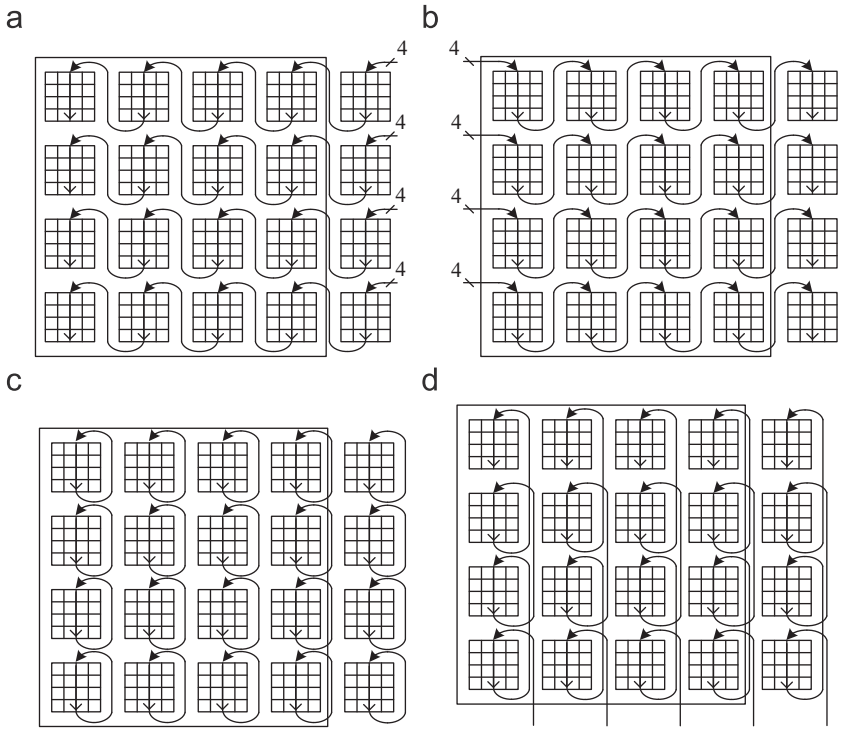

Fig. 8. Operation modes of REGS: (a) right-to-left shift, (b) left-to-right shift, (c) rotation and (d) down.

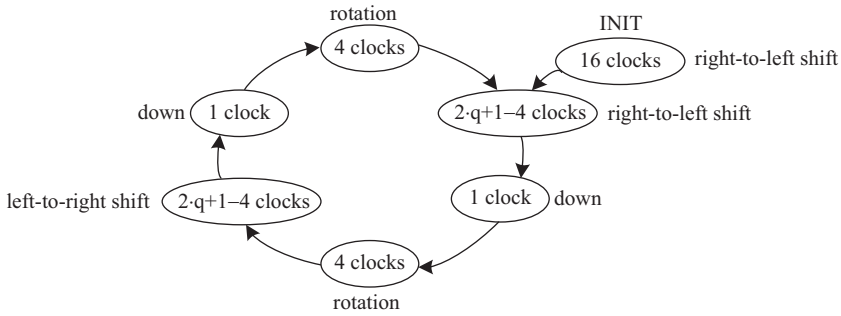

Fig. 9. Flow diagram of the processing unit.

split up among the four adders of the circuit acting as a carry input. The adder tree scheme calculates partial SADs by summing all absolute differences. Here, a pipeline stage has been inserted to reduce the critical path. The final adder accumulator circuit obtains the final SAD for a subblock after 4 clock cycles and the register is initialized to 0 to compute the next SAD. Fig. 11 also shows all bus widths to prevent overflow. The maximum value of SAD, which in the worst case of all absolute differences is 255, is $255 \times 16 = 4080$, which can be represented by 12 bits in an unsigned number.

3.2. Motion estimation

The motion estimation module selects the best 41 MVs and their Lagragian cost from those 16 SADs generated in the computing unit that minimize the matching error of Eq. (3). In this equation, the Lagrangian cost only depends on local MVs as the subblock's prediction vectors are considered to be null. This makes the implementation of the motion estimation module easier because all SADs, which are generated in the same clock cycle, have the same associated MV as they are computed in parallel. As a result, the Lagrangian cost term $\lambda_{\text{motion}} R(|v_x| + |v_y|)$ is common to every subblock and the corresponding SADs for subblocks of other sizes can be computed in a hierarchical tree from the 16 4×4 input SADs.

The pipeline architecture of the motion estimation module is illustrated in Fig. 12a. The input data are introduced each clock cycle and processed in a pipeline scheme from the lowest subblock size 4×4 (Mode 7) to the biggest size 16×16 (Mode 1) according to a hierarchical tree of 5 levels based on the partitions shown in Fig. 2. Each of these 41 different subblocks has registers to store the minimum cost and MVs. In the first level, the circuit shown in Fig. 12b is used. It is made up of 3 registers, an adder, a comparator and

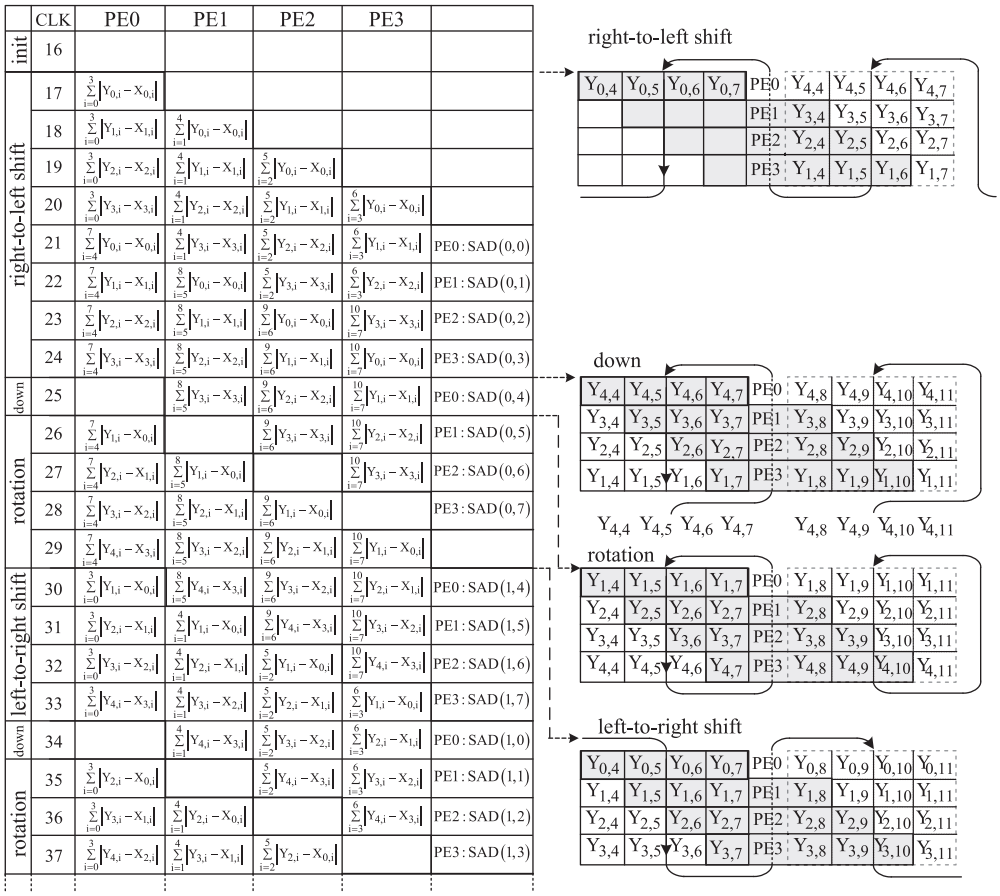

Fig. 10. Timing diagram of the processing unit.

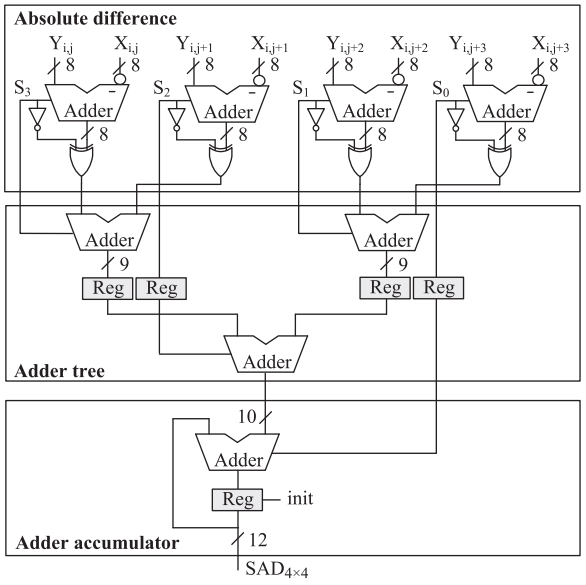

Fig. 11. Architecture of PE.

two multiplexers. RegA stores the minimum cost and it is initially set to a maximum value, RegB stores the MVs and RegC stores the SAD to be used in the following level. In this circuit, the addition $SADi+\lambda_{\text{motion}}R(|v_x|+|v_y|)$ is compared with the minimum cost stored in the RegA, then RegA and RegB will be updated depending on the result of that comparison. The rest of levels use the circuit in Fig. 12c. Here, an additional adder computes the SAD of the current subblock from the SAD's small input subblocks before adding $\lambda_{\text{motion}}R(|v_x|+|v_y|)$. RegC is used to store the new SAD to be passed on to the next level and to reduce the critical path of the arithmetic to an addition and a comparator. As a result, the minimum cost and MVs for all 41 cases are concurrently generated in a regular pipeline workflow with a latency of 6 clock cycles.

3.3. Computing of $\lambda_{motion}R(|v_x|+|v_y|)$

In the term $\lambda_{\text{motion}}R(|v_x|+|v_y|)$, λ_{motion} is the Lagrangian multiplier imposed by a suitable rate constraint and it is calculated from the following empirical formula

$$\lambda_{motion} = \sqrt{0.85 \times 2^{(QP-12)/3}} \tag{5}$$

where QP is the quantization parameter. It takes an integer value (from 0 to 51) and determines the level of coarseness of the quantization process. The relationship between QP and λ_{motion} in Eq. (5) is extracted through experiments similar to the one described in [38] for

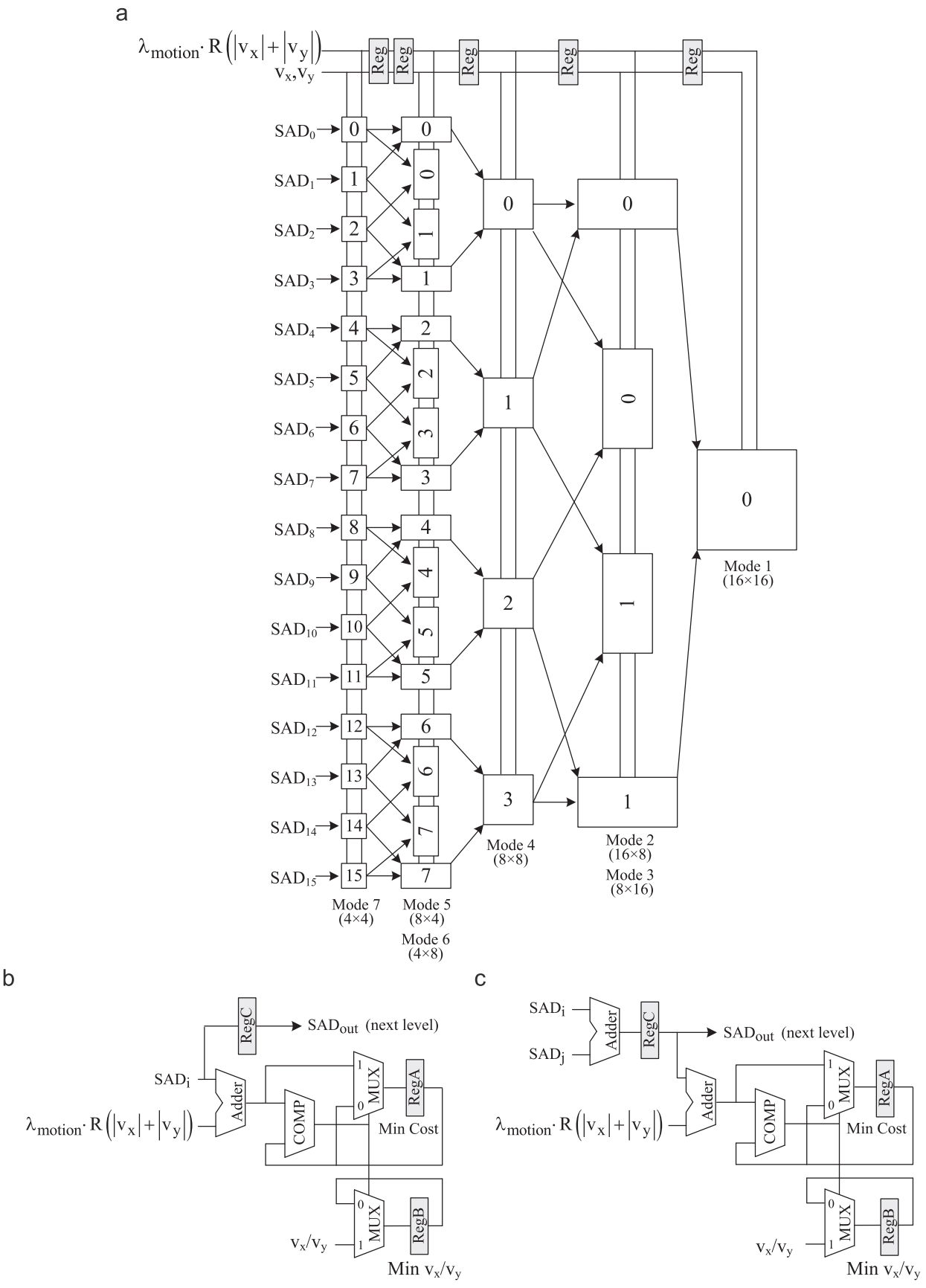

Fig. 12. Motion estimation module: (a) architecture, (b) structure of Mode 7 subblocks and (c) structure of the other modes.

Table 2
Relationship between λ_{motion} and QP.

QP	λ_{motion}	QP	λ_{motion}	QP	λ_{motion}	QP	λ_{motion}
0	1	13	1	26	5	39	23
1	1	14	1	27	6	40	25
2	1	15	1	28	6	41	29
3	1	16	2	29	7	42	32
4	1	17	2	30	8	43	36
5	1	18	2	31	9	44	40
6	1	19	2	32	10	45	45
7	1	20	3	33	11	46	51
8	1	21	3	34	13	47	57
9	1	22	3	35	14	48	64
10	1	23	4	36	16	49	72
11	1	24	4	37	18	50	81
12	1	25	4	38	20	51	91

H.263. Table 2 shows the values of λ_{motion} derived from the QP parameter used in the JM to minimize the Lagragian Cost.

$R(|v_x|+|v_y|)$ represents an estimation of the bits used to encode the motion information and they are usually obtained from a look-up table. In JM, this function R is defined as

$$R(|v_x|+|v_y|) = \text{mvbits}[|v_x|]+\text{mvbits}[|v_y|] \quad (6)$$

where mvbits can be expressed as

$$\begin{aligned}&\text{mvbits}[|v_x|] = 2|v_x|+1\\ &\text{mvbits}[|v_y|] = 2|v_y|+1\end{aligned} \quad (7)$$

Indeed, $\lambda_{motion}R(|v_x|+|v_y|)$ is equivalent to

$$\begin{aligned}\lambda_{motion}R(|v_x|+|v_y|) &= 2\lambda_{motion}(|v_x|+|v_y|+1)\\ &= (\lambda_{motion}(|v_x|+|v_y|+1)) << 1\end{aligned} \quad (8)$$

Fig. 13 shows the implementation of Eq. (8) where the modulus is computed as follows:

$$|v_x| = \begin{cases} v_x & \text{if } v_x \geq 0 \\ \overline{v}_x+1 & \text{if } v_x < 0 \end{cases} \quad \text{and} \quad |v_y| = \begin{cases} v_y & \text{if } v_y \geq 0 \\ \overline{v}_y+1 & \text{if } v_y < 0 \end{cases} \quad (9)$$

In this circuit, first v_x and v_y (or their complements depending on the sign) are added. The term "+1" in Eqs. (8) and (9) are added up in a single adder. In this case, the operation $\text{sign}(v_x)+\text{sign}(v_y)+1$ gives the result 1, 2 or 3, and this number is then added in a second adder to obtain the final value of $|v_x|+|v_y|+1$ which is stored in a register. In parallel, λ_{motion} is generated by a look-up table from the QP parameter to implement the relationship shown in Table 2. Finally, the multiplication and the subsequent shifter operation compute the $\lambda_{motion}R(|v_x|+|v_y|)$.

3.4. Motion decision

In the inter motion estimation of the H.264/AVC, a 16×16 MB is split up into subblock partitions of varying size according to a two-level hierarchy. The first level includes modes of 16×16, 16×8, 8×16, while in the second level every four 8×8 subblocks include modes of 8×8, 8×4, 4×8, and 4×4 (see Fig. 2). The mode decision module selects the best mode by comparing the total

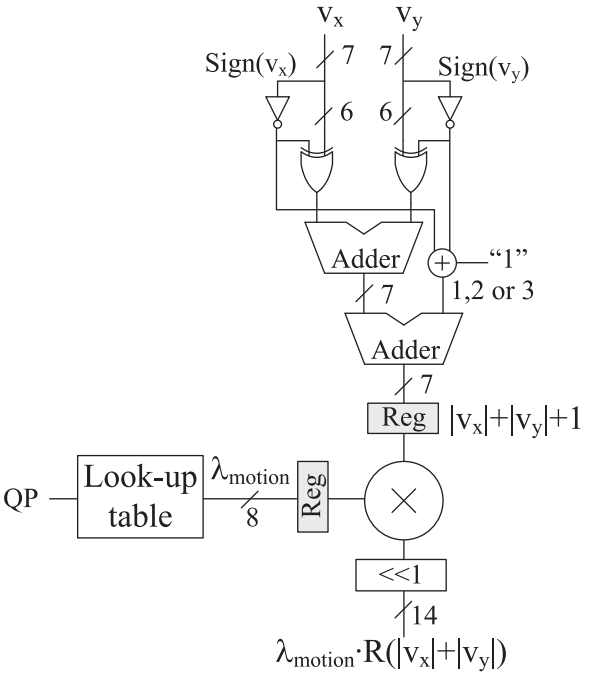

Fig. 13. Computing of $\lambda_{motion}R(|v_x|+|v_y|)$.

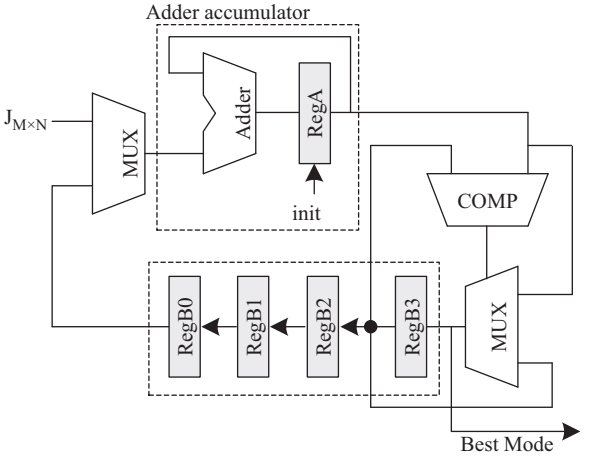

Fig. 14. Motion decision module.

minimum Lagrangian cost of all subblocks belonging to a mode once all 41 subblocks have been processed in the motion estimation module. Fig. 14 shows the proposed architecture of this module. As it is active for a short-time period to perform a specific operation, a serial architecture was chosen as the best option to save the area. Although it takes more clock cycles in comparison with a parallel architecture, this is not significant in the whole latency of the IME processing. In this circuit, the adder accumulator computes the minimum cost of all subblocks belonging to a mode, the registers RegB0 to RegB3 temporally store the best mode and minimum cost, and the comparator updates the new best mode and minimum cost in RegB3 by making a comparison between the current best mode stored in RegB3 and the candidate from RegA.

The Lagrangian cost $J_{M\times N}$ is serially input from the lowest subblocks to the highest ones and, after 65 clock cycles, the best mode for the MB is obtained. Initially, RegA is set to null and RegB0 to RegB3 are set to a maximum value. On computing a mode, the signal init resets RegA to null and a new mode is processed. The mode decision module consists of the following steps according to the level of hierarchy:

- **Second level**. Processing of the modes 4×4, 4×8, 8×4 and 8×8 which is repeated for every four $8\times8(i)$ subblocks (i=0, 1, 2 and 3) belonging to a MB. Afterwards RegB0 stores the best mode and minimum cost for $8\times8(0)$, RegB1 for $8\times8(1)$, RegB2 for $8\times8(2)$ and RegB3 for $8\times8(3)$. The scheduling of this level is:
  - *Step* 1: The four 4×4 subblocks belonging to an $8\times8(i)$ are serially input and accumulated in RegA. After 4 clock cycles, RegB3 is initialized to the result $J_{4\times4}(0)+J_{4\times4}(1)+J_{4\times4}(3)+J_{4\times4}(4)$.
  - *Step* 2: Two 4×8 subblocks belonging to $8\times8(i)$ are serially input. After 2 clock cycles, the data in the RegA, $J_{4\times8}(0)+J_{4\times8}(1)$, is compared with that stored in RegB3 and the minimum value is considered as the best mode and the minimum cost is put into RegB3.
  - *Step* 3: Two 8×4 subblocks belonging to $8\times8(i)$ are serially input. After 2 clock cycles, the data in RegA, $J_{8\times4}(0)+J_{8\times4}(1)$, is compared with that stored in RegB3 and the minimum is considered as the best mode and put into RegB3.
  - *Step* 4: The $J_{8\times8}$ is directly passed to RegA and compared with that stored in the RegB3 to select the final best mode of this $8\times8(i)$ subblock.

  On finishing Step 4, a shifter operation is performed which passes the data from RegB3 to RegB2 to release RegB3 for processing the next 8×8 subblock. This level takes 52 clock cycles and, as a result, RegB0 to RegB3 store the best mode and minimum cost for each four 8×8 subblocks.
- **First level**. Processing of modes 8×8, 8×16, 16×8 and 16×16.
  - *Step* 5: Initially, the minimum costs in the registers RegB0 to RegB3 are added and stored in RegB3. It takes 4 clock cycles for computing the operation RegB0+RegB1+RegB2+RegB3.
  - *Step* 6: Two 8×16 subblocks are added and the minimum cost is compared with that in RegB3, either updating or not this register. It takes 2 clock cycles.
  - *Step* 7: Two 16×8 subblocks are added and the minimum cost is compared with that in RegB3, and this register is updated or not. It takes 2 clock cycles.
  - *Step* 8: The minimum cost of the 16×16 subblock is compared with that in RegB3 to make the decision on which is the best mode in the MB. It takes 1 clock cycle.

As a result, the mode decision module generates the best mode and its MVs which are stored in RAM3. These MVs will be used in the MV prediction to generate the prediction vector $\mathbf{p}_{16\times16}$ corresponding to the next MB to be processed.

4. ASIC implementation and comparisons

A prototype of the IME processor chip based on the architecture in Fig. 7 has been designed using standard cells in a semicustom methodology. Initially, the processor was described in Verilog. The test bench was made by simulating the design with NC-VHDL from Cadence® and comparing the results obtained with those provided by the rewritten JM reference software to simulate the proposed Lagragian cost for different input samples and values of QP. This processor was synthesized with Synopsys® design compiler in Faraday Technology Corporation UMC 0.18 μm. The layout shown in Fig. 15 was generated using the Cadence Encounter P&R tool resulting in a circuit with a total 32.3 k gates and 6 RAMs (4 double port of 400×32, 1 single port of 64×32 and 1 single port of 180×32). The total size is 4.5 mm^2 (core 3.5 mm^2) and it has 116 I/O Pads (36 inputs, 46 outputs and 34 power supply pads). In typical working conditions (1.8 V, 25 °C), the maximum estimated operating frequency is about 300 MHz and the power consumption is 115 mW (SRAM included).

In H.264, the inter-prediction module is one of the most significant parts that affect overall computing performance. In real-time HDTV applications (1920×1088@30 fps) the work of processing all 41 subblocks belonging to a 16×16 MB should take 1225 clock cycles at a clock frequency of 300 MHz, which is available for most of current technologies; this should take 2770 clock cycles for 1280×720@30 fps and 7400 clock cycles for 720×480@30 fps. The proposed IME processor enables the search range to be selected from among the following options 8×8, 16×16, 32×32 and 64×64 pixels, although other geometries are available. Table 3 depicts the number of clock cycles needed to process different search ranges.

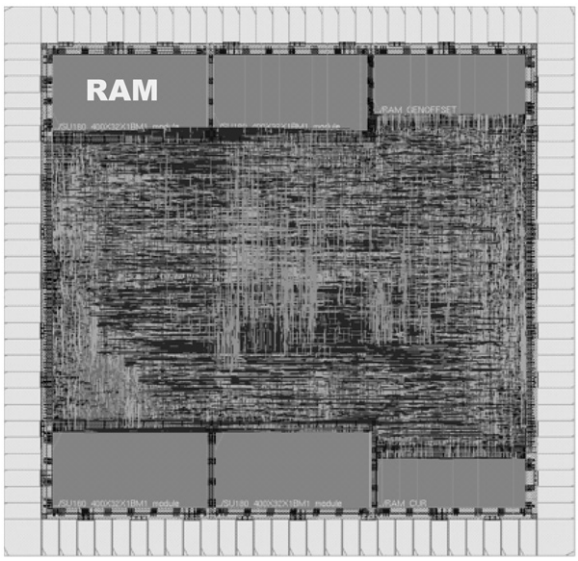

Fig. 15. Layout of the IME processor chip.

Table 3
Number of clock cycles for processing a MB with different search ranges.

Search range	Clock cycles
64×64	4375
32×32	1207
16×16	391
8×8	165

According to these results, this processor is able to process one MB in a search range of up to 32×32 for HDTV video and any range for lower standard resolutions. However, the motion estimation module takes most of the clock cycles defined as $(2p+2) \times (2p+1)+16$, where p is the dimension of the search range $2p \times 2p$ $(-p, p)$. However, to achieve 300 MHz as an operating frequency, the critical circuit paths have been carefully analyzed and balanced at the synthesis stage by means of pipeline levels without introducing an excessive latency. The critical path has been limited to only 2 adders and this has been used as a reference during the synthesis process. Moreover, a combination of a "clock gating" strategy and a balanced clock tree distribution has been fundamental to minimize the clock skew and to ensure the clock specifications.

Table 4 shows the characteristics and the performance of previously published ASIC processors for comparison purposes only, which are based on the full-search algorithm and implemented in a similar technology. The 1-D architecture presented in [13] is able to process the 41 MV subblocks in the same number of clock cycles. In [14], the block-matching is carried out by 16 cascading 1-D systolic arrays with local RAM to attain low latency, low power and high throughput. Another 1D architecture with simple control logic, regular workflow, a reduced number of PEs and one single-port SRAM for storing the search area data is presented in [15]. A simple and regular 2D datapath with 256 PEs is proposed in [16,17]. Ref. [18] presents an analysis of various dataflows and their impact in hardware architectures. As a result, a design based on a modified algorithm is proposed for Lagrangian mode decision making and hardware cost reduction by pixel truncation capable of processing 1280×720@30 fps. The IME processor architecture in [19] is based on a 2-D systolic processor array and minimizes the off-chip memory bandwidth using local memories to achieve the highest level of on-chip data reuse. In order to also obtain high data reuse, the architecture presented in [20] makes a three scan direction search through a reconfigurable 2D computing array and 16 local SRAMs. The design proposed in [21] uses a simplified prediction motion vector combined with an early termination motion estimation to reduce the computation complexity of the IME. This design includes SAD and Lagrangian cost in a 2D systolic array configuration resulting in a chip with a gate count of 191 k including memory modules. The bit-serial architecture in [22] uses a pipeline design, a reduced number of PEs and a pixel truncation technique to obtain a high clock frequency and low area cost but with a latency of 26 624 clock cycles. In [27], different memory-efficient, parallel 2-D architectures based on $16 \times 16 \times 16$ PEs are analyzed. In Table 4, the proposed design (b) is depicted,

Table 4
Comparison of the proposed processor with other IME designs.

Ref.	Year	Searching range	Cost	PE	Latency	Tech (μm)	Gate count	RAM	Freq. (MHz)	Power	Video
1D											
[13]	2004	16×16	SAD	16	4096	TSMC 0.13	61 k	NA	294	23.76 mW@QCIF@30fps	720×576@45fps
[14]	2004	16×16	SAD	256	1024	UMC 0.18	597 k	NA	200	503 mW@HDTV@60fps	720×576@60fps, 2 ref. frames
[15]	2006	16×16	SAD	16	4096	TSMC 0.18	51.7 k	7.9 kB	266	131 mW@266MHz	352×288@25fps
2D systolic array											
[16]	2005	16×16	SAD	256	1024	TSMC 0.18	154 k	60 kB	100	NA	–
		32×32			4097						704×576@15fps
[17]	2005	65×65	SAD	256	5216	0.18	260	63.5 kB	260	NA	720×576@30fps
[18]	2006	128×64	SAD+MVCOST	8192	NA	UMC 0.18	330 k	208 kB	108	NA	1280×720@30fps
											5-bit pixel truncation
[19]	2007	65×33	SAD	256	NA	0.18	168k	190kB	216	NA	720×576@30fps, 2 ref. frames
[20]	2008	33×33	SAD	256	1129	TSMC 0.18	160 k	26.6 kB	200	423mW@200MHz	1280×720@45fps
[21]	2008	48×32	SAD+MVCOST	256	1614	TSMC 0.13	191 k	NA	200	NA	1280×720@30fps
[22]	2009	32×32	SAD	16	26 624	0.18	44 k	NA	440	NA	352×288@30fps
											3-bit pixel truncation
[27]	2009	128×32	SAD	4096	512	TSMC 0.18	1511 k	12.9 kB	130	NA	1920×1088 @ 30fps
											½ down sampling and 5-bit pixel truncation
[39]	2009	16×16	SAD	52	NA	TSMC 0.18	113 k	51 kB	296	NA	1920×1088 @ 34fps
Ours	2010	32×32	SAD+MVCOST	64	1207	TSMC 0.18	32.3 k	59 kB	300	115 mW@300MHz	1920×1088 @ 30fps
		64×64			4375						Best mode and best MVs
											720×576@25fps

with which a very low latency is obtained with a high area cost. Architecture with throughput of 72.52 M samples per second that can process 34 1080HD frames per second is presented in [39]. The processor chip proposed in this paper uses a configurable 2D systolic array, modified Lagrangian MV cost and a motion decision module to compute the best mode and best MVs of a MB having a processing capacity for HDTV with a search range of 32×32. This processor has a quite reduced area of only 32 k gates and 59 kB on-chip RAM memory, less than other proposed designs operating at a frequency of 300 MHz, which is achieved by introducing balanced pipeline stages and carrying out a careful synthesis process. This speed is the fastest in comparison with the designs shown in Table 4, except for implementation [22] where a bit-serial architecture is used.

5. Conclusions

In this paper, we propose a high-performance VLSI processor chip for IME in H.264/AVC based on the full-search block matching algorithm (FSBMA) with enough processing capacity for 1080HD real-time video streaming with a search range of 32×32. The proposed design benefits greatly from a configurable 2D systolic array to obtain a high data reuse of the search area. It supports a three-direction scan format, a computing array of 64 PEs and a modified Lagrangian cost as matching criterion to find the best 41 variable-size blocks. It uses a tree pipeline parallel architecture, and a serial data flow mode-decision module to find the best mode and best MVs. This processor was designed with only 32.3 k gates and 4.4kBytes of RAM in standard UMC 0.18 μm technology at an operating frequency of 300 MHz. Compared with previous works, it presents a high speed and low area cost architecture suitable for H.264 encoders over a wide range of video applications.

Acknowledgment

We wish to acknowledge the financial help from the Spanish Ministry of Education and Science through TEC2006-12438/TCM.

Appendix A

List of acronyms used throughout the paper

BR or Bit-Rate
FSBMA or Full-Search Block Matching Algorithm
FME or Fractional Motion Estimation
FPS or Frames per Second
IME or Integer Motion Estimation
ME or Motion Estimation
MV or Motion Vector
MB or 16×16 Macroblock
PE or Processing Element
RDO or Rate-Distortion Optimization
PSNR or Peak-Signal-to-Noise-Ratio
VBSME or Variable Block-Size Motion Estimation
VLSI or Very Large Scale Integration

References

[1] ITU-T Rec., H.264/ISO/IEC 11496-10, "Advanced Video Coding," Final Committee , Document JVTG050, 2003.
[2] T. Wiegand, G.J. Sullivan, G. Bjontegaard, A. Luthra, Overview of H.264/AVC video coding standard, IEEE Transactions on Circuits and Systems for Video Technology 13 (7) (2003) 560–576.
[3] K. Sühring, H.264/AVC Software Coordination, 〈http://iphome.hhi.de/suehring/tml/〉, Fraunhofer Institute for Telecommunications, Heinrich Hertz Institute, Image Processing Research Department, Berlin, Germany.
[4] T.C. Chen, S.Y. Chien, Y.W. Huang, C.H. Tsai, C.Y. Chen, T.W. Chen, L.G. Chen, Analysis and architecture design of an HDTV720p 30 frames/s H.264/AVC encoder, IEEE Transactions on Circuits and Systems for Video Technology 16 (6) (2006) 673–688.
[5] T.C. Chen, Y.H. Chen, S.F. Tsai, S.Y. Chien, L.G. Chen, Fast algorithm and architecture design of low-power integer motion estimation for H.264/AVC, IEEE Transactions on Circuits and Systems for Video Technology 17 (5) (2007) 568–577.
[6] Z. Liu, Y. Song, M. Shao, S. Li, L. Li, S. Ishiwata, M. Nakagawa, S. Goto, T. Ikenaga, HDTV1080p H.264/AVC encoder chip design and performance analysis, IEEE Transactions of Solid-State Circuits 44 (2) (2009) 594–608.
[7] Y.K. Lin, C.C. Lin, T.Y. Kuo, T.S. Chang, A hardware-efficient H.264/AVC motion-estimation design for high-definition video, IEEE Transactions on Circuits and Systems-I 55 (6) (2008) 1526–1535.
[8] Y.W. Huang, C.Y. Chen, C.H. Tsai, C.F. Shen, L.G. Chen, Survey on block matching motion estimation algorithms and architectures with new results, Journal of VLSI Signal processing 42 (2006) 297–320.
[9] J.C. Tuan, T.S. Chang, C.W. Jen, On the data reuse and memory bandwidth analysis for full-search block-matching VLSI architecture, IEEE Transactions on Circuits and Systems for Video Technology 12 (1) (2002) 61–72.
[10] T. Komarek, P. Pirsh, Array architectures for block matching algorithms, IEEE Transactions on Circuits and Systems 36 (2) (1989) 1301–1308.
[11] L. de Vos, M. Stegherr, Parametrizable VLSI architectures for the full-search block-matching algorithm, IEEE Transactions on Circuits and Systems 36 (10) (1989) 1309–1316.
[12] J.F. Lopez, P. Cortes, S. Lopez, R. Sarmiento, Design of a 270 MHz/340 mW processing element for high performance motion estimation systems application, Microelectronic Journal 33 (2002) 1123–1134.
[13] S.Y. Yap, J.V. McCanny, A VLSI architecture for variable block size video motion estimation, IEEE Transactions on Circuits and Systems 36 (2) (2004) 1301–1308.
[14] C.M. Ou, C.F. Le, W.J. Hwang, An efficient VLSI architecture for H.264 variable block size motion estimation, IEEE Transactions on Consumer Electronics 51 (4) (2005) 1291–1299.
[15] Y. Song, Z. Liu, T. Ikenaga, S. Goto, A VLSI architecture for variable block size motion estimation in H.264/AVC with low cost memory organization, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences E89-A (12) (2006 : [15].) 3594–3601.
[16] M. Kim, I. Hwang, S.I. Chae, A fast VLSI architecture for full-search variable block size motion estimation in MPEG-4 AVC/H.264, in: Proceedings of the Asp-DAC 2005, vol. 1, January 2005, pp 631–634.
[17] L. Deng, W. Gao, M.Z. Hu, Z.Z. Ji, An efficient hardware implementation for motion estimation of AVC standard,, IEEE Transactions on Consumer Electronics 51 (4) (2005) 1360–1366.
[18] T.C. Chen, S.Y. Chien, Y.W. Huang, C.H. Tsai, T.C. Chen, T.C. Wang, L.G. Chen, Analysis and architecture design of variable block-size motion estimation for H.264/AVC, IEEE Transactions on Circuits and Systems-I 53 (2) (2006) 578–593.
[19] D.X. Li, W. Zheng, M. Zhang, Architecture design for H.264/AVC integer motion estimation with minimum memory bandwidth, IEEE Transactions on Consumer Electronics 53 (3) (2007) 1053–1060.
[20] C. Wei, H. Hui, T. Jiarong, L. Jinmei, M. Hao, A high-performance reconfigurable VLSI architecture for BSME in H.264, IEEE Transactions on Consumer Electronics 4 (3) (2008) 1338–1344.
[21] A.C. Tsai, K.I. Lee, J.F. Wang, J.F. Yang, VLSI architecture designs for effective H.264/AVC variable block-size motion estimation, in: Proceedings of the International Conference on Audio, Language and Image Processing, July 2008, pp. 413–417.

[22] M.R.H. Fatemi, H.F. Ates, R. Salleh, A bit-serial sum of absolute difference accelerator for variable block size motion estimation of H.264, in: Proceedings of the Conference on Innovative in Intelligent Systems and Industrial Applications, pp. 1–4, 2009.
[23] J.H. Lee, N.S. Lee, Variable block size motion estimation algorithm and its hardware architecture for H.264/AVC, in: Proceedings of the IEEE International Symposium on Circuits and Systems, 2004, pp. 741–744.
[24] L. Zhang, W. Gao, Improved FFSBM algorithm and its VLSI architecture for variable block size motion estimation on H.264, in: Proceedings of 2005 Intelligent Signal Processing and Communication Systems, December 2005, pp. 445–448.
[25] Y.L. Xi, C.Y. Hao, Y.Y. Fan, H.Q. Hu, A fast block-matching algorithm based on adaptative search area and its VLSI architecture for H.264/AVC, Signal Processing: Image Communication 21 (2006) 626–646.
[26] H. Chang, S. Kim, S. Lee, K. Cho, High-performance architecture of H.264 integer-pixel motion estimation IP for real-time 1080HD video CODEC, in: Proceedings of the 23rd IEEE International SOC Conference, September 2010, pp. 419–422.
[27] C.Y. Kao, Y.L. Lin, A memory-efficient and highly parallel architecture for variable block size integer motion estimation in H.264/AVC, IEEE Transactions on Circuits and Systems for Video Technology 18 (6) (2010) 866–874.
[28] P. Kuhn, Algorithms, Complexity a and VLSI Architectures for MPEG-4 Motion Estimation, Kluwer Academic Publishers, Boston, 1999.
[29] M. Manikandan, P. Vijayakumar, N. Ramadass, Motion estimation method for video compression—an overview, in: Proceedings of the IFIP International Conference on Wireless and Optical Communications Networks, August 2006, pp. 5.
[30] H.M. Jong, L.G. Chen, T.D. Chiueh, Parallel architecture for 3-step hierarchical search block-matching algorithm, IEEE Transactions on Circuits and Systems for Video Technology 4 (4) (1994) 407–416.
[31] C. Zhu, X. Lin, L.P. Chau, Hexagon-based search pattern for fast block motion estimation,, IEEE Transactions on Circuits and Systems for Video Technology 12 (5) (2002) 349–355.
[32] C. Zhu, K.K. Ma, A new diamond search algorithm, for fast block matching motion estimation,, IEEE Transactions on Image Processing 9 (2) (2000) 287–290.
[33] J. Vanne, K. Kuusilinna, T.D. Hämäläinen, A configurable motion estimation architecture for block-matching algorithms,, IEEE Transactions on Circuits and Systems for Video Technology 19 (4) (2009) 466–476.
[34] J.Q. Wang, D. Zhao, W. Gao, S. Ma, Low complexity RDO mode decision based on a fast coding-bits estimation model for H.264/AVC, in: Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), vol. 4, May 2005, pp. 3467–3470.
[35] J.L. Nunez-Yanez, V.A. Chouliaras, D. Alfonso, F.S. Rovati, Hardware assisted rate distortion optimization with embedded CABAC accelerator for the H.264 advanced video codec, IEEE Transactions on Consumer Electronics 52 (2) (2006) 590–597.
[36] R. Hashimoto, K. Kato, G. Fujita, T. Onoye, VLSI architecture of H.264 RD-based block size decision for 1080 HD, in: Proceedings of Picture Coding Symposium, ThPM4.9.1–ThPM4.9.4, November 2007.
[37] J. Vanne, E. Ahn, T.D. Hämäläinen, K. Kuusilinna, A high-performance sum of absolute difference implementation for motion estimation, IEEE Transactions on Circuits and Systems for Video Technology 16 (7) (2006) 876–883.
[38] G.J. Sullivan, Thomas Wiegand, Rate-distortion optimization for video compression, IEEE Signal Processing Magazine 15 (1998) 74–90.
[39] R. Porto, L. Agostini, S. Bampi, Hardware design of the H.264/AVC variable block size motion estimation for real-time 1080HD video encoding, in: Proceedings of the 2009 IEEE Computer Society Annual Symposium on VLSI, 2009, pp. 115–120.
[40] G. Bjontegaard, Calculation of average PSNR differences between RD curves, in: ITU-T SC16/Q6, 13th VCEG Meeting, Austin, Texas, USA, 2001, Doc. VCEG-M33. Available from 〈http://wftp3.itu.int/av-arch/video-site/0104_Aus/VCEG-M33.doc〉.