

# An efficient low-cost FPGA implementation of a configurable motion estimation for H.264 video coding

Wajdi Elhamzi · Julien Dubois · Johel Miteran ·  
Mohamed Atri

Received: 28 March 2012 / Accepted: 14 August 2012 / Published online: 5 September 2012  
© Springer-Verlag 2012

**Abstract** Despite the diversity of video compression standard, the motion estimation still remains a key process which is used in most of them. Moreover, the required coding performances (bit-rate, PSNR, image spatial resolution, etc.) depend obviously of the application, the environment and the network communication. The motion estimation can therefore be adapted to fit with these performances. Meanwhile, the real time encoding is required in many applications. To reach this goal, we propose in this paper a flexible hardware implementation of the motion estimator which enables the integer motion search algorithms to be modified and the fractional search as well as variable block size to be selected and adjusted. Hence, this novel architecture, especially designed for FPGA targets, proposes high-speed processing for a configuration which supports the variable size blocks and quarter-pel refinement, as described in H.264. The proposed low-cost architecture based on Virtex 6 FPGA can process integer motion estimation on 1080 HD video streams, respectively,

at 13 fps using full search strategy (108k Macroblocks/s) and up to 223 fps using diamond search (1.8M Macroblocks/s). Moreover subpel refinement in quarter-pel mode is performed at 232k Macroblocks/s.

**Keywords** Configurable motion estimation · H.264 · IME · FME · Video coding performances

## 1 Introduction

Video coding has been the subject of many research works in last decades. A large number of coding solutions have been described to fit with the diversity of the compression standards and the requested coding performances, which are correlated to the constraints defined by the user or fixed by the environment (i.e. networks used for data transmission and the target receiver setup). Consequently, a large number of video codec has been developed. Despite this diversity, some particular processing stages, such as motion estimation [1], are implemented in most of the proposed solutions. The motion estimation is well known to be the most computation-intensive stage of video coding process. Any improvement on this stage has therefore impact on the whole video codecs performances. From another point of view, the motion estimation configuration can be adjusted to fit the applications constraints (image spatial resolution, frame-rate, bit-rate, PSNR). For instance the motion estimation stage, described in the recent standards such as H.264 video or VP8, is highly efficient as well as highly sophisticated and complex. Meanwhile, different configurations can be defined to optimize coding performances. According to our analysis and as depicted in Fig. 1, the key features of motion estimation to be adjusted in current standard as H.264 are:

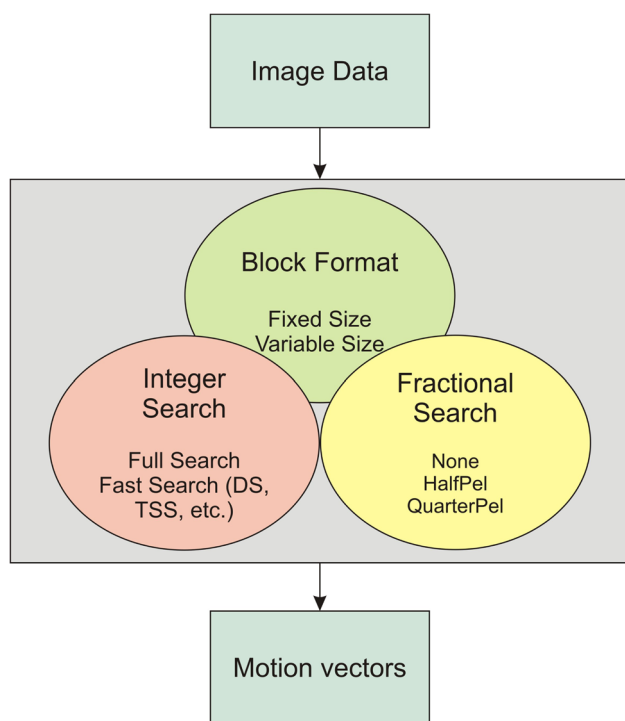
---

W. Elhamzi (✉) · J. Dubois · J. Miteran  
Laboratory Le2i, University of Burgundy,  
UMR CNRS 6063,  
21000 Dijon, France  
e-mail: Wajdi.Elhamzi@u-bourgogne.fr

J. Dubois  
e-mail: julien.dubois@u-bourgogne.fr

J. Miteran  
e-mail: johel.miteran@u-bourgogne.fr

M. Atri  
Laboratory of EμE-LAB-IT-06,  
Faculty of Sciences of Monastir,  
University of Monastir, Monastir, Tunisia  
e-mail: Mohamed.Atri@fsm.rnu.tn



**Fig. 1** H.264 motion estimation key features

- The format of input data (i.e. the size of the blocks to match),
- The integer search method,
- The optional fractional search.

Software solutions can easily support any configuration, nevertheless they may struggle to match the application's requirements. Indeed, for high-quality applications, the computational cost often exceeds the available resources of a standard computer. Meanwhile, it is still nowadays a challenge to define an efficient hardware accelerator which supports such flexibility as well as high coding performances. Therefore, we propose in this paper, a motion estimation accelerator, fully compatible with H.264, which supports different configurations especially modifications on the three key features previously identified.

The main contributions of this paper are (1) the definition of an architecture allowing flexibility, since the user can select the variable block size, the type of search method (full or fast search), the type of the accuracy (integer or fractional search), (2) some new particular hardware optimizations of the architecture of the motion estimation allowing to improve the performance (computing time or hardware resources) compared to the state of the art. Indeed, the motion estimator's architecture has been designed and optimized to propose an efficient FPGA implementation in respect with the complexity and the regularity of the integer search and the fractional search.

The paper is structured as follows. In the Sect. 2, we review the basic principles of the motion estimation. We study the impact of the search strategy, recalling and comparing several integer search algorithms. The impact of the Variable Block Size Motion Estimation (VBSME) and the fractional search on the coding performances (PSNR, processing-time) are discussed. In Sect. 3, we propose an analysis of the integer motion estimation (IME). The common parts of the studied algorithms are then bringing out to prove that a generic structure can be proposed. The hardware implementation results, in terms of required hardware resources and processing time are finally presented and discussed in Sect. 4.

## 2 Motion estimation technique

The motion estimation (ME) is an effective stage to detect temporal redundancies between successive frames in a video sequence. Therefore, it has become a crucial part of many video compression standards. The motion estimation aims to predict, as accurately as possible, the next frame from the current frame. The frame is split into fixed size macro-blocks, currently  $16 \times 16$  pixels. The prediction is processed for each macro-block of the current frame. As the motion estimation algorithm is not fixed by the video standard, many solutions have been proposed. The most popular is the Block-Matching Algorithm (BMA). The basic idea is to localize a reference block within the search area in the previous frame. A matching criterion is used to estimate similarities between two blocks. The applied BMA is performed using the sum of absolute differences (SAD) matching criterion given by:

$$\text{SAD}(u, v) = \sum_{i=1}^{i=K} \sum_{j=1}^{j=K} |S(i+u, j+v) - R(i, j)|$$

$$\text{SAD}_{\min} = \min(\text{SAD}(u, v))$$

where  $K$  is the block width and height,  $R(i, j)$  is a pixel in current (reference) frame and  $S(i+u, j+v)$  is a pixel in previous frame with an offset of  $(u, v)$ . This approach is known as the integer motion search as only integer displacements of the reference macroblock into the search window are performed.

To improve the precision of the estimation of the motion vector, some sub-pixel refinement can be processed. The Fractional Motion Estimation (FME) is usually done for half-pel and quarter-pel accuracy [2].

The VBSME [3] is another major refinement which is included in recent standards. This approach is based on the BMA, combined with a dynamic selection of the blocks size. The macro-block is split into smaller blocks and the estimation is performed on each sub-blocks. The VBS

motion estimation can be processed at both integer and fractional levels. The motion estimation performances are, therefore, highly correlated not only to the selected integer search algorithm but also to the optional refinement stages VBSME and FME. Therefore, the following sections discuss the impact of the integer search algorithms as well as the VBSME and the FME algorithms on the quality of image and other encoder’s requirements.

### 2.1 Impact of the search strategy—related work

Many ME algorithms have been described in the literature. The most accurate strategy is the Full Search (FS) algorithm, which by exhaustively comparing all positions in the search window, gives the most accurate motion vector which causes SAD to be minimum. On the other hand, fast but sub-optimal algorithms compute the best matching candidate by guiding the search procedure using predefined search patterns. For instance, square-shaped or hexagon-shaped or diamond-shaped search patterns with different sizes are employed in several fast motion algorithms such as, Three-Step-Search (TSS) [4], the New Three Step Search (NTSS) [5], the Four Step Search (4SS) [6], the Hexagon-Based Search (HEXBS) [7], the Diamond Search (DS) [8], the Cross-Diamond Search (CDS) [9], and the Block-Based Gradient Descent Search (BBGDS) [10] algorithms. Referring to these previous works, it is possible to note that these algorithms performed well in relatively small search range and low-resolution video sequences.

Using these algorithms impact the global coding performances (i.e. bit-rate, image quality and processing time). Improving the bit-rate or the quality of image is achieved by finding the best possible motion vectors, which means motion vectors that generate the smallest residual difference during the motion compensation. Meanwhile, reducing the total search time is achieved by selecting the proper fast motion estimation, which consists to reduce the number of points per block (NSP) to be checked. Nevertheless, the bit-rate and the image quality can be decreased compared with a FS approach. For instance, the mentioned fast search algorithms are evaluated in [11] regarding the output PSNR. For low and medium motion activity video sequences (Carphone, Foreman, and Mobile), the degradation of PSNR is slightly or negligible but the speed-up is much improved. The situation is changed for the high-motion activity video sequences (Tennis Table, Football): the PSNR significantly decreases and the degradation of image quality is then visible. Despite, DS and BBGDS algorithms provide better PSNR performances than other fast algorithms while maintaining nearly the same search speed for the sequences Table tennis and Football.

An enhanced efficient DS algorithm, named Modified Diamond Search (MDS), is proposed and compared with

other fast approach and FS method in [12]. The proposed method as well as the other fast search methods DS, 4SS and N3SS achieves significant speed-up compared to FS. Hence the processing time, respectively, decreases of 99, 94, 73 and 65 % for high motion video sequence (Football). A negligible degradation in both PSNR and bit-rate is observed. The low complexity of DS approach family induces that this kind of algorithms can be considered for hardware implementation. The FS algorithm is suitable to high-speed motion and/or high texture variation.

### 2.2 Impact of VBSME and FME—related work

H.264 introduces two new features to ME, the VBSME and the sub-pixel accuracy motion estimation. The VBSME is carried out in two phases: integer motion estimation (IME) and fractional motion estimation (FME). In H.264, VP8 and other video codec, a  $16 \times 16$  sized macro-block can be further partitioned into  $16 \times 8$ ,  $8 \times 16$ ,  $8 \times 8$ ,  $8 \times 4$ ,  $4 \times 8$  and  $4 \times 4$  sub-blocks as shown in Fig. 2.

When all sub-blocks are in uniform motion, all sub-block motion vectors will be the same as the motion vector for the entire macro-block. Nevertheless, when sub-blocks partitions are moving in different directions, sub-block motion vectors can differ significantly from each other and from the motion vector of the macro-block. Consequently, the ME unit must be able to generate a separate motion vector for each of the sub-blocks.

The advantages of a large block size are (1) simplicity and (2) the limited number of vectors that must be encoded and transmitted. However, in areas of complex spatial structures and motion, better performance can be achieved with the smaller block size. Usually, the motion of blocks does not match exactly in the integer positions. So, to find best matches, fractional position accuracy is used. If the best motion vector is a fractional position, an interpolation is needed to predict the current block. According to [4] and [13], FME upgrades rate distortion efficiency by +4 dB in peak signal-to-noise ratio (PSNR) and requests 45 % of the inter-prediction processing time. In [14], four sequences with different characteristics are used for the experiment.

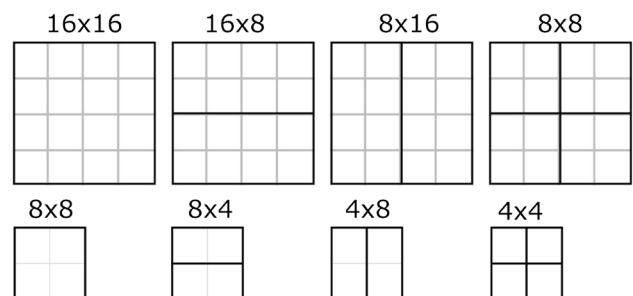


Fig. 2 Variable block size partitions

Foreman stands for medium motions, Soccer sequence for high motions. Mobile and Optis have complex textures. Clearly using half or quarter-pel increases image quality. The accuracy of motion compensation is in quarter-pel resolution for H.264/AVC, which can provide significantly better compression performance, especially for images with complex texture.

As shown in the state of the art analysis, the three key features which are the data block sizes (defined with VBSME), the IME strategy and the optional FME have high impact on the video codec's performances. Therefore, an efficient hardware implementation of a configurable motion estimator which supports modification on these three features can be considered as a significant contribution.

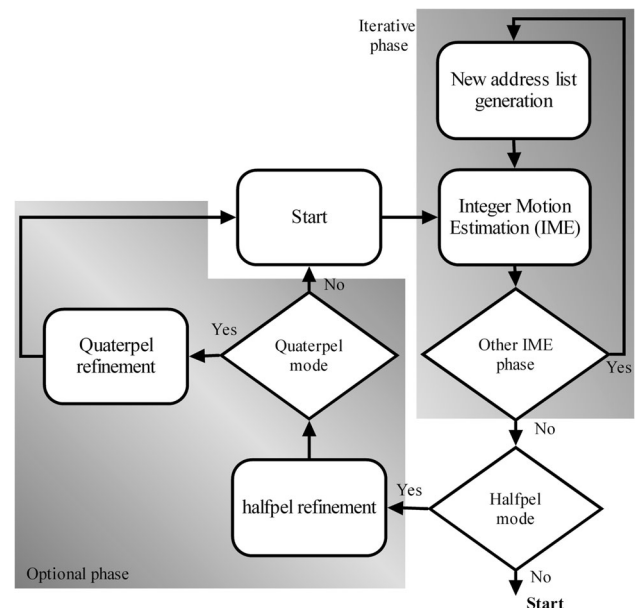
### 3 A flexible motion estimation architecture

#### 3.1 Overview of the proposed architecture

Using a full search strategy, the motion detection process is regular. All possible positions of the pattern in the search window are scanned contrary to fast search approach. All the search strategies are intended to converge to the right motion vector with a regular process and can eventually be initialized by previous information (such as the vectors previously computed). Therefore, using fast or reduced search strategies enable the number of matching to be reduced, decreasing the processing time. As mentioned previously, even if the number of matching is reduced, it is possible to achieve optimal coding results using appropriate (for the video application) reduced search algorithms.

According to our analysis, for a majority of IME algorithms, a list of matching positions is processed during each Integer Motion Search Phase (IMEP). As shown in Fig. 3, after the reception of the list of positions and the number of matching, each matching is processed until all positions have been considered. Finally, the best vector and, optionally, all the resulting vectors are available. All the resulting vectors may be used to generate the next set of matching positions. The iterative phase IMEP is depicted in Fig. 3. Hence, we propose an architecture which supports the iterative processing of lists of positions.

For all configurations, the address generation unit is described using a scheduler. This description could be regular and very simple for a Full Search algorithm or more complex and irregular for more complex algorithms. For instance, the Diamond Search has several phases of address generation. The full-search strategy is obviously supported in this scheme. Note that in this case, all possible addresses of the reference block in the search window are then transmitted to the operating part, and therefore a unique

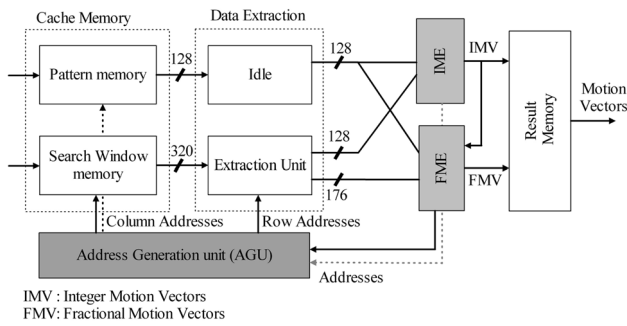


**Fig. 3** Proposed algorithm with iterative interger search and fractional search

phase is required to determine the best motion vector. These FS and DS methods are both implemented in our design. Other kind of fast search, based on the iterative principles [5, 7], may be eventually implemented instead of the DS mode.

Finally, depending on the user configuration, the optional fractional motion estimation (FME) can be performed with half-pel or quarter-pel accuracy. The half-pel stage is processed systematically before the quarter-pel refinement, to reduce the search area and therefore to reduce memory requirement. Indeed, the sub-pel refinement and especially the interpolation phase are costly in terms of hardware resources [4, 13, 14]. Note that using fast search strategy decreases significantly IME processing time. Consequently, FME becomes the slower stage of the motion estimation. Therefore, an optimized and efficient architecture should be proposed for the FME unit. The architecture proposed is depicted in Fig. 4. The addresses of all matching are provided by the external address generation unit to the cache memory unit and to the row extraction unit. As a trade-off between the efficiency and the required resources, one matching is processed in several stages. A full parallel approach would require a large cache memory, to avoid important bottlenecks on the external memory, and extremely large processing resources for real-time implementation. We propose a trade-off between the complexity and performances with a matching done row-by-row.

So as to guarantee the random access to any position of a search window, the search window pixels have to be accessible to the matching engine and need to be stored in



**Fig. 4** Top-level view of the proposed motion estimation architecture

the FPGA to reduce the number of access to the external memory, so as not to exceed the available bandwidth. The cache memory permits access of two rows, one extracted from the block and the corresponding one in the search window. The architecture allows to access any row in one clock cycle. Consecutive pattern matching evaluations do not need to be adjacent in terms of memory locations. The cache memory is obtained with dual-port memory blocks available into FPGA component. Our architecture enables an efficient random access, without any latency to be obtained. Moreover, any search-window width can be set according to the available processing.

The Address Generator Unit (AGU) is in charge of address generation for all configurations. The AGU allows selecting one row into the search window and one into the current block. The extraction unit enables the right amount of pixels to be selected into the search window row according to the selected position and the motion estimation to be processed. Hence, for IME, 16 pixels (128 bits) are systematically extracted from the  $16 \times 16$  macro-block and all possible sub-blocks can be processed in parallel. For FME, an interpolation phase is required, using a set of six-tap filters. Therefore, the region to be extracted is slightly larger than the block width. The pixel number extracted also depends on the selected mode ( $16 \times 16$ ,  $16 \times 8$ ,  $8 \times 8$ ,  $8 \times 4$ ,  $4 \times 4$ , etc.). For blocks width equal to 16, 8 and 4 pixels, respectively, 22, 14 and 10 pixels should be extracted.

### 3.2 Proposed integer motion estimator

The architecture of the Processing Unit is a key point of the integer motion estimation, in terms of hardware resources and processing time. Several architectures have been proposed in the literature, some implementing Fixed Block Size Motion Estimation (FBSME) based on the FS algorithm, and some implementing VBSME, as the Propagate Partial SAD [3, 15], the SAD Tree [16], and the Parallel Sub-Tree [17].

Since the regularity of data dependency of full search motion estimation, 1D and 2D systolic arrays are generally

used for efficient implementation of VBSME. One of the first 1D-systolic PEs-array implementations of VBSME was presented by Yap and McCanny [3], and later improved upon by Song et al. [15] and Fatemi et al. [18]. In [19], Lee uses 1-D dimensional array architecture with 64 PEs to process all 41 motion vectors within 1,027 or 4,099 clock cycles, respectively, for  $16 \times 16$  or  $32 \times 32$  search range. Two dimension array architectures [20, 21] have also been proposed for high-end application domains, such as HDTV. All these structures support only FS strategy. Other related works support DS strategy, but the originality of the unified PU architecture that we propose in the next section is to support several strategies and VBSME.

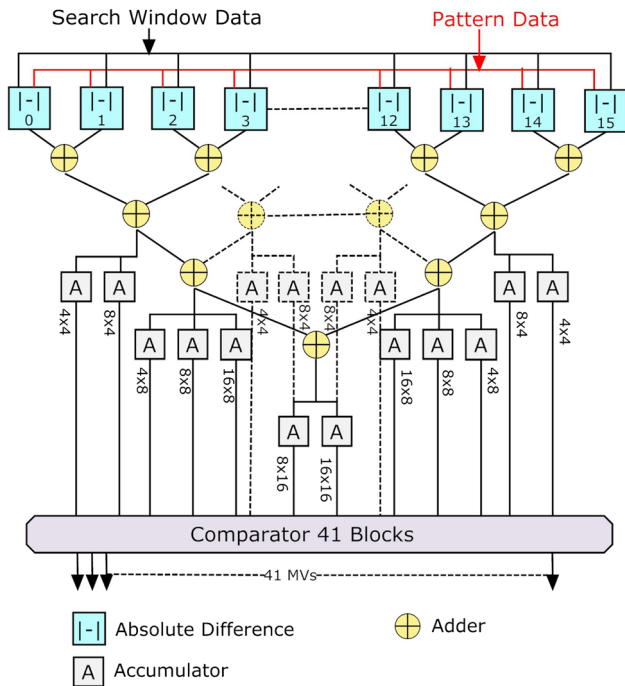
#### 3.2.1 PE unit architecture

The IME phase is highly regular. As shown in Fig. 4, the overall structure of the Processing Unit architecture is based on the Propagate Partial SAD architecture [3, 15, 18, 21]. Four kinds of operators are therefore required: absolute difference, adder, accumulator and comparators. The 16 differences are added with a six-stage pipelined structure. The 16 accumulators, which are included in this structure, enable all 40 sub-blocks defined in VBSME to be processed. Each matching is operated row-by-row. At each cycle, the SAD operation is performed on the two rows, one extracted from the reference macroblock and the corresponding row in the search window. The full macroblock matching is obtained by accumulation of each row comparison. The SAD results can be re-used and accumulated to compute values for several block sizes. For instance, the results of two  $4 \times 4$  sub-block computations can be combined to derive results for a  $4 \times 8$  or  $8 \times 4$  or  $4 \times 8$  computation, and so on. All SAD values are passed through accumulator operators (labelled A in Fig. 5). One accumulator can be used several times, for instance four times for a  $4 \times 4$  accumulator. A matching can be performed every 16 cycles. The 41 motion vectors are available at comparator output, 6 cycles after the transfer of the 16th macro-block's row. The comparator unit is constituted of 41 comparing units: one for each sub-block. The originalities of this architecture are (1) the parallel use of accumulators, allowing to save one clock cycle during the VBS process (2) several matching can be processed sequentially without any latency.

#### 3.2.2 User search impact: study case using FS and DS methods

The described structure enables a matching to be processed sequentially, row-by-row, in 16 cycles. This architecture is fixed for any search strategy without any processing time

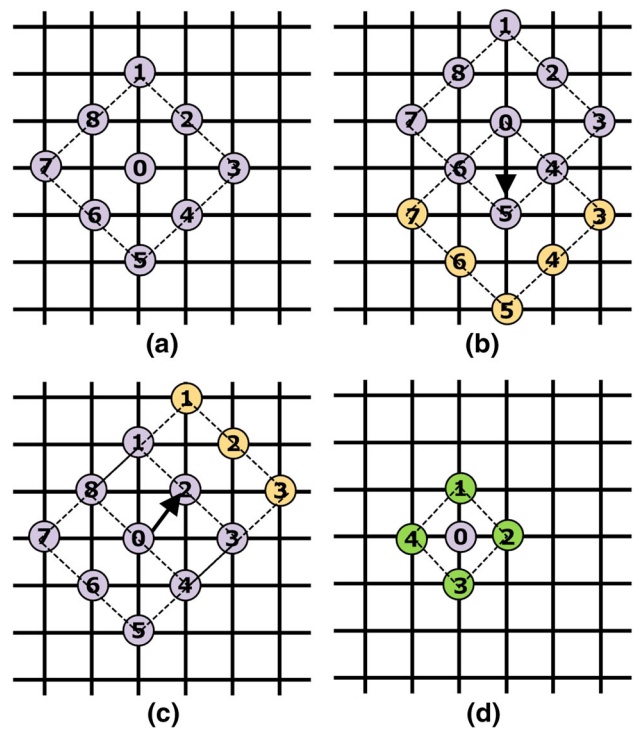




**Fig. 5** IME processing unit with VBSME supported

overhead. Indeed, two random matching can be checked without any latency. We have implemented two search strategies to check flexibility of the proposed architecture: Full Search (FS) and Diamond Search (DS). As described previously, an external module, called AG, is in charge of address generation. In our implementation, a state machine has been used to generate the list of addresses. The Diamond Search is a regular and not complex algorithm therefore the implementation is low cost in terms of hardware resources. Moreover, for each Diamond Search phase, this description enables all addresses (i.e. maximum nine addresses as shown in Fig. 6 to be generated in parallel at 438 MHz. Therefore, higher frequency can be achieved compared to a sequential micro-processor based solution (embedded in the FPGA, limited to 300 MHz for our selected component). A micro-processor represents a flexible solution as described in [22], nevertheless it only processes a maximum of one address per cycle. For FS method, all possible addresses should be considered and are generated sequentially. The set of addresses is predictable and depends on the search region size. For instance for  $N \times M$  search area and  $P \times P$  macroblock size, there are  $Nofm = (N - P + 1) \times (M - P + 1)$  possible matching. Therefore, 16  $Nofm$  addresses are required. The row data are transferred row-by-row to the proposed structure. As illustrated in Fig. 7, the data are transferred without interruption. The result is obtained six cycles after the last row transferred.

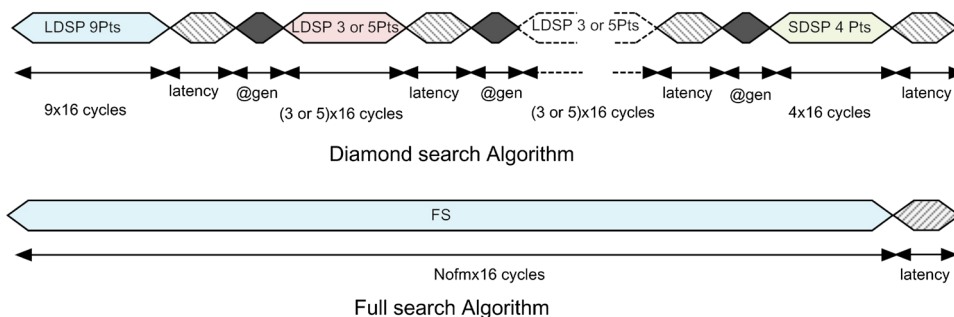
The Diamond search has several phases of address generation. At first, a Large Diamond Search Pattern



**Fig. 6** a DS pattern large, b DS along a corner point, c DS along an edge point, d DS pattern small

(LDSP) step is applied in this case and required nine matching as shown in Fig. 6a. LDSP is repeated until the step in which the minimum block distortion (MBD) occurs is at the center point. A Small Diamond Search Pattern (SDSP) step is then achieved. The number of matching in a LDSP depends on the best matching obtained in the previous LDSP. As presented in Fig. 6b and c, three and five matching are required when LDSP is, respectively, performed on an edge or corner point. The SDSP is processed with four matching. Pixels are diffused from the cache memory unit to the Raw Extraction unit. The SAD module starts the processing for each macro-block to return the best score associated with the address and its position into the diamond pattern. This position enables the generation of the new addresses for the next step. For instance, an example of LDSP phases and a final SDSP phase is presented in Fig. 6d. Contrary to the FS method, several sets of matching addresses are required. All matching of current phase should be performed to enable the next set of addresses to be processed: the position of best matching is obviously crucial. After a LDSP, the next set of addresses is fixed for DS method, and performed by the external AG unit in two cycles. The process row-by-row enables a low-cost architecture to be proposed. Note that for each matching 16 rows need to be transferred. Nevertheless, high speed performances can be achieved as all sub-blocks defined by VBSME method can be processed

**Fig. 7** Data transfer and task scheduling for Diamond Search and Full Search algorithms



simultaneously. Moreover, a set of matching can be processed without any latency despite the fact that the matching positions are not adjacent in terms of memory locations. Several sets of matching can be processed which enables multi-phase based fast search algorithms to be implemented.

### 3.3 Proposed fractional motion estimator

As discussed in Sect. 2.1, an optional step can be added to IME: the sub-pel refinement can be performed to increase coding performances. Using DS mode, high throughput (1800k macro-blocks/s) can be reached. As the DS output is the input of FME stage, and as the FME is a complex algorithm (all sub-blocks have to be refined), it is therefore crucial to optimise the FME implementation.

#### 3.3.1 FME architecture

After the best integer motion vector is estimated, the fractional motion estimation accuracy can start. The half-pel refinements of the surrounding eight half-search positions are computed, and then the quarter-pel refinements of eight quarter-search positions surrounding the best half-search position are computed. In the MPEG-4/AVC H.264 standard, the quarter-pel accuracy luminance picture is interpolated with two successive filtering operations. The half-pel refinement is more complex than the quarter-pel one and requires 6-tap separable FIR filters with coefficients [1, -5, 20, 20, -5, 1] instead of bilinear filters. As shown in Fig. 8a, each half-pel value is calculated from six adjacent pixels horizontally or vertically. The horizontal value  $h_{1,1}$  is computed from the six adjacent integer pixel samples located at horizontal direction according to the following equation :

$$h_{1,1} = i_{1,-1} - 5i_{1,0} + 20i_{1,1} + 20i_{1,2} - 5i_{1,3} + i_{1,4} \quad (1)$$

In a similar way, the vertical half-pel value  $v_{1,1}$  is performed using the six adjacent pixel values located in the vertical direction as:

$$v_{1,1} = i_{-1,1} - 5i_{0,1} + 20i_{1,1} + 20i_{2,1} - 5i_{3,1} + i_{4,1} \quad (2)$$

The diagonal half-pel value  $d_{1,1}$  is obtained from the six adjacent horizontal values  $h_i$  or alternatively, verticals values  $v_{i,j}$  according to:

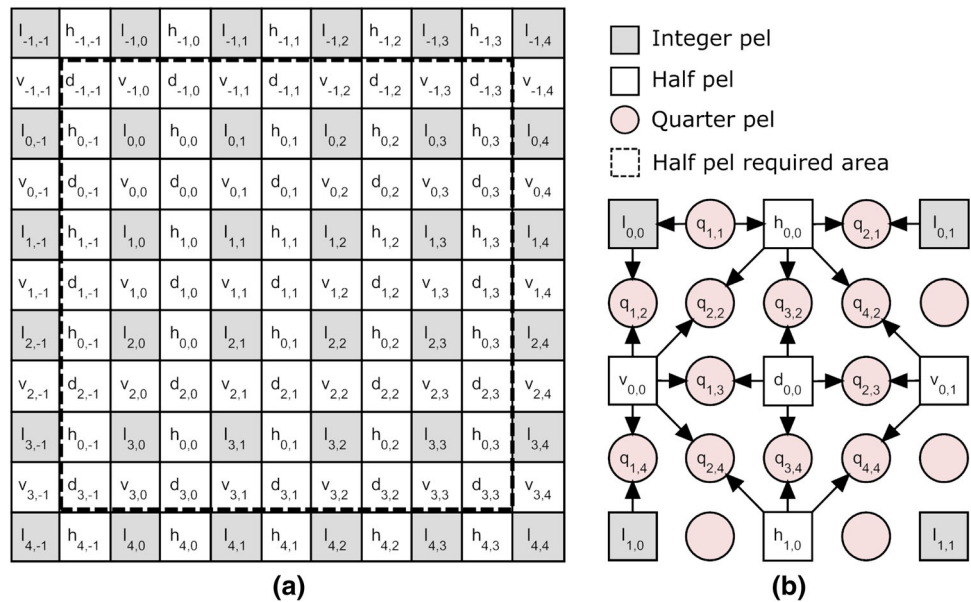
$$d_{1,1} = v_{1,-1} - 5v_{1,0} + 20v_{1,1} + 20v_{1,2} - 5v_{1,3} + v_{1,4} \quad (3)$$

$$d_{1,1} = h_{-1,1} - 5h_{0,1} + 20h_{1,1} + 20h_{2,1} - 5h_{3,1} + h_{4,1}$$

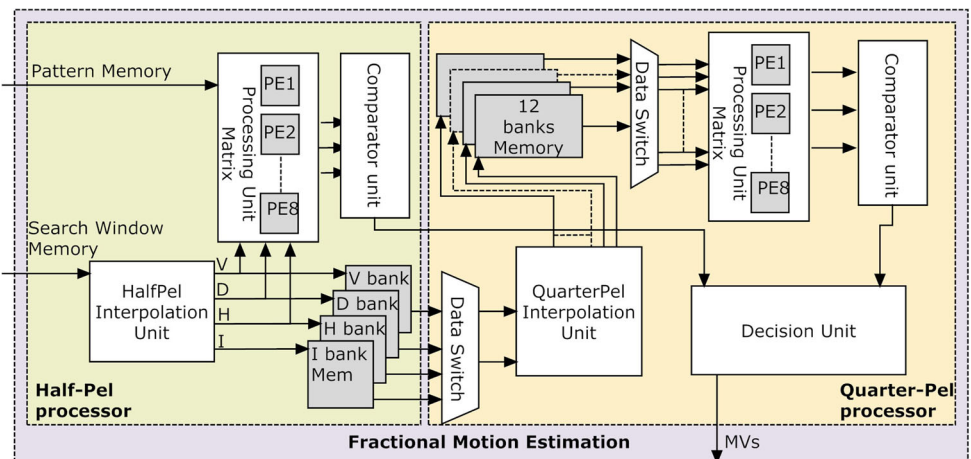
Once half-pel samples are available, the pixel values at quarter-pel locations are processed with basic bilinear weighting of the values at half-pel and integer-pel positions. Nevertheless, the quarter-pel processing is less regular. As shown in Fig. 8b, the orientation of the pixels is considered, and 12 different kinds of processing, which generate the quarter-pel positions, can be observed. Therefore, we propose a novel architecture using four different memory banks for half-pel processing and 12 banks for the quarter-pel refinement. For instance for half-pel refinement, the original pixel named  $i$  and three kinds of interpolated pixel are stored, respectively, in I, H, V and D memory banks. This approach has been proposed by Ruiz [23] only for half-pel refinement, we propose in this paper to apply it also to quarter-pel refinement. This architecture is highly efficient in terms of data broadcasting, therefore the processing time decreases. Nevertheless, the number of small cache memory is increased. Each bank is implemented with dual port memory embedded into the FPGA component. Only two memory blocks are required to store the reduced number of each class of interpolated pixels. Therefore, the used hardware resources are still low and suitable for FPGA implementation. For instance, the 32 memory blocks represent less than 8 % of Virtex 6v1x240 FPGA.

The Fig. 9 depicts the overall block diagram of the proposed architecture of FME. It consists of two processors used in pipeline: half-pel and quarter-pel processors which are interpolation-based units, processors units, memory unit and comparator unit. The architecture of processor and comparator unit is the same for both half-pel and quarter-pel. In each refinement stage, eight candidates around the refinement center are evaluated simultaneously.

**Fig. 8** a Half-pel pixel values, b Quarter-pel pixel values



**Fig. 9** Overall fractional motion estimation architecture



Our half-pel interpolation unit is based on the well-known Yang’s solution [13], which processes a row 16-pixel interpolation unit. Indeed, a problem related to Chen’s 4-pixel interpolation unit [2] is the redundant interpolating operations which appear in the overlapping area of the adjacent interpolation window. To overcome this problem, a new architecture based on 16-pixel interpolation unit with nine or eighteen  $16 \times 16$  processing units is proposed by Yang’s which removes all the redundancies. Our design, as Yang’s, adopts a short-latency 16-pixel wide interpolator to increase throughput and eliminate redundant interpolation. Moreover, all sizes of blocks are processed by  $16 \times 16$  processing units. Therefore, the hardware utilization is low when processing small size blocks ( $4 \times 4$  and  $4 \times 8$ ). When  $4 \times 8$  and  $4 \times 4$  blocks are processed in parallel, the processing units require simultaneous accesses to two memory areas.

Indeed, the integer motion vector could be different for each sub-block. Hence, the memory banks should be doubled for reading the reference pixels in parallel. Yang’s architecture enables higher processing performance to be obtained than Chen’s implementation. We used Yang’s architecture principles for our solution, nevertheless we reduced the number of processors and the memory resources thanks to our memory organization as discussed in Sect. 4.

### 3.3.2 Scheduling dataflow

The interpolation is different for half-pel and quarter-pel refinement as explained previously in this section, nevertheless the task scheduling is absolutely identical for both sub-pel Processing Unit Matrix (PUM). Each matrix

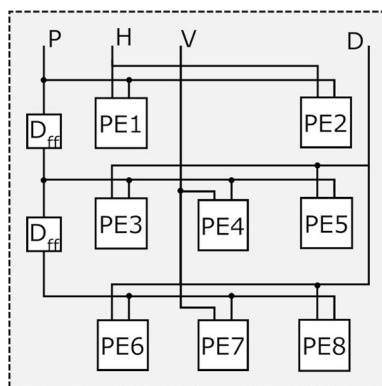


**Table 1** Scheduling diagram for  $4 \times 4$  block processing

Cycle	P	H	D	V	PE1	PE2	PE3	PE4	PE5	PE6	PE7	PE8	SAD
0	$p_{0,j}$	$h_{0,j-1}h_{0,j}$			$ h_{0,j-1}-p_{0,j} $	$ h_{0,j}-p_{0,j} $	-	-	-	Previous block			
1	$p_{1,j}$	$h_{1,j-1}h_{1,j}$	$d_{-1,j-1}d_{-1,j}$	$V_{-1,j}$	$ h_{1,j-1}-p_{1,j} $	$ h_{1,j}-p_{1,j} $	$ d_{-1,j-1}-p_{0,j} $	$ v_{-1,j}-p_{0,j} $	$ d_{-1,j}-p_{0,j} $	-	-	-	
2	$p_{2,j}$	$h_{2,j-1}h_{2,j}$	$d_{0,j-1}d_{0,j}$	$V_{0,j}$	$ h_{2,j-1}-p_{2,j} $	$ h_{2,j}-p_{2,j} $	$ d_{0,j-1}-p_{1,j} $	$ V_{0,j}-p_{1,j} $	$ d_{0,j}-p_{1,j} $	$ d_{0,j-1}-p_{0,j} $	$ V_{0,j}-p_{0,j} $	$ d_{0,j}-p_{0,j} $	
3	$p_{3,j}$	$h_{3,j-1}h_{3,j}$	$d_{1,j-1}d_{1,j}$	$V_{1,j}$	$ h_{3,j-1}-p_{3,j} $	$ h_{3,j}-p_{3,j} $	$ d_{1,j-1}-p_{2,j} $	$ V_{1,j}-p_{2,j} $	$ d_{1,j}-p_{2,j} $	$ d_{1,j-1}-p_{1,j} $	$ V_{1,j}-p_{1,j} $	$ d_{1,j}-p_{1,j} $	-
4	-	-	$d_{2,j-1}d_{2,j}$	$V_{2,j}$	-	-	$ d_{2,j-1}-p_{3,j} $	$ V_{2,j}-p_{3,j} $	$ d_{2,j}-p_{3,j} $	$ d_{2,j-1}-p_{2,j} $	$ V_{2,j}-p_{2,j} $	$ d_{2,j}-p_{2,j} $	-
5			$d_{3,j-1}d_{3,j}$	$V_{3,j}$			-	-	-	$ d_{3,j-1}-p_{3,j} $	$ V_{3,j}-p_{3,j} $	$ d_{3,j}-p_{3,j} $	PE1 PE2
6					Next	block				-	-	-	PE3 PE4 PE5
7													PE6 PE7 PE8
8													-

consists of eight PE. As explained before eight positions are considered for half-pel as well as quarter-pel refinement. Hence, each of processor is responsible for one search position. Therefore, the performances are identical for both modes. As both task scheduling are identical only half-pel mode is described in this subsection.

For one search position, one sort of interpolated pixel is involved: **i**, **h**, **v** or **d**. The interpolated pixel is compared with the pattern values, noted P. As each of processor is responsible for only one search position, the eight processors are connected as depicted in Fig. 10. This processor network enables the task scheduling to be improved compared to Ruiz’s solution [23]. A eight sub-pel positions can be processed in 8 cycles instead of 12 cycles in Ruiz’s architecture. The Table 1 presented the task scheduling for a half-pel FME. To simplify the data flow explanation only a  $4 \times 4$  sub-block is considered. As depicted in Fig. 9, for this  $4 \times 4$  sub-block,  $5 \times 4$  horizontal samples **{h}**,  $4 \times 5$  vertical samples **{v}** and  $5 \times 5$  diagonal samples **{d}** are



**Fig. 10** Network routing data for half-pel processor matrix

processed. The interpolated pixels are transferred row by row by the interpolation unit. The **h** pixels are available at output of the interpolation unit one cycle before **d** and **v** samples. At each cycle, each PE enables a 4-pixel macro-block row to be compared with one 4-pixel row of the interpolated pixels. For instance, the pattern row  $\mathbf{p}_{(0,j)}$ ,  $j = 0..3$ , is constituted of the following four pixels  $\{\mathbf{p}_{0,0}, \mathbf{p}_{0,1}, \mathbf{p}_{0,2}, \mathbf{p}_{0,3}\}$ . For **h** and **d** samples, the 5-pixel row is split in two 4-pixels, for instance  $\mathbf{h}_{0,j-1}$  and  $\mathbf{h}_{0,j}$ , which are, respectively, constituted of  $\{\mathbf{h}_{0,-1}, \mathbf{h}_{0,0}, \mathbf{h}_{0,1}, \mathbf{h}_{0,2}\}$  and  $\{\mathbf{h}_{0,0}, \mathbf{h}_{0,1}, \mathbf{h}_{0,2}, \mathbf{h}_{0,3}\}$ . The two 4-pixel rows enable two positions to be processed simultaneously. To minimize data access to pattern memory, the pattern row is used for **h** pixels then transferred for the comparison with **d** and **v** pixels. As depicted in Fig. 10, the P data are transferred from the pattern memory to processors PE1 and PE2 then through delay elements  $D_{ff}$ , to PE3, PE4, PE5 and finally to PE6, PE7 and PE8. The p, h, d, and v data are inputted at each cycle. At cycle 0, only p and h are available, PE1 and PE2 processors are activated.  $\mathbf{h}_{(0,j-1)}$  and  $\mathbf{h}_{(0,j)}$  are available, respectively, to PE1 and PE2.  $\mathbf{p}_{(0,j)}$  is broadcasted to processors PE3-4-5. The **h**, **v** and **d** pixels are available from cycle 1, five processors can then be activated. At cycle 2, three extra search positions can be considered with **v** and **d** samples, therefore all eight processors perform. Each PE requires six cycles to complete the  $4 \times 4$  matching. The first SAD scores, resulting from PE1 and PE2 are then available at cycle 5. At cycle 8, all the SAD values are available at comparison unit level. The half-pel refinement of the next block can start at cycle 5. We use Yang’s architecture principles, therefore the FME can be applied on a  $16 \times 16$  block or any sub-blocks ( $4 \times 4$ ,  $4 \times 8$ ).

#### 4 Implementation results and discussion

The proposed architecture can be considered as a low-cost implementation of a motion estimator. The hardware resources requested for our implementation are presented in Table 2. The implementation has been done on a Virtex6 FPGA target (6v1x240tff784-3). The integer and fractional estimators are regrouped in this table. Note that only two search strategies are currently implemented: FS and DS.

The Table 3 shows the comparison between our proposed architecture of IME based on Virtex 6 FPGA and previously published ASIC- and FPGA-based processors of VBSME [3, 15, 18–20]. The first three selected architectures enable a matching to be processed row-by-row. Therefore, the number of PEs as well as the data bandwidth is reduced compared to the resources required by solutions which process a matching in once. For these three low-cost architectures, the 41 possible motion vectors are carried out by a common pipelined structure constituted of 16 PEs which enables a matching to be processed in 16 cycles. A  $16 \times 16$  search range can therefore be processed in 4,096 cycles. The architecture described in [18] is original as a  $32 \times 32$  searching range can be performed, keeping low the number of PEs. High clock frequency range can be reached due to the 16 PEs pipelined structure and a pixel truncation technique. Nevertheless, the process still required 26,624 cycles due to the large searching range and the higher number of sub-blocks to be performed. The architectures described in [19] and [20] enable also large search ranges to be processed, respectively,  $32 \times 32$  and  $65 \times 65$ , nevertheless the number of processing elements is, respectively, extended to 64 and 256 PE. The required hardware resources is therefore higher.

The originality of our approach is to propose an architecture, which enables several search strategies to be implemented (i.e FS and DS). The FS mode enables higher accuracy to be obtained as described in Sect. 2.1, meanwhile the DS mode provides higher performances in terms of throughput. These two methods are both implemented in our design. The user can change on the fly the search mode according to context (i.e. the required Quality of Service or event detected in the video scene). Moreover, we are investigating on the implementation of other fast motion

algorithm instead of DS mode to extend user's choice. A main challenge was to provide the input data without latency even for two matching with non-consecutive access in the cache memory. In term of frequency we overcome all designs with 438 MHz, as we use more recent technology (40 nm), which enables a matching to be processed in 36 ns. Then our architecture processes  $1/(256 \times 36e - 9) = 108$ , 5k macro-blocks/s. In a FS mode and using a  $16 \times 16$  search window,  $720 \times 576$  (resp.  $1,920 \times 1,080$ ) video streams can be processed at 67 (resp. 13) fps. The architecture proposed by Su-Jin in [19] is faster (151,9k macro-blocks/s), but our structure allows to obtain better results using a DS method, considering a realistic average range of 15–30 matching per macro-blocks. In this case, a 1080 HD video stream can be processed between 223 and 111 fps.

The Table 4 shows the comparison of the FME designs implemented in similar technology (0.18  $\mu\text{m}$ ) and our implementation based on FPGA (6v1x240tff784-3). Our implementation can be considered as a low-cost FME architecture based on Yang's design. In this architecture, we use a 16-pixel interpolation unit allowing speed-up interpolation step against Chen's design. Moreover, all blocks are performed sequentially. So all 41 MV are refined in 1,062 cycles. As expected and detailed in Sect. 3.3, the very low-cost architecture proposed by Chen's is less performing than Yang's. Our architecture proposes very competitive performances and similar results to Yang's one. Comparing the two architectures, our solution reduces by two the memory size and decreases by 8 (instead of 18) the processor number for each sub-pel refinement. Nevertheless, it doubles the processing time for  $4 \times 8$  and  $4 \times 4$  blocks. Our solution proposes competitive performances with state of the art. Meanwhile, a current investigation aims to present a high performance architecture based on two concatenated 8-pixel interpolation unit which operate in parallel. The structure requires a double number of memory banks and PEs. Using the principles detailed in Sect. 3.3, the number of cycles could be reduced not only for 4-pixel wide blocks but also 8-pixel wide blocks. In this configuration, the number of processors would reach 16 processors for half-pel accelerator as well as the quarter-pel one. This optimized architecture

**Table 2** Motion estimation implementation results for IME and FME modes

Motion estimation (device:6v1x240tff784-3)					
Logic utilization	Used (IME/FME)		Available	Utilization (IME/FME) (%)	
Number of slice register	1,168	11,944	301,440	>1	3
Number of slice LUTs	1,281	17,426	150,720	>1	11
Number of BRAMs/FIFOs	1	32	416	>1	8
Maximum frequency	Frequency IME: 438 MHz Frequency FME: 253 MHz				

**Table 3** Performances comparison of IME architectures supporting VBSME

Ref.	nb PE's	Searching range	Latency	Tech ( $\mu\text{m}$ )	Freq (MHz)	Max k Mblocks/s using FS	Max k Mblocks/s using DS
Swee [3]	16	$16 \times 16$	4,096	0.13	294	72.9	n/a
Su-Jin [19]	64	$16 \times 16$	1,027	Spartan 3 (0.9)	178	151.9	n/a
		$32 \times 32$	4,099			41.9	n/a
Lei [20]	256	$65 \times 65$	5,216	0.18	260	48.6	n/a
Song [15]	16	$16 \times 16$	4,096	0.18	266	9.9	n/a
Fatemi [18]	16	$32 \times 32$	26,624	0.18	316	11.8	n/a
Ours	16	$16 \times 16$	4,096	Virtex 6 (0.04)	436	108.5	1,806

**Table 4** Comparison of fractional motion estimation architectures performances

	Chen's [2]	Yang's [13]	Ruiz's [23]	Our's
Year	2004	2006	2010	2012
Cycles	1664	790	870	1062
Nbr of PEs	9	18	8	8
Interpolation unit width	10	22	10	22
Tech ( $\mu\text{m}$ )	UMC 0.18	TSMC 0.18	UMC 0.18	Virtex-6 vlx240tff784-3
Freq (MHz)	100	285	290	253
Throughput (k Mblocks/s)	49	250	NA	232

allows processing two sub-blocks simultaneously when width is 8 or 4. Preliminary results of our modified architecture, so it achieves all 41 refinement sub-blocks in 610 cycles. Yang's architecture has been implemented with a 0.18  $\mu\text{m}$  technology. It can process 1080 HD video streams at a frame rate of 30 fps when running at 200 MHz. The very low-cost version architecture, using the 40 nm technology available on Virtex 6 FPGA, can process this video stream at frame rate of 29 fps at 250 MHz (around 232k macro-blocks/s).

## 5 Conclusion

We proposed in this paper, a flexible motion estimator which enables the integer search strategy to be adjusted and the optional VBSME and sub-pel refinements to be processed. This low-cost implementation, based on Virtex FPGA enables to reach high-speed performances. Hence for IME, 1080 HD video streams can be processed up to 223 fps using fast search strategy (around 1800k macro-blocks/s). Moreover for FME mode, the same video streams can be processed at frame rate of 29 fps at 250 MHz (around 232k macro-blocks/s). Current developments aim to improve these performances, specially the sub-pel interpolation units. This solution can, therefore, represents an efficient adaptative solution for many video coding applications. Finally, the use of FPGA technology enables the Dynamic Partial Reconfiguration (DPR) to be

considered. Therefore, the ME accelerators could be even more scalable and can be dynamically adjusted according to the events happening in the video scene or some environment modifications (as a network bandwidth reduction). Obviously, a modification of the architecture would impact power consumption, and therefore can be investigated for power saving for mobile devices.

## References

1. Wiegand, T., Sullivan, G.J., Bjontegaard, G., Luthra, A.: Overview of the H.264/AVC video coding standard. *IEEE Trans. Circuits Syst. Video Technol.* **13**(7), 560–576 (2003)
2. Chen, T.C., Huang, Y.W., Chen, L.G.: Fully utilized and reusable architecture for fractional motion estimation of H.264/AVC. In: *IEEE ICASSP*, pp. 9–12 (2004)
3. Swee, Y.Y., McCanny, J.V.: A VLSI architecture for variable block size video motion estimation. *IEEE Trans. Circuits Syst. Video Technol.* **51**(7), 384–389 (2004)
4. Koga, T., Iinuma, K., Hirano, A., Iijima, Y., Ishiguro, T.: Motion compensated interframe coding for video conferencing. In: *Proc. Nat. Telecommun Conf.*, pp. G5.3.1–G5.3.5, New Orleans (1981)
5. Li, R., Zeng, B., Liou, L.M.: A new three-step search algorithm for fast motion estimation. *IEEE Trans. Circuits Syst. Video Technol.* **4**(4), 438–442 (1994)
6. Po, L.M., Ma, W.C.: A novel four-step search algorithm for fast block motion estimation. *IEEE Trans. Circuits Syst. Video Technol.* **6**(3), 313–317 (1996)
7. Zhu, C., Lin, X., Chau, L.P.: Hexagon-based search pattern for fast block motion estimation. *IEEE Trans. Circuits Syst. Video Technol.* **12**(5), 349–355 (2002)

8. Zhu, S., Ma, K.K.: A new diamond search algorithm for fast block matching motion estimation. *IEEE Trans. Image Process* **9**(2), 287–290 (2000)
9. Cheung, C., Po, L.M.: A novel cross-diamond search algorithm for fast block motion estimation. *IEEE Trans. Circuits Syst. Video Technol.* **12**(12), 1168–1177 (2002)
10. Liu, L., Feig, E.: A block-based gradient descent search algorithm for block motion estimation in video coding. *IEEE Trans. Circuits Syst. Video Technol.* **6**(4), 419–422 (1996)
11. Lee, Y.G., Ra, J.B.: Fast motion estimation robust to random motions based on a distance prediction. *IEEE Trans. Circuits Syst. Video Technol.* **16**(7), 869–875 (2006)
12. Ismail, Y., McNeelly, J., Shaaban, M., Bayoumi, M.: Enhanced efficient diamond search algorithm for fast block motion estimation. In: *IEEE ISCAS*, pp. 3198–3201, Taipei (2009)
13. Yang, C., Goto, S., Ikenaga, T.: High performance VLSI architecture of fractional motion estimation in H.264 for HDTV. In *Proceedings of the IEEE ISCAS*, pp. 2605–2608, Greece (2006)
14. Chen, Y.H., Chen, T.C., Chien, S.Y., Huang, Y.W., Chen, L.G.: VLSI architecture design of fractional motion estimation for H.264/AVC. *J. Signal Process. Syst.* **53**(3), 335–347 (2008)
15. Song, Y., Liu, Z., Ikenaga, T., Goto, S.: A VLSI architecture for variable block size motion estimation in H.264/AVC with low cost memory organization. *IEICE Trans. Fundam.* **E89**(12), 3594–3601 (2006)
16. Chen, T.C., Chien, S.Y., Huang, Y.W., Tsai, C.H., Chen, C.Y., Chen, T.W., Chen, L.G.: Analysis and architecture design of an HDTV720p 30 frames/s H.264/AVC encoder. *IEEE Trans. Circuits Syst. Video Technol.* **16**(6), 673–688 (2006)
17. Liu, Z., Song, Y., Liu, Z., Ikenaga, T., Goto, S.: A fine-grain scalable and low memory cost variable block size motion estimation architecture for H.264/AVC. *IEICE Trans. Fundam.* **E89-C**(12), 1928–1936 (2006)
18. Fatemi, M.R.H., Ates, H.F., Salleh, R.: 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)
19. Su-Jin, L., Cheong, G.K., Shin, D.K.: A pipelined hardware architecture for motion estimation of H.264/AVC. In: *Proceedings of the 10th Asia-Pacific conference on Advances in Computer Systems, Architecture ACSAC'05*, pp. 79–89 (2005)
20. Lei, D., Wen, G., Ming, Z.H., Zhen, Z.J.: An efficient hardware implementation for motion estimation of AVC standard. *IEEE Trans. Consumer Electron.* **51**(4), 1360–1366 (2005)
21. Chen, C.Y., Chien, S.Y., Huang, Y.W., Chen, T.C., Wang, T.C., Chen, L.G.: Analysis and architecture design of variable block size motion estimation for H.264/AVC. *IEEE Trans. Circuits Syst. Regul. Pap.* **53**(3), 578–593 (2006)
22. Dubois, J., Mattavelli, M., Pierrefeu, L., Miteran, J.: Configurable motion-estimation hardware accelerator module for the Mpeg-4 reference hardware description platform. In: *Proceedings of IEEE International Conference on Image Processing (ICIP05)*, Genova (2005)
23. Ruiz, G.A., Michell, J.A.: An efficient VLSI architecture of fractional motion estimation in H.264 for HDTV. *J. Signal Process. Syst.* **62**(3), 443–457 (2010)