J Sign Process Syst
DOI 10.1007/s11265-010-0475-8

An Efficient VLSI Architecture of Fractional Motion Estimation in H.264 for HDTV

G. A. Ruiz · J. A. Michell

Received: 17 September 2009 / Revised: 9 March 2010 / Accepted: 11 March 2010

Abstract Fractional Motion Estimation (FME) in high-definition H.264 presents a significant design challenge in terms of memory bandwidth, latency and area cost as there are various modes and complex mode decision flow, which require over 45% of the computation complexity in the H.264 encoding process. In this paper, a new high-performance VLSI architecture for Fractional Motion Estimation (FME) in H.264/AVC based on the full-search algorithm is presented. This architecture is made up of three different pipeline processors to establish a trade-off between processing time and hardware utilization. The computing scheme based on a 4-pixel interpolation unit with a 10-pixel input bandwidth is capable of processing a macroblock (MB) in 870 clock cycles. The final VLSI implementation only requires 11.4 k gates and 4.4kBytes of RAM in a standard 180 nm CMOS technology operating at 290 MHz. Our design generates the residual image and the best MVs and mode in a high throughput and low area cost architecture while achieving enough processing capacity for 1080HD (1920×1088@30fps) real-time video streams.

Keywords H.264 · High-definition television (HDTV) · Fractional motion estimation · Video coding

1 Introduction

The video coding standard H.264/AVC, developed by the Joint Video Team (JVT), achieves higher coding efficiency than previous coding standards especially in high-definition and high-rate video sequences. The superior coding performance of H.264/AVC originates from new techniques such as quarter-pixel fractional motion estimation, variable block sizes, multiple reference frame motion estimation, complex intra prediction modes, context-based entropy coding, and so on [1, 2]. However, these advanced video coding techniques require huge computational complexity and memory bandwidth for the encoding process. Thus, hardware acceleration encoder design is still essential to enable implementation of fast architectures for real-time video applications.

Motion estimation (ME) is the most important part of H.264/AVC in exploiting the temporal redundancy between successive frames and 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 [3]. Each MB may then be segmented into subblocks of different sizes, as illustrated in Fig. 1. 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 subblocks 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. In H.264/AVC, the latter process takes about 45% of ME time and, for high resolution application, VLSI implementation is essential.

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
e-mail: ruizrg@unican.es

Published online: 06 April 2010

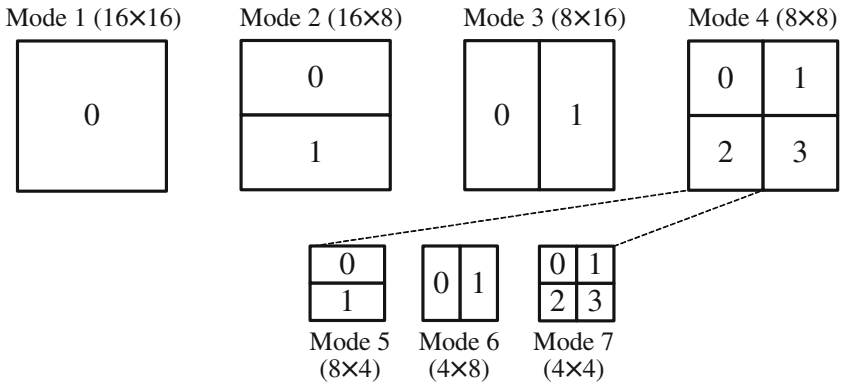

Figure 1 Sub-macroblock partitions.

There are many proposed IME algorithms based on one-dimension [5] or two-dimension [6] processing elements, different search strategies such as the three-step search [7], hexagon search [8] and diamond search [9], or implementations aiming to reduce the power [10] or to minimize the off-chip memory bandwidth [11]. However, only a few FME implementations have been discussed in spite of FME having a strong impact on the peak-signal-to-noise ratio (PSNR) and the amount of computation required for FME is even more than needed for IME. Several algorithms have been proposed to speed up the FME process, although they decrease the video quality to some extent, such as those based on early termination techniques [12] (average ΔPSNR= −0.02 dB, ΔBitrate=2.91%), search control by using neighbouring motion vectors [13] (ΔPSNR=−0.03 dB), size reduction of tap filters [14] (ΔPSNR=−0.003 dB) or reduction of search area [15] (ΔPSNR=−0.17 dB, ΔBitrate= 4.08%). Different FME implementations which make a trade-off among input bandwidth of reference pixels, hardware overhead and number of clock cycles for processing all MBs have been described. In [16], VLSI architecture for FME with a regular schedule and high utilization is presented. It uses a 4-pixel interpolation unit to process 10 integer pixels in parallel at 100 MHz taking 1648 clock cycles to compute a whole macroblock (MB). An improved architecture with a 16-pixel interpolation unit to reduce this number of cycles to 790 is described in [17]. It achieves high processing capacity for encoding real-time HDTV video streams. However, this high throughput has the penalty of large area (2.4×) and memory bandwidth up to 22 pixels. A high throughput hardware architecture based on three parallel processing engines reduces the number of clock cycles to only 616, but with a high cost in area [19]. Other FME designs search for hardware reduction or high efficiency by making some constraints in the FME algorithm. Thus, the scheme proposed in [20] achieves a reduction of more than 50% in computation by exploring neighbourhood position and early termination with acceptable loss of quality. The architecture described in [21] reduces the search area and number of MVs needed in order to achieve low-latency and hardware efficiency. Finally, designs suitable for a FPGA implementation are presented in [22, 23]

In this paper, a VLSI architecture based on the full-search algorithm for implementation of FME is described. Its architecture is made up of three different pipeline processors: a half-pixel processor, a quarter-pixel processor and a mode decision processor. As a result, our design generates the residual image and the best MVs in a high throughput, low area cost architecture. The design is implemented with only 11.4 k gates and 4.4kBytes of RAM in a standard 180 nm CMOS technology and it operates at 290 MHz. The latency of 870 clock cycles is sufficiently low to process 1080HD real-time videos. The remainder of this paper is organized as follows. In Section 3, a brief description of the FME scheme is discussed. Section 4 presents some details of the proposed architecture based on three processors, and Sections 5, 6 and 7 describe some details about the design of those processors. The implementation results and comparisons are listed in Section 7. Finally, the conclusions are stated in Section 8.

2 Description of the FME Algorithm

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 @ 30fps) the work of processing all 41 subblocks belonging to a 16× 16 MB should take less than 4.1 μsec, equivalent to 1025 clock cycles at a clock frequency of 250 MHz, which is available for most of current technologies. IME and FME must be computed in these 1025 cycles which will affect the efficiency of the hardware implementation. IME is performed prior to FME. Integer pixel search tries to find the best matching integer position and the best integer pixel motion vectors (MV) are determined by using a performance cost metric. Then, the FME performs a half-pixel refinement about the integer search positions and then a quarter-pixel one is performed around the best half-pixel

positions. As a result, a pipeline architecture is a must to implement IME and FME [24, 25].

In an efficient FME implementation, the trade-off among processing time, memory access data bus and hardware utilization should be balanced. Here the memory access and sub-pixel interpolation must comply with some time constraints, taking into account that FME requires more input data to be processed than is used in sub-pixel motion estimation. Besides, FME shares on-chip memory previously stored in the IME stage as FME fetches reference data to that memory when IME has just finished. In our design, a 4-pixel interpolation unit is adopted because it provides a trade-off between the processing time and hardware utilization, and it is also highly compatible with many proposed IME architectures. This unit is capable of processing every subblock in a MB, since each subblock can be decomposed in terms of single 4×4 basic blocks. However, the 6-tap filter used in interpolation process requests extra input pixels so that each 4×4 subblock is extended with a border of 3 pixels around it, resulting in a (3+4+3)×(3+4+3)=10×10 window; thus, the input data is set to 10 pixels' width. To reduce accesses to memory and reuse interpolated data, vertically adjacent 4×4 blocks are processed using a similar scheme to that proposed in [16], which helps to avoid redundant memory accesses (about 25–35% of total bandwidth). In this scheme, vertical scan order is adopted to facilitate the interpolation operations between two adjacent vertical columns for subblocks of 8× and 16×. For the sake of clarity, Fig. 2 shows the vertical arrangement to carry out the interpolation process of a 16×16 MB. With 10 pixels of input data, the processing of a column takes 22 clock cycles, and the whole block 22×4=88 clock cycles. Table 1 lists the clock cycles needed to process all 41 different modes, the total number being 832. A problem related to vertical interpolation is the redundant interpolating operations which appear in the overlapping area of the adjacent interpolation window. To overcome this problem, a new schedule based on a 16-pixel interpolation is proposed in [17], which removes all the redundant columns. Although this architecture can save more than 50% clock cycles, it uses input data of 22-pixels' width and the cost in area is about 2.4 times greater than 4-pixel interpolation. Moreover, it incurs low computational redundancy but is inefficient in handling variable size.

Table 1 Clock cycling for different subblocks.

Subblock type	Number of blocks	Cycles/block	Total cycles
16×16	1	22×4	88
16×8	2	14×4	112
8×16	2	22×2	88
8×8	4	14×2	112
8×4	8	10×2	160
4×8	8	14×1	112
4×4	16	10×1	160
Total latency			832

2.1 Langrangian Cost

The ME algorithm determines the best mode which minimizes the matching error between reference MB and candidate MB. In the JM version 15.1 of the H.264 reference software, which is available on-line at [4], the ME chooses the best mode by using a Lagrangian mode decision to compute not only the sum of absolute differences (SAD) but also an estimation of the bits required to code MVs. For each subblock of a MB, the ME algorithm minimizes the following Langrangian cost (J) defined as

$$J = SAD + \lambda\, MV\cos t(MVcur - MVpred) \quad (1)$$

where SAD denotes a distortion measure (in our case it is the SAD) between the original and the coded partition predicted from the reference frames, MVcost represents the number of bits (according to a table of entropy coding) required to code the difference of current MV (MVcur) and motion prediction MVpred, and λ is the Langrangian multiplier imposed using a suitable rate constraint. To calculate the MVpred, the MV of the neighbouring blocks must be available or sufficiently estimated as they not only depend on neighbouring MBs, but also on earlier blocks within a MB [3]. Figure 3 shows an example of definition

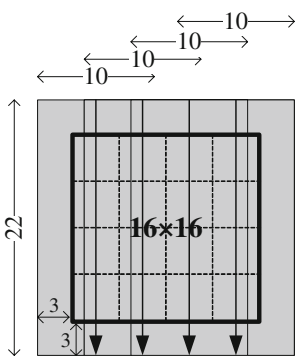

Figure 2 Vertical interpolation of 16×16 MB.

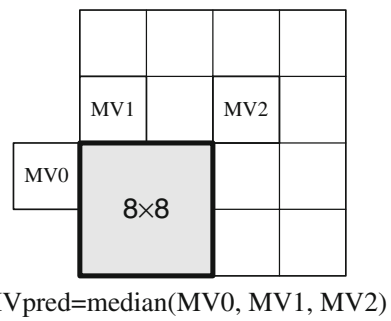

Fig. 3 Example of MVpred of 8×8 macroblock.

of MVpred for an 8×8 subblock. In this case, the MVpred is computed from the median of left (MV0), top (MV1) and top-right (MV2). MV0 belongs to the previous 16×16 MB, and MV1 and MV2 belong to 4×4 subblocks of the same MB that have just been processed.

The proposed FME algorithm uses the well-known tree structured motion compensation method of H.264 to obtain the MVpred of each subblock. In this method, the subblocks are processed in a particular order to guarantee that every MVpred neighbouring a subblock is available before processing. Otherwise, an incorrect MVpred significantly worsens coding results because it leads the motion estimation in the wrong direction. The Lagrangian cost function can only be computed after the MVs of neighbouring blocks are determined, which causes an inevitable sequential processing. MBs and subblocks in a MB cannot be processed in parallel. As a result, while processing a subblock in the half-pixel processor, the MVs of neighbours at quarter resolution are not available because the quarter processor has not computed them yet. This problem only arises in the subblocks labelled 1 in Fig. 1 belonging to Mode 2, 3, 4, 5 and 6, and subblocks labelled 1, 2 and 3 in Mode 7. In the proposed FME, these subblocks only use half pixel resolution in MVpred because MVpreds with quarter resolution are not available. Simulations made with typical video sequences have proved that the effects of this restriction on overall PSNR and bitrate are insignificant (average ΔPSNR=−0.003 dB and ΔBitrate=0.05%).

3 Proposed Architecture

Figure 4 shows the block diagram for the proposed design of FME hardware architecture based on three different pipeline processors: half-pixel processor, quarter-pixel processor and mode decision processor. This architecture makes a trade-off between the processing time and the hardware utilization to reach the capacity of encoding the high-resolution real-time video stream for HDTV at low cost in area. It uses a completely standard-compatible full-search algorithm based on 4×4 block decomposition and a vertical arrangement to reduce the encoding time.

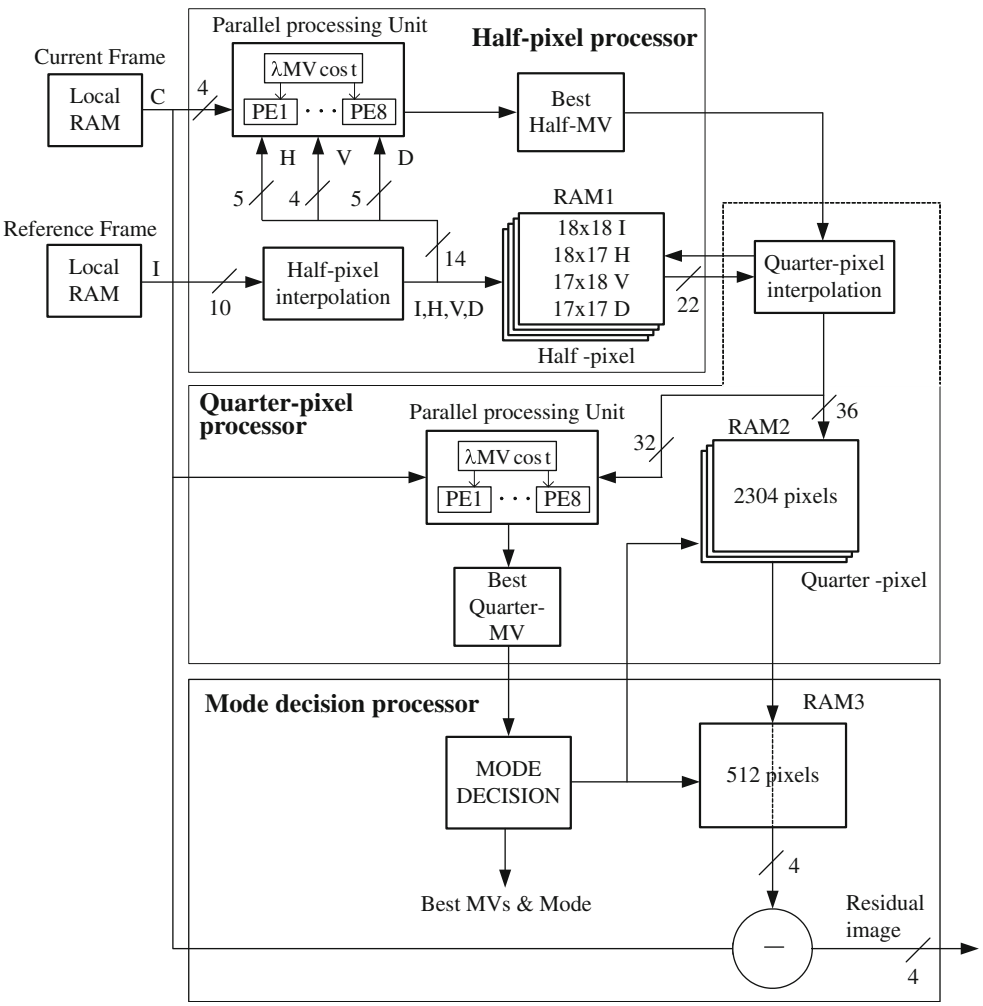

Figure 4 Block diagram of general FME architecture.

The input data include the best integer MV with its Langrangian cost acquired in IME, MVpred of the adjacent MBs and search area data from a local memory which is input row by row. The half-pixel processor reads the reference input data from a local RAM and performs half-pixel interpolation and a full half-pixel search for each subblock size. The interpolated samples are stored in RAM1 and processed in the processing unit. This unit is made up of 8 parallel processing elements (PE) to obtain the best half-MV according to minimum Lagrangian cost. The quarter-pixel processor uses the best half-MV and the interpolated samples stored in RAM1 to generate all the quarter-pixels around the half/integer samples using a bilinear filter. These are stored in RAM2. Similarly to the previous processor, these interpolated quarter-pixels are processed in the processing unit in order to extract the best quarter-MVs. After that, the mode decision processor evaluates, temporarily stores the best matching interpolated image in RAM3 and makes a decision about block modes and final MVs. However, the final decision is not taken until all 41 subblocks are processed. Finally, this processor generates the best MVs and the best mode of the MB, as well as the residual image to be coded.

Figure 5 depicts the timing diagram for performing FME to process one MB. It dispatches all 41 subblocks ranging from 16×16 to 4×4 reads according to the tree-structured motion compensation order specified in the JM reference software. The input reading process consumes 832 clock cycles (see Table 1) to load all input data which bounds the whole processing time. The half-pixel processor performs a half-pixel interpolation, which is stored in a bank of four double port SRAMs (RAM1), and takes a decision about the best half-MV after computing the Langrangian cost. On taking this decision, the quarter-pixel processor starts off fetching data to RAM1. Here, the reading process takes fewer clock cycles than the former processor; for a M×N subblock, it takes (M+1)×N/4 clock cycles in comparison with the former's (M+6)×N/4. As a result, this quarter-pixel processor has idle clocks waiting to finish the processing of some subblocks in the half-pixel processor. After taking the best quarter-MVs decision and computing a new Langragion cost, the mode decision processor starts off fetching data to a bank of three double-port SRAMs (RAM2). The reading process takes the same number of clock cycles as the size of the processing subblock (M×N); there are also some idle clock cycles here. The candidate samples specified by the best quarter-MVs are stored in RAM3 according to a scheme described in Section 7. After 870 clock cycles, the final decision is taken by processing all 41 subblocks. The proposed architecture for generating the residual image of a MB takes a total of 936 clock cycles: 832 cycles for reading the input data, 38 cycles of latency and 66 cycles to generate the residual image.

4 Half-Pixel Processor

The half-pixel processor performs half-pixel interpolation and a half-pixel search for each subblock. It is mainly composed of a half-pixel interpolation unit, four double-port SRAMs (RAM1) to store integer and half-pixel samples, a processing unit (PU) to compute the Lagrangian

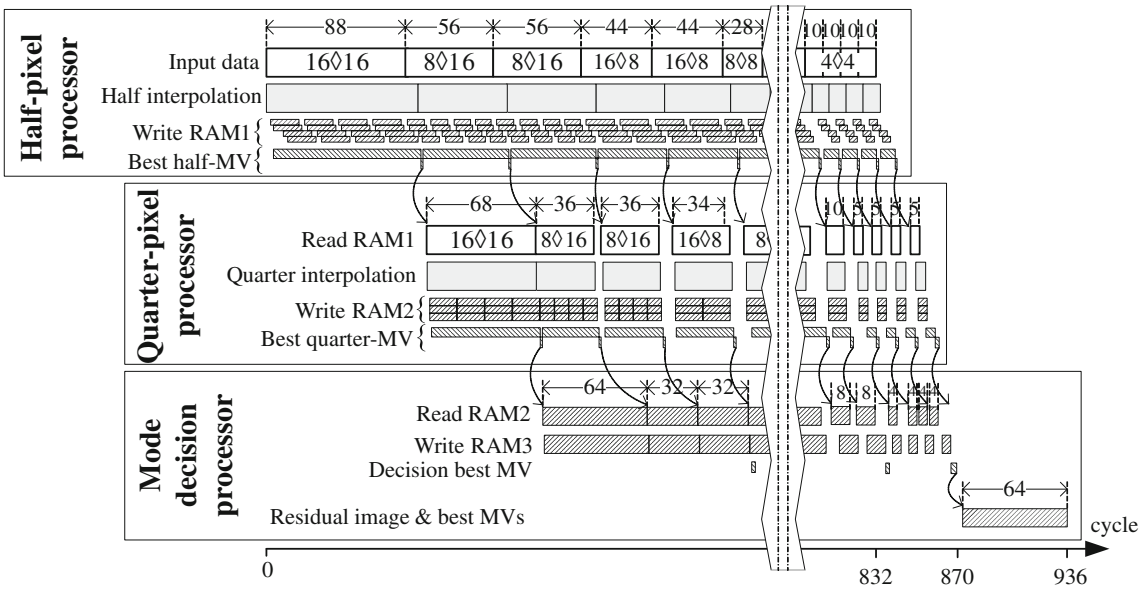

Figure 5 Timing diagram for performing FME for one MB.

cost, and a best half MV unit to select the MVs with minimum Lagrangian cost. As a result, the best MVs are passed to the quarter-pixel processor.

4.1 Half-Pixel Interpolation Unit

In the H.264/AVC, the prediction luma sample values at the half pixel are calculated by applying a 6-tap Wiener filter in both horizontal and vertical directions. The tap coefficients are (1, −5, 20, 20, −5, 1). For the sake of clarity, Fig. 6 illustrates the spatial relationship of the integer {I}, and half-pixel vertical {V}, horizontal {H} and diagonal {D} positions in the luminance interpolation. The horizontal half-pixel value $H_{0,0}$ is computed from an intermediate value $H'_{0,0}$ which is calculated in turn from the six nearest integer pixel values located at horizontal direction according to the following equation

$$H'_{0,0} = I_{0,-2} - 5I_{0,-1} + 20I_{0,0} + 20I_{0,1} - 5I_{0,2} + I_{0,3} \quad (2)$$

The half sample $H_{0,0}$ is calculated by clipping $H'_{0,0}$to lie in the range [0,255] as

$$H_{0,0} = \left(H'_{0,0} + 1 << 4\right) >> 5 \quad (3)$$

In a similar way, the vertical half-pixel value $V_{0,0}$ is calculated from an intermediate value $V'_{0,0}$ using the six nearest integer pixel values located in the vertical direction as

$$V'_{0,0} = I_{-2,0} - 5I_{-1,0} + 20I_{0,0} + 20I_{1,0} - 5I_{2,0} + I_{3,0} \quad (4)$$

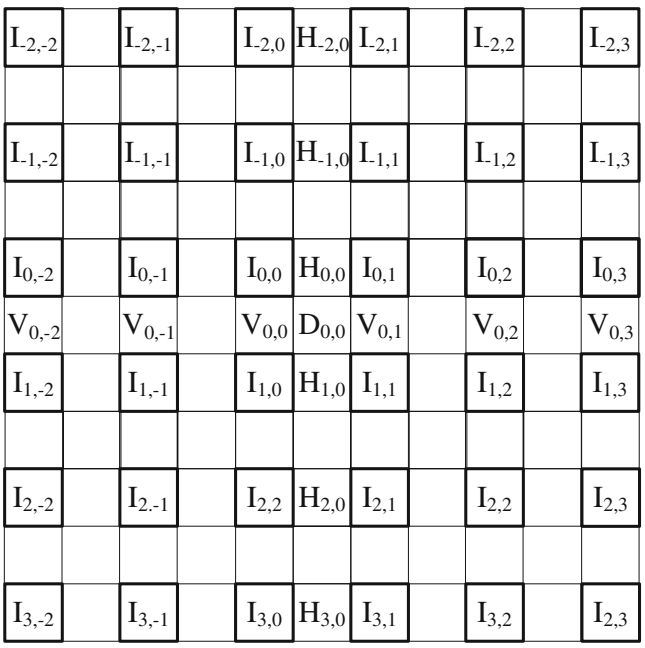

Figure 6 Spatial relationship of integer (I) and, half-pixel for vertical (V), horizontal (H) and diagonal (D) positions in the luminance interpolation.

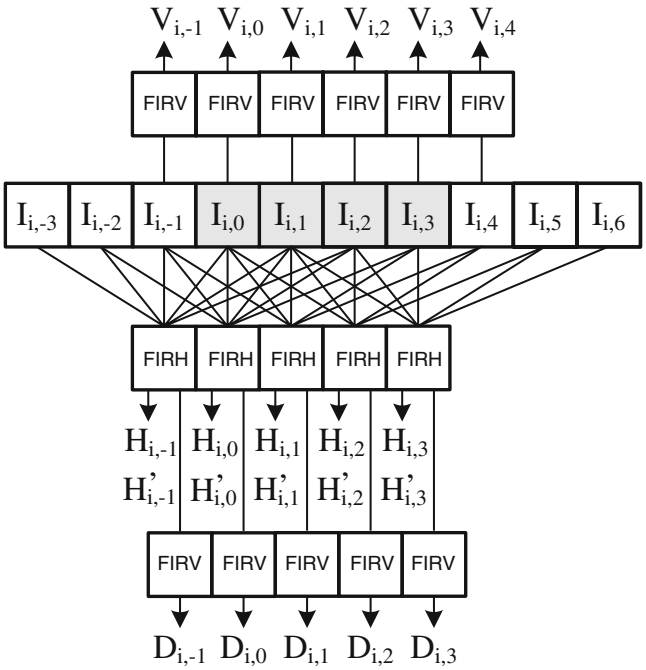

Figure 7 Architecture of half-pixel interpolator unit.

The half sample $V_{0,0}$ is calculated by clipping $V'_{0,0}$to the range [0,255] as

$$V_{0,0} = \left(V'_{0,0} + 1 << 4\right) >> 5 \quad (5)$$

The diagonal half-pixel value $D'_{0,0}$ is obtained from the six nearest intermediate horizontal values $H'_{i,j}$, or alternatively, vertical values $V'_{i,j}$, according to

$$D'_{0,0} = H'_{-2,0} - 5H'_{-1,0} + 20H'_{0,0} + 20H'_{1,0} - 5H'_{2,0} + H'_{3,0} \quad (6)$$

$$D'_{0,0} = V'_{0,-2} - 5V'_{0,-1} + 20V'_{0,0} + 20V'_{0,1} - 5V'_{0,2} + V'_{0,3} \quad (7)$$

The final value $D_{0,0}$ is computed as

$$D_{0,0} = \left(D'_{0,0} + 1 << 9\right) >> 10 \quad (8)$$

Figure 7 shows the proposed interpolator architecture based on a 2-D FIR approach which aims for high throughput and minimum latency. The 2-D FIR is decomposed into 1-D FIR horizontal (FIRH) and 1-D FIR vertical (FIRV) filters. Two parallel groups of FIRV and FIRH process the 10-pixel integer input data {I}. The first group generates the interpolated half-pixel vertical data {V} according to Eqs. 4 and 5. The second one generates the intermediated data {H′} according to Eq. 2 which are

clipped (Eq. 3) to compute the half-pixel horizontal data {H}. {H′} are processed in series in the FIRV filters to get the diagonal half-pixel data {D} which are implemented in Eqs. 6 and 8.

The half-pixel interpolation performed by the 6-tap Wiener FIR filter is implemented by shifters, additions and subtraction operations. Moreover, the symmetry of these filter coefficients can be exploited in order to reduce the number of operations. Taking into account that 5X=X+X<<2, and 20X=(X+X<<2)<<2, then Eq. 2 can be computed [26] as

$$H'_{0,0} = \left(\left(I_{0,0} + I_{0,1}\right) << 2 + \left(\left(I_{0,0} + I_{0,1}\right) - \left(I_{0,-1} + I_{0,2}\right)\right)\right) << 2 + \left(I_{0,-2} + I_{0,3}\right) - \left(I_{0,-1} + I_{0,2}\right) \tag{9}$$

A similar process can be applied to Eqs. 4, 6 and 7. Figure 8 shows the implementation of the proposed FIRH datapath according to the decomposition scheme in Eq. 9. It takes 6 integer input pixels and calculates the intermediate {H′} and clipped {H} horizontal pixels. The datapath is pipelined into 2 stages using registers to increase clock frequency and interpolation throughput. The input luminance integer pixels {I} are in the range [0,255] using an 8-bit accuracy. In the interpolation datapath, dynamic range of the intermediate results leads to modification of the bit widths required in the arithmetic computation to prevent overflow, they may even be negative. Although, in practice, the probability of overflow in intermediate data is low, the bus widths must be fixed to support the minimum and maximum value. Figure 8 also indicates bus width and the range of data in brackets at the output of every arithmetic element. In this notation, U stands for unsigned number and S signed number in two's complement. A 15-bit width in a signed representation for output {H′} is enough to deal with all possible values while the clipping circuit limits the {H} samples to the range [0,255] in an 8-bit unsigned representation.

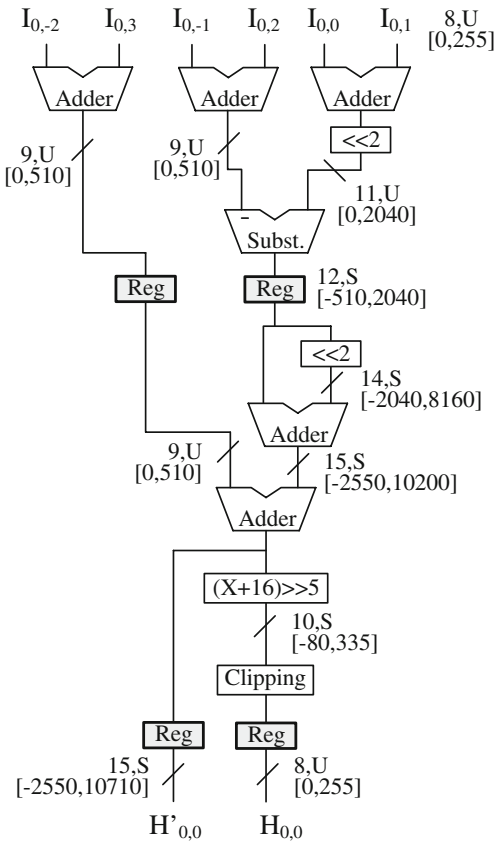

Figure 8 Half-pixel FIRH interpolation datapath. Notation used: bus width, U for unsigned or S for two's complement,[min value, max value].

Figure 9 shows the 5-stage pipeline datapath for the FIRV. There are two implementations of the same circuit depending on the bus width of the input datapath: unsigned 8-bit width for integer samples {I} and signed 15-bit width for intermediate data {H′}. New input data arrives in each clock cycle and the output {D or V} is computed after 5 clock cycles. In the worst case, intermediate data range is [−2550, 10455] with 15-bit width for {V} or [−214200, 475320] with 20-bit width for {D}.

The timing diagram of the half-pixel interpolation unit is shown in Fig. 10. The 10-pixel integer data {I} is input row by row. The interpolated samples, 6 pixels for {V} and 5 pixels for {H} and {D}, are computed with different latency: 2 clock cycles for {H}, 5 for {V} and 7 for {D}. {H} and {V} are directly generated from {I}, and {D} from the intermediate data {H′}.

4.2 Processing Unit (PU)

The PU computes for each subblock the Langragian cost of the half-pixel search. It is made up of 8 processing elements (PE), as shown in Fig. 11, operating in parallel in order to perform the eight half-pixel searches around the integer pixel. Each PE processes 4 interpolated half pixels and is composed of an absolute difference module, an adder tree and a final adder-accumulator.

The absolute difference module implements the absolute difference operation [27] expressed as

$$\left|R_{i,j} - C_{i,j}\right| = \begin{cases} R_{i,j} + \overline{C}_{i,j} + 1, & \text{if } R_{i,j} > C_{i,j} \\ \overline{R_{i,j} + \overline{C}_{i,j}}, & \text{if } R_{i,j} \le C_{i,j} \end{cases} \quad \forall j \in [0,3] \tag{10}$$

where, $R_{i,j} \in \{H_{i,j}, V_{i,j}, D_{i,j}\}$ represents the interpolated reference pixel and $C_{i,j}$ denotes the current pixel. To implement Eq. 10, a first level of adders compute $R_{i,j}+\overline{C}_{i,j}$ and the most significant bits of output $\{S_3,S_2,S_1,S_0\}$ are used to decide whether to invert the output through a bit-XOR or not. In Eq. 10, a 1 must be added if $R_{i,j}>C_{i,j}$, which

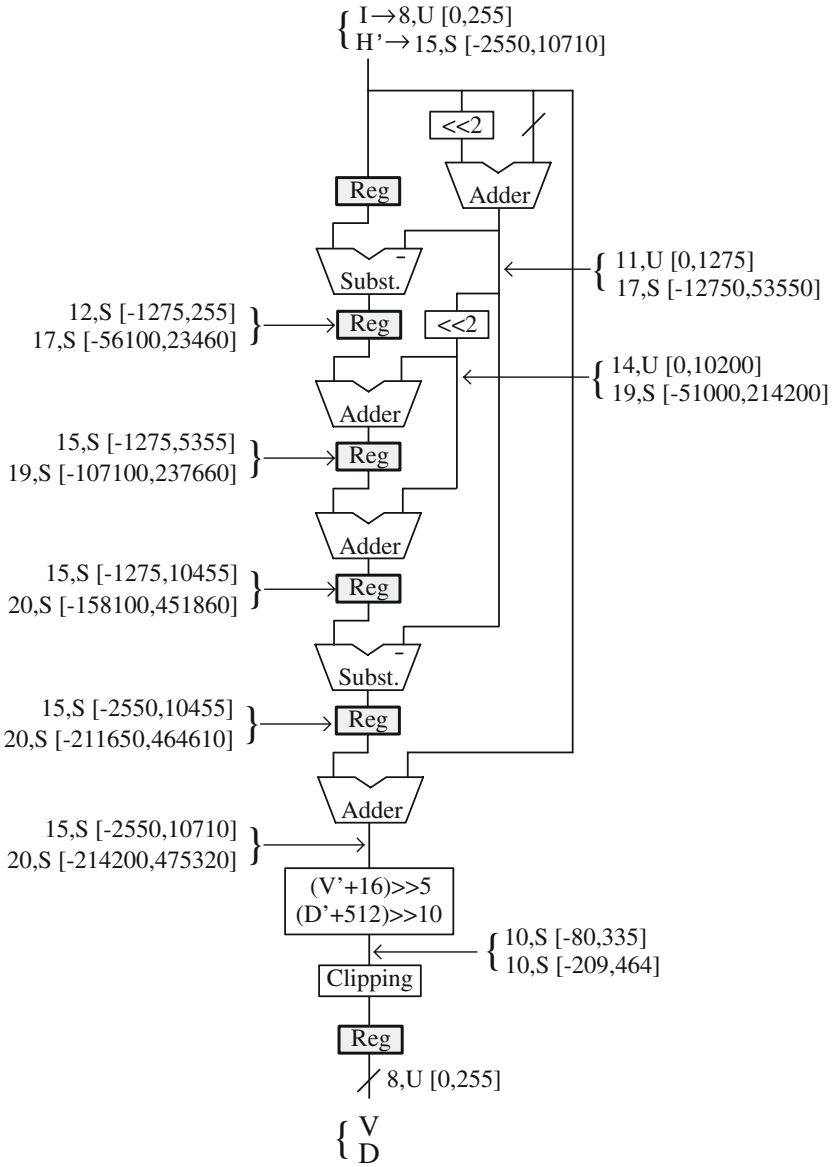

Figure 9 Half-pixel FIRV interpolation datapath. Notation used: bus width, U for unsigned or S for signed numbers in two's complement, [min value, max value].

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 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 total Langragian cost for a subblock, the register being initialized at λMVcost whose value has been calculated previously. Figure 11 also shows all bus widths to prevent overflow. In the worst case, the biggest SAD corresponds to the 16×16 partition where all absolute differences are 255, resulting in a maximum value of 255×16×16=6528, which can be represented by 16 bits. Taking into account that λMVcost has a 12-bit precision, the final Lagrangian cost of PE must be 17 bits.

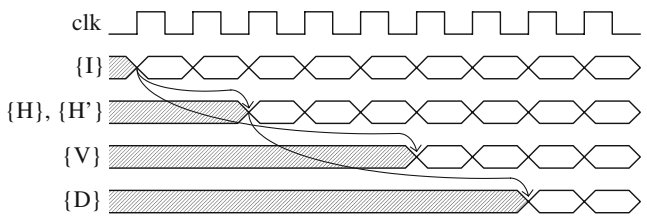

Figure 10 Half-quarter interpolation timing diagram.

In the PU, each PE is responsible for one search position around the integer sample: PE1 for (0,−1/2), PE2 for (0,1/2), PE3 for (−1/2,0), PE4 for (1/2,0), PE5 for (−1/2,−1/2), PE6 for (−1/2,1/2), PE7 for (1/2,−1/2) and PE8 for (1/2,1/2). However, the half interpolation unit generates the interpolat-

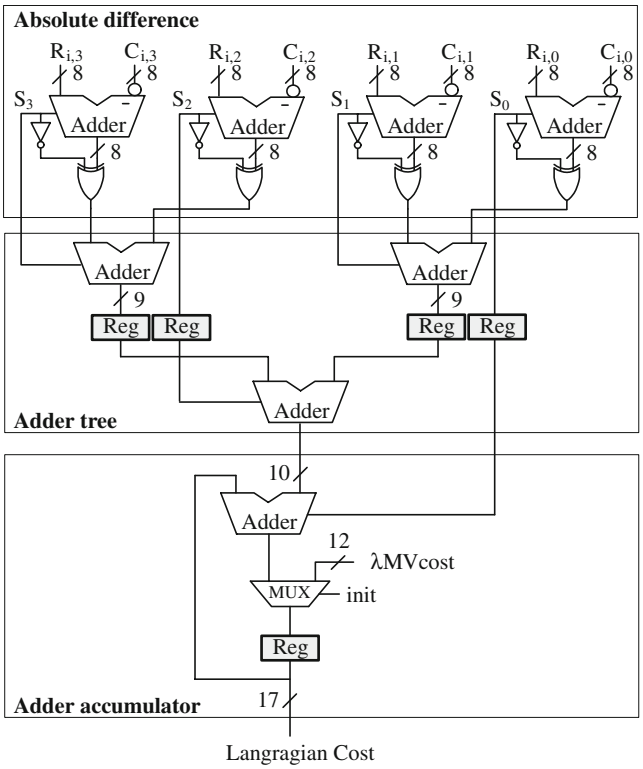

Figure 11 Processing element (PE) circuit.

ed samples with different latency so PEs must process the input data according to a data flow schedule. Figure 12.a) shows the distribution of interpolated half−pixels around the integer pixels; In order to simplify the data flow explanation, only a 4×4 subblock is considered. For this subblock, 5×4 horizontal samples {H}, 4×5 vertical samples {V} and 5×5 diagonal samples {D} are processed, which means a total of 65 samples,. Figure 12.b) shows the timetable scheduling used by PU to process the input samples arriving at different times specified in Fig. 10. In cycle 0, the five horizontal input samples $\{H_{0,-1}, H_{0,0}, H_{0,1}, H_{0,2}, H_{0,3}\}$ are processed, while $\{H_{0,-1}, H_{0,0}, H_{0,1}, H_{0,2}\}$ in PE1 and $\{H_{0,0}, H_{0,1}, H_{0,2}, H_{0,3}\}$ are processed in PE2. The notation $\left|H_{i,j} - C_{i,j}\right|$ means $\sum_{j=0}^{4} \left|H_{i,j} - C_{i,j}\right|$ and this arithmetic operation implemented by the circuit in Fig. 11 takes two clock cycles to be done. In cycle 4, the last row $\{H_{3,-1}, H_{3,0}, H_{3,1}, H_{3,2}, H_{3,3}\}$ allows the Lagrangian cost to be computed after a latency of two clock cycles. PE3 starts off when the four vertical samples $\{V_{-1,0}, V_{-1,1}, V_{-1,2}, V_{-1,3}\}$ are input and PE4 begins one clock cycle later with the following row $\{V_{0,0}, V_{0,1}, V_{0,2}, V_{0,3}\}$. The Lagrangian cost of PE3 and PE4 are generated at cycle 9 and 10 respectively. Likewise, in cycle 6 PE5 and PE6 begin to process the five diagonal samples $\{D_{-1,-1}, D_{-1,0}, D_{-1,1}, D_{-1,2}, D_{-1,3}, D_{-1,4}\}$ and one clock cycle later PE7 and PE8 with the following row. The Lagrangian costs are generated in cycle 11 for PE5 and PE6, and in cycle 12 for PE7 and PE8. All Lagrangian costs generated in PU lead to the best half-MV unit.

The processing time of the PU of a 4×4 subblock takes 10 clock cycles and is limited by the 10 integer input data of 10 pixels each used in the half-pixel interpolation unit. As a result, there are 6 idle clock cycles in the PE. In general, there are 6 idle clock cycles during the processing of each vertical column belonging to a subblock. As a result, the number of idle clock cycles for each subblock are: 6 for 4×4 and 8×4, 12 for 4×8, 8×8 and 16×8, and 24 for 8×16 and 16×16.

4.3 Best Half-MV Unit

The Best half-MV unit finds the best half-MV by searching for the minimum Lagrangian cost generated in PU. Figure 13 shows the schematic of this circuit made up of two comparators and a register which stores the minimum Lagrangian cost and its corresponding best half MV. This register is initialized by the data from the IME with the Lagrangian cost and MV for the best integer position. The Best half-MV unit processes in parallel two Lagrangian costs generated in the PU; to maintain the regularity, the cost of PE3 is delayed 1 clock cycle to coincide in time with the cost of PE4. The register stores new data whether a minimum Lagrangian cost is found or not. This process finishes when all Lagrangian costs are compared. As a result, the data stored in the register, Lagrangian cost and half-MV, are passed to the quarter-pixel processor.

4.4 RAM1 Memory

A good compromise between the memory usage and computational complexity is to interpolate the half-pixel values and to store all of them in a memory to be computed in the quarter processor when they are needed. In RAM1 the pixel values {I}, {H}, {V} and {D} generated in the half-pixel interpolation unit are stored in a bank of four double port RAMs. For the sake of clarity, Fig. 14 shows the distribution in RAM1 of pixels for a 4×4 subblock. The white core contains the pixels used in the half-pixel processor and the pixels in the grey frame are only used by the bilinear filters in the quarter interpolation. To simplify the quarter interpolation, the pixels are stored row by row using words of 6 pixels for {I} and {V} and words of 5 pixels for {H} and {D}. Thus, RAM1 must be able to store the interpolated pixels for a 16×16 block. In this case, the block is split up in 4 rows of 16+1+1 elements each for {I} and {H}, which implies 18×4×6 bytes for {I} and 18×4×5 bytes for {H}. The number of elements for {V} and {D} in each row is lower resulting in 17×4×6 bytes for {V} and 17×4×5 bytes for {D}. The total size of RAM1 is 1540 bytes.

Figure 12 Data flow schedule of PU for a 4×4 subblock: a) Half-pixel interpolation, b) timing diagram.

$D_{-1,-1}$	$V_{-1,0}$	$D_{-1,0}$	$V_{-1,1}$	$D_{-1,1}$	$V_{-1,2}$	$D_{-1,2}$	$V_{-1,3}$	$D_{-1,3}$
$H_{0,-1}$	$\mathbf{I_{0,0}}$	$H_{0,0}$	$\mathbf{I_{0,1}}$	$H_{0,1}$	$\mathbf{I_{0,2}}$	$H_{0,2}$	$\mathbf{I_{0,3}}$	$H_{0,3}$
$D_{0,-1}$	$V_{0,0}$	$D_{0,0}$	$V_{0,1}$	$D_{0,1}$	$V_{0,2}$	$D_{0,2}$	$V_{0,3}$	$D_{0,3}$
$H_{1,-1}$	$\mathbf{I_{1,0}}$	$H_{1,0}$	$\mathbf{I_{1,1}}$	$H_{1,1}$	$\mathbf{I_{1,2}}$	$H_{1,2}$	$\mathbf{I_{1,3}}$	$H_{1,3}$
$D_{1,-1}$	$V_{1,0}$	$D_{1,0}$	$V_{1,1}$	$D_{1,1}$	$V_{1,2}$	$D_{1,2}$	$V_{1,3}$	$D_{1,3}$
$H_{2,-1}$	$\mathbf{I_{2,0}}$	$H_{2,0}$	$\mathbf{I_{2,1}}$	$H_{2,1}$	$\mathbf{I_{2,2}}$	$H_{2,2}$	$\mathbf{I_{2,3}}$	$H_{2,3}$
$D_{2,-1}$	$V_{2,0}$	$D_{2,0}$	$V_{2,1}$	$D_{2,1}$	$V_{2,2}$	$D_{2,2}$	$V_{2,3}$	$D_{2,3}$
$H_{3,-1}$	$\mathbf{I_{3,0}}$	$H_{3,0}$	$\mathbf{I_{3,1}}$	$H_{3,1}$	$\mathbf{I_{3,2}}$	$H_{3,2}$	$\mathbf{I_{3,3}}$	$H_{3,3}$
$D_{3,-1}$	$V_{3,0}$	$D_{3,0}$	$V_{3,1}$	$D_{3,1}$	$V_{3,2}$	$D_{3,2}$	$V_{3,3}$	$D_{3,3}$

a)

CLK	PE1	PE2	PE3	PE4	PE5	PE6	PE7	PE8	COST
1	$\lvert H_{0,j}-C_{0,j}\rvert$	$\lvert H_{0,j+1}-C_{0,j}\rvert$							PE5 PE6
2	$\lvert H_{1,j}-C_{1,j}\rvert$	$\lvert H_{1,j+1}-C_{1,j}\rvert$							PE7 PE8
3	$\lvert H_{2,j}-C_{2,j}\rvert$	$\lvert H_{2,j+1}-C_{2,j}\rvert$			Previous subbblock				
4	$\lvert H_{3,j}-C_{3,j}\rvert$	$\lvert H_{3,j+1}-C_{3,j}\rvert$	$\lvert V_{-1,j}-C_{0,j}\rvert$						
5			$\lvert V_{0,j}-C_{1,j}\rvert$	$\lvert V_{0,j}-C_{0,j}\rvert$					
6			$\lvert V_{1,j}-C_{2,j}\rvert$	$\lvert V_{1,j}-C_{1,j}\rvert$	$\lvert D_{-1,j}-C_{0,j}\rvert$	$\lvert D_{-1,j+1}-C_{0,j}\rvert$			PE1 PE2
7			$\lvert V_{2,j}-C_{3,j}\rvert$	$\lvert V_{2,j}-C_{2,j}\rvert$	$\lvert D_{0,j}-C_{1,j}\rvert$	$\lvert D_{0,j+1}-C_{1,j}\rvert$	$\lvert D_{0,j}-C_{0,j}\rvert$	$\lvert D_{0,j+1}-C_{0,j}\rvert$	
8				$\lvert V_{3,j}-C_{3,j}\rvert$	$\lvert D_{1,j}-C_{2,j}\rvert$	$\lvert D_{1,j+1}-C_{2,j}\rvert$	$\lvert D_{1,j}-C_{1,j}\rvert$	$\lvert D_{1,j+1}-C_{1,j}\rvert$	
9					$\lvert D_{2,j}-C_{3,j}\rvert$	$\lvert D_{2,j+1}-C_{3,j}\rvert$	$\lvert D_{2,j}-C_{2,j}\rvert$	$\lvert D_{2,j+1}-C_{2,j}\rvert$	PE3
10	Next subblock						$\lvert D_{3,j}-C_{3,j}\rvert$	$\lvert D_{3,j+1}-C_{3,j}\rvert$	PE4
11	$\lvert H_{0,j}-C_{0,j}\rvert$	$\lvert H_{0,j+1}-C_{0,j}\rvert$							PE5 PE6
12	$\lvert H_{1,j}-C_{1,j}\rvert$	$\lvert H_{1,j+1}-C_{1,j}\rvert$							PE7 PE8
13	$\lvert H_{2,j}-C_{2,j}\rvert$	$\lvert H_{2,j+1}-C_{2,j}\rvert$							

b)

5 Quarter-Pixel Processor

The quarter-pixel processor performs quarter -pixel interpolation and a quarter-pixel search for each subblock size. Once the best half-pixel search is completed, the quarter pixel values are computed around it by bilinear filters according to the scheme shown in Fig. 15. Here, quarter pixels are indicated in circles and half and integer pixels in squares. The half-pixel processor selects 1 out of 9 possible options: $D_{-1,-1}$, $V_{-1,0}$, $D_{-1,0}$, $H_{0,-1}$, $I_{0,0}$, $H_{0,0}$, $D_{0,-1}$, $V_{0,0}$ and $D_{0,0}$. Thus, only 8 quarter pixels around the best half-pixel selection must be interpolated into quarter-pixel resolution.

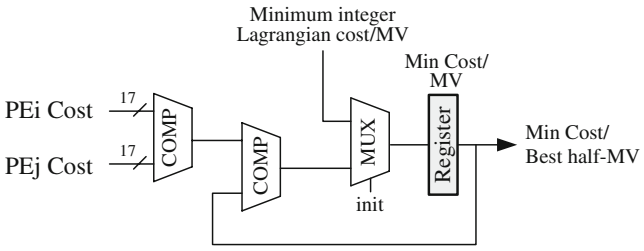

Figure 13 Schematic of best half-MV unit.

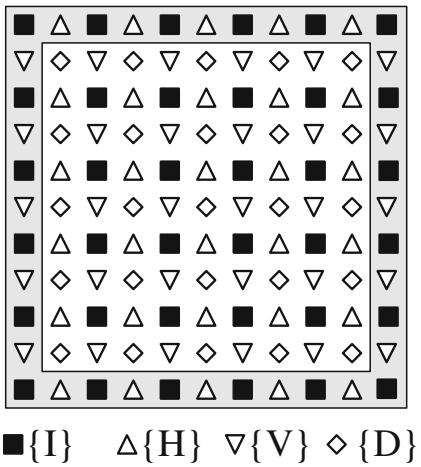

Figure 14 Distribution of pixels in RAM1 for a 4×4 subblock.

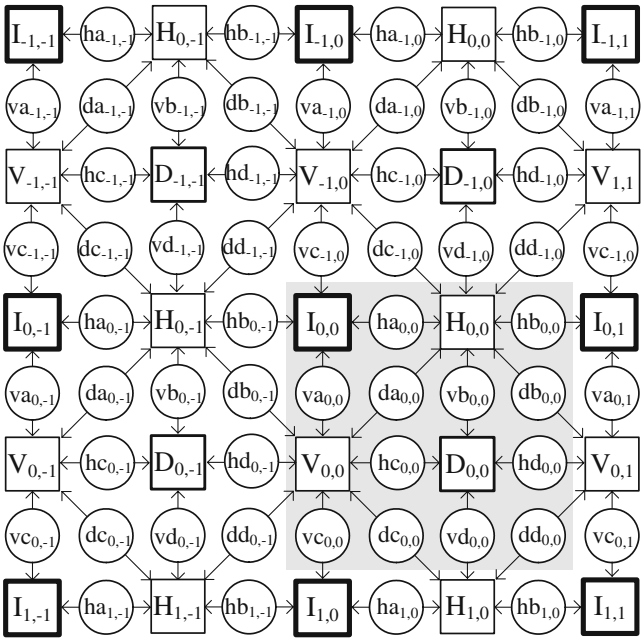

Figure 15 Interpolation of quarter-pixel samples (shown in circles) by bi-linear filters centred around integer sample $I_{0,0}$.

In the grey box in Fig. 15, twelve different types of quarter pixels are highlighted, classified as horizontal {ha, hb, hc, hd}, vertical {va, vb, vc, vd} and diagonal {da, db, dc, dd}, according to the direction used in their generation from half and integer pixels. The four quarter pixels in the horizontal direction are defined as

$$ha_{0,0} = \left(I_{0,0} + H_{0,0} + 1\right) >> 1 \tag{11}$$

$$hb_{0,0} = \left(H_{0,0} + I_{0,1} + 1\right) >> 1 \tag{12}$$

$$hc_{0,0} = \left(V_{0,0} + D_{0,0} + 1\right) >> 1 \tag{13}$$

$$hd_{0,0} = \left(D_{0,0} + V_{0,1} + 1\right) >> 1 \tag{14}$$

The four quarter-pixel values in the vertical direction are defined as

$$va_{0,0} = \left(I_{0,0} + V_{0,0} + 1\right) >> 1 \tag{15}$$

$$vb_{0,0} = \left(H_{0,0} + D_{0,0} + 1\right) >> 1 \tag{16}$$

$$vc_{0,0} = \left(V_{0,0} + I_{1,0} + 1\right) >> 1 \tag{17}$$

$$vd_{0,0} = \left(D_{0,0} + H_{1,0} + 1\right) >> 1 \tag{18}$$

Finally, the four quarter-pixel values in the diagonal direction are defined as

$$da_{0,0} = \left(H_{0,0} + V_{0,0} + 1\right) >> 1 \tag{19}$$

$$db_{0,0} = \left(H_{0,0} + V_{0,1} + 1\right) >> 1 \tag{20}$$

$$dc_{0,0} = \left(V_{0,0} + H_{1,0} + 1\right) >> 1 \tag{21}$$

$$dd_{0,0} = \left(V_{0,1} + H_{1,0} + 1\right) >> 1 \tag{22}$$

Figure 16 illustrates the scheme of the quarter-pixel interpolation unit. During each clock cycle, 22 input pixels {I}, {H}, {V} and {D} are read in parallel from RAM1 row by row in vertical order, which are selected according to best half-pixel MVs. If the best half-pixel is {V} or {D}, then two adjacent rows of {I} and {H} samples are necessary to perform the quarter-pixel interpolation. Otherwise, if the best half-pixel is {I} or {H} then two adjacent rows of {V} and {D} samples are used. Two multiplexers perform this kind of selection, storing the two adjacent rows in registers REG1 and REG2 and the best half-pixel in REG3. The 33 pixels of these registers run into the bilinear filter array arranged in 8 blocks of 4 basic elements belonging to the same type of quarter pixel. Each basic bilinear filter is implemented by means of the optimized scheme of Fig. 17 where a 7-bit adder is used instead of the 8-bit adder used in a classic implementation. The rounding

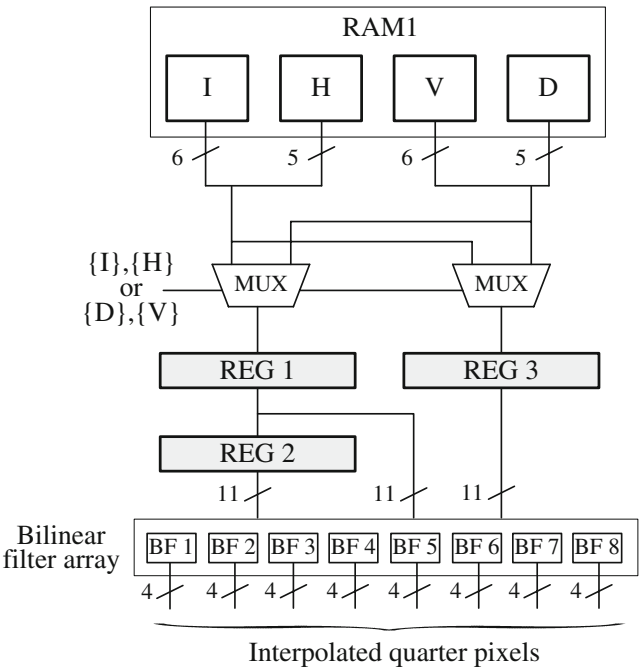

Figure 16 Architecture of quarter-pixel interpolator unit.

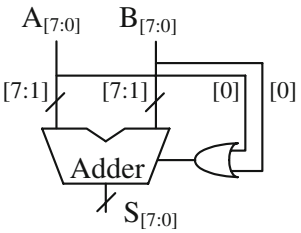

Figure 17 Bilinear filter implementation.

operation defined in Eqs. 11 to 22 is performed by an OR logic gate of less significant bits labelled as $A_{[0]}$ and $B_{[0]}$, which act as carry-in. The result S is the 8-bit interpolated quarter pixel.

The 8 outputs of 4 pixels each generated by the quarter-pixel interpolation unit are stored in RAM2 and processed in the PU to compute the Lagrangian cost and the best quarter MVs. RAM2 has been split into a bank of three double port SRAMs due to limitations in the input bandwidth of the technology. Each RAM has a capacity to store 64×12 pixels, the full capacity of RAM2 being 2304 bytes.

The PU and best quarter MV units are similar to those described in the half-pixel processor. Here, each PE is responsible for one search position around the integer/half sample: PE1 for (x,y-1/4), PE2 for (x,y+1/4), PE3 for (x-1/4,y), PE4 for (x+1/4,y), PE5 for (x-1/4,y-1/4), PE6 for (x-1/4,y+1/4), PE7 for (x+1/4,y-1/4) and PE8 for (x+1/4,y+1/4), where $x,y \in \{0, \pm 1/2\}$. Unlike the former processor, eight Lagrangian costs are computed in parallel so all quarter interpolated samples are generated at same time. Thus, the best quarter-MV unit shown in Fig. 18 performs a parallel comparison of these costs using binary tree architecture of comparators in order to obtain the minimum value. This value is finally compared with the minimum Lagrangian cost derived in the half pixel processor. Here, a pipeline stage has been inserted to reduce the critical path. As a result, only the best quarter-MV is stored in the output register which will be used by the mode decision processor.

6 Mode Decision Processor

In the inter prediction for the H.264, the MBs are processed in a specific order to ensure that in ME all MVpreds neighbouring each MB are available. Moreover, the tree-structured motion compensation method enables a 16×16 MB to be 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 includes modes of 8×8, 8×4, 4×8, and 4×4 (see Fig. 1). The mode decision processor uses the best quarter-MVs obtained in quarter-pixel processor as input. It selects the best mode by comparing the total minimum Langrangian cost of all the subblocks belonging to a mode (J_{Mode}). The final best mode decision is worked out once all 41 modes have been processed. As a result, the mode decision processor obtains the best MVs and mode, and generates the residual image to be coded. In this process, the memory RAM3 temporally stores the best subblock candidates and it is split into two halves: The first one contains the best candidate in the first level hierarchy and the second one contains the best candidate in the second level.

The mode decision processor starts off when the quarter-pixel processor has just finished with the first 16×16 MB and the best quarter-MV is obtained. The mode decision method consists of the following steps according to level of hierarchy.

- **First level**. Modes 16×16, 16×8 and 8×16. Only the first half of RAM3 is used.

  Step 1: Initial best mode is 16×16. The interpolated 16×16 MB defined by best quarter-MV (lowest value of $J_{16\times16}$) is stored in RAM3.
  Step 2: If $J_{8\times16}$(subblock 0)+$J_{8\times16}$(subblock 1)<$J_{16\times16}$, then the best mode is 8×16 and both 8×16 subblocks are stored in RAM3.

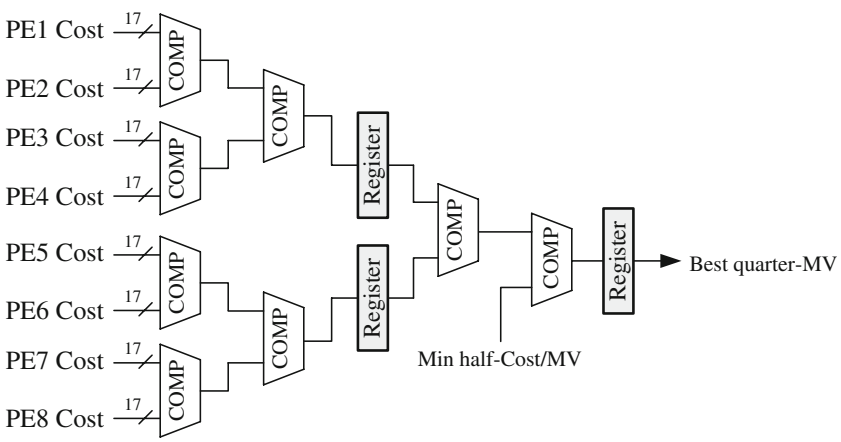

Figure 18 Schematic of best quarter-MV unit.

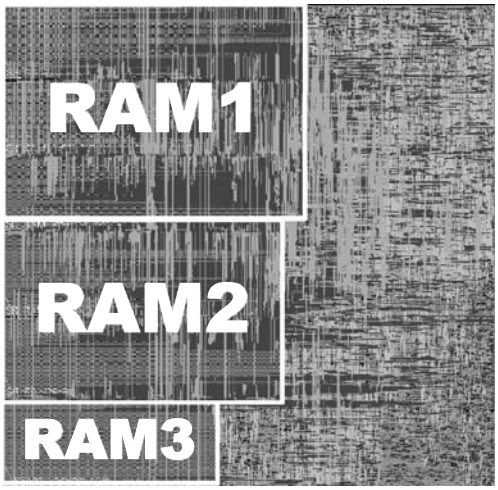

Figure 19 Lay-out of proposed FME with Faraday 180 nm CMOS technology.

Step 3: If $J_{16\times8}(0)+J_{16\times8}(1)<J_{(\text{best mode step 2})}$, then the best mode is 16×8 and both 16×8 subblocks are stored in RAM3.

- **Second level**. Modes 8×8, 8×4, 4×8, and 4×4. Repeated for each four 8×8 subblocks. Only the second half of RAM3 is used, which is split into four parts each one managed by an 8×8 subblock.

  Step 4: The initial best mode is 8×8. The interpolated 8×8 subblock defined by the best quarter-MV (lowest value of $J_{8\times8}$) is stored in RAM3.
  Step 5: If $J_{4\times8}(0)+J_{4\times8}(1)<J_{8\times8}$, then the best mode is 4×8 and both 4×8 subblocks are stored in RAM3.
  Step 6: If $J_{8\times4}(0)+J_{8\times4}(1)<J_{(\text{best mode step 5})}$, then the best mode is 8×4 and both 8×4 subblocks are stored in RAM3.
  Step 7: If $J_{4\times4}(0)+J_{4\times4}(1)+J_{4\times4}(3)+J_{4\times4}(4)<J_{(\text{best mode step 6})}$, then the best mode is 4×4 and all 4×4 subblocks are stored in RAM3.

- **Final decision**. The best mode for candidates in first and second level are compared searching for the minimum J_{Mode}, then the best mode and MVs are found. The residual image is computed by substracting the best interpolated image and the current image.

7 Implementation Results

The proposed FME architecture has been designed aiming for regular flow and efficient hardware utilization. This circuit has been implemented in Verilog VHDL at RTL level and it has been synthesized using a TSMC 180 nm CMOS library. The final layout is shown is Fig. 19 and its area is 1.2×1.1 mm^2. It uses 11.4 k gates and a total of 4356 Bytes in different RAMs. In typical working conditions (1.8 V, 25°C), the maximum frequency of 290 MHz can be achieved including wire delays. It takes a total of 870 clock cycles (832 for input data and 38 of latency) to process a MB and an additional 66 clock cycles to generate the residual image. It can provide enough processing capacity for 1920×1088@30fps real-time video streams.

Table 2 shows the hardware comparison of the FME designs implemented in a similar technology. Our design generates the residual image and the best MVs with a high throughput and low area cost architecture. Part of this hardware reduction is achieved using SAD as a distortion measure instead of the sum of absolute transformed differences (SATD) implemented in [16–18, 20]. The designs [16, 20] and ours use the same input bandwidth of 10 pixels. However, our design roughly multiplies by three the operating frequency and the reduction of latency is enough to process 1080p@30fps videos. In [17], the input bandwidth is increased to 22 pixels to remove all the redundant columns by adopting a 16-pixel interpolation unit. It operates at the same frequency as ours with a slight reduction in latency and a bigger area cost (2.7 mm^2 in comparison with 1.32 mm^2). Moreover, it does not explain whether the best interpolated MB is stored or not. A very different scheme is used in the design presented in [18] which is focused on low hardware cost but with a great latency. Finally, the three-engine architecture [19] operates at 150 MHz and it takes 616 clock cycles to process a MB.

Table 2 Comparison of the FME with other designs.

Ref.	[16]	[17]	[18]	[20]	[19]	Ours
Tech. (μm)	UMC 0.18	TSMC 0.18	TSMC 0.18	UMC 0.18	TSMC 0.13	UMC 0.18
Freq. (MHz)	100	285	274	100	150	290
Gate Count	79 k	188 k	24 k	48 k	188 k	11.4 k
RAM	No	No	1904 bits	No	9724 Bytes	4356 Bytes
Area (mm^2)	NA	1.8×1.5	0.58×0.66	NA	NA	1.2×1.1
Resolution	720×576	1920×1088	NA	720×576	1920×1088	1920×1088
Input pixels	10	22	1	10	NA	10
Latency	1648	790	39551	2000	616	870

Each engine processes different kinds of subblocks in a pipelined way to increase throughput. However, it requires more gates and RAM than ours and its area is much bigger even using a better technology.

8 Conclusions

In this paper, we propose a high performance VLSI architecture for FME in H.264/AVC with enough processing capacity for 1080HD real-time video streams. This architecture is made up of three different pipelined processors to provide a trade-off between processing time and hardware utilization. These processors implement a completely standard-compatible full-search algorithm and are capable of processing a macroblock (MB) in 870 clock cycles using 4-pixel interpolation units with a 10-pixel input bandwidth of reference pixels. Our design is implemented with only 11.4 k gates and 4.4kBytes of RAM in a standard 180 nm CMOS technology at an operating frequency of 290 MHz. Compared with previous works, it presents a high throughput and low area cost architecture, which can generate the residual image and the best MVs ready to be encoded.

Acknowledgment We wish to acknowledge the Spanish Ministry of Education and Science for the financial help TEC2006-12438/TCM received to support this work.

References

1. Ostermann, J., Bormans, J., List, P., Marpe, D., Narroschke, M., Pereira, F., et al. (2004). Video coding with H.264/AVC: tools, performance, and complexity. *IEEE Circuits and Systems Magazine, 4*(1), 7–28. First Quarter.
2. Wiegand, T., Sullivan, G. J., Bjontegaard, G., & Luthra, A. (2003). Overview of H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology, 13* (7), 560–576.
3. ITU-T Rec. H.264/ISO/IEC 11496-10 (2003). Advanced Video Coding. Final Committee Draft, Document JVTG050.
4. Online document. http://iphome.hhi.de/suehring/tml/. Accessed 17 September 2009.
5. Yap, S. Y., & McCanny, J. V. (1989). A VLSI architecture for variable block size video motion estimation. *IEEE Transactions on Circuits and Systems for Video Technology, 36*(2), 1301–1308.
6. Komarek, T., & Pirsh, P. (2006). Array architectures for block matching algorithms. *IEEE Transactions on Circuits and Systems for Video Technology, 16*(7), 876–883.
7. Jong, H. M., Chen, L. G., & Chiueh, T. D. (1994). Parallel architecture for 3-step hierarchical search block-matching algorithm. *IEEE Transactions on Circuits and Systems for Video Technology, 4*(4), 407–416.
8. Zhu, C., Lin, X., & Chau, L. P. (2002). Hexagon-based search pattern for fast block motion estimation. *IEEE Transactions on Circuits and Systems for Video Technology, 12*(5), 349–355.
9. Zhu, C., & Ma, K. K. (2000). A new diamond search algorithm, for fast block matching motion estimation. *IEEE Transactions on Image Processing, 9*(2), 287–290.
10. Chen, T. C., Chen, Y. H., Tsai, S. F., Chien, S. I., & Chen, L. G. (2007). 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), 568–577.
11. Li, D. X., & Zhang, M. (2007). Architecture design for H.264/AVC integer motion estimation with minimum memory bandwidth. *IEEE Transactions on Consumer Electronics, 53*(3), 1053–1060.
12. Zhenyu, W., Baochen, J., Xudong, Z., & Yu, C. (2004). A new full-pixel and sub-pixel motion vector search algorithm for fast block-matching motion estimation in H.264. *Proceedings of the Third International Conference on Image and Graphic*s, 345–348.
13. La, B., Eom, M., & Choe, Y. (2007). Fast sub-pixel search control by using neighbour motion vector in H.264. *9th International Conference on Advanced Communication Technology, 1*, 62–65.
14. Hyun, C. J., Kim, S. D., & Sunwoo, M. H. (2006). Efficient memory reuse and sub-pixel interpolation algorithms for ME/MC of H.264/AVC. *IEEE Workshop on Signal Processing Systems Design and Implementation*, 377–382. October.
15. Song, Y., Ma, Y., Liu, Z., Ikenaga, T., & Goto, S. (2008). Hardware-oriented direction-based fast fractional motion estimation algorithm in H.264/AVC. *IEEE International Conference on Multimedia and Expo*, 1009–1012, June.
16. Chen, T. C., Huang, Y. W., & Chen, L. G. (2004). Fully utilized and reusable architecture for fractional motion estimation of H.264/AVC. *IEEE International Conference on Acoustics, Speech, and Signal Processing, 5*, 9–12.
17. Yang, C., Goto, S., Ikenaga, T. (2006). High performance VLSI architecture of fractional motion estimation in H.264 for HDTV. *IEEE International Symposium on Circuits and Systems*, 2605–2608.
18. Song, Y., Liu, Z., Goto, S., & Ikenaga, T. (2005). A VLSI architecture for Motion compensation interpolation in H.264/AVC. *6th International Conference on ASIC*, 279–282. October.
19. Wu, C. L., Kao, C. Y., & Lin, Y. L. (2008). A high performance three-engine architecture for H.264/AVC fractional motion estimation. *IEEE International Conference on Multimedia and Expo*, 133–136.
20. Wang, Y. J., Cheng, C. C., & Chang, T. S. (2007). A fast algorithm and its VLSI architecture for fractional motion estimation for H.264/MPEG-4 AVC video coding. *IEEE Transactions on Circuits and Systems for Video Technology, 17*(5), 578–583.
21. Lin, Y. K., Lin, C. C., Kuo, T. Y., & Chang, T. S. (2008). A hardware-efficient H.264/AVC motion-estimation design for high-definition video. *IEEE Transactions on Circuits and Systems, 55* (6), 1526–1535.
22. Yalcin, S., & Hamzaoglu, I. (2006). A high performance hardware architecture for half-pixel accurate H.264 motion estimation. *IFIP International Conference on Very Large Scale Integration*, 63–67. October.
23. Rahman, C. A. & Badawy, W. (2005). A quarter pel full search block motion estimation architecture for H.264/AVC. *IEEE International Conference on Multimedia and Expo*, 414–417. July.
24. Chen, T. C., Chien, S. Y., Huang, Y. W., Tsai, C. H., Chen, C. Y., Chen, T. W., et al. (2006). 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), 673–688.
25. Huang, Y. W., Chen, T. C., Tsai, C. H., Chen, C. Y., Chen, T. W., Chen, C. S., et al. (2005). A 1.3TOPS H.264/AVC Single-Chip Encoder for HDTV Applications. *ISSCC Digest of Technical Paper*, 128–129. February.
26. Sihvo, T., & Niittylahti, J. (2005). H.264/AVC interpolation optimization. *IEEE Workshop on Signal Processing Systems Design and Implementation*, 307–312. November.
27. Vanne, J., Ahn, E., Hämäläinen, T. D., & Kuusilinna, K. (2006). A high-performance sum of absolute difference implementation for motion estimation. *IEEE Transactions on Circuits and Systems for Video Technology, 16*(7), 876–883.

Gustavo A. Ruiz was born in Burgos, Spain, in 1962. He received the M.Sc. degree in physics in 1985 from the University of Navarra, Spain, and the Ph.D. degree in physical science in 1989 from the University of Cantabria, Santander, Spain. Since 1985, he has been with the Department of Electronics and Computers at the University of Cantabria, where he is currently an Associate Professor. His current research interests are mainly focused on VLSI architectures for signal processing and high-speed arithmetic circuits.

Juan A. Michell was born in Cáceres, Spain, in 1952. He received the M.S. and the Ph.D. degrees in physical sciences from the University of Cantabria, Spain, in 1974 and 1977, respectively. Since 1974 he has been with the Department of Electronics and Computers at the University of Cantabria, where he was appointed Professor in Electronics in 1991. His current research interests are VLSI architectures and integrated circuit design for digital signal processing applications.