HAL
open science

A flexible heterogeneous hardware/software solution for real-time high-definition H.264 motion estimation

Fabrice Urban, Ronan Poullaouec, Jean François Nezan, Olivier Déforges

To cite this version:

Fabrice Urban, Ronan Poullaouec, Jean François Nezan, Olivier Déforges. A flexible heterogeneous hardware/software solution for real-time high-definition H.264 motion estimation. IEEE Transactions on Circuits and Systems for Video Technology, 2008, 18 (12). hal-00336473

HAL Id: hal-00336473

https://hal.science/hal-00336473v1

Submitted on 4 Nov 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A flexible heterogeneous hardware/software solution for real-time high-definition H.264 motion estimation

Fabrice URBAN, Ronan POULLAOUEC, Jean-François NEZAN, Olivier DEFORGES

***Abstract*—The MPEG-4 AVC/H.264 video compression standard introduces a high degree of motion estimation complexity. Quarter-pixel accuracy and variable block-size significantly enhance compression performances over previous standards, but increase computation requirements. Firstly, a DSP-based solution achieves real-time integer motion estimation. Nevertheless, fractional-pixel refinement is too computationally intensive to be efficiently processed on a software-based processor. Secondly, to address this restriction, a flexible and low complexity VLSI sub-pixel refinement coprocessor is designed. Thanks to an improved datapath, a high throughput is achieved with low logic resources. Finally, we propose a heterogeneous (DSP-FPGA) solution to handle real-time motion estimation with variable block-size and fractional-pixel accuracy for high-definition video. It combines efficiency and programmability. The flexibility offers complexity versus performance trade-offs. The system achieves motion estimation of 720p sequences at up to 60 frames per second.**

***Index Terms*—Field programmable gate arrays, Digital signal processors, Motion estimation, Parallel processing, Real-time, H.264**

I. Introduction

Motion estimation is known to be a key operation in video encoders. It is aimed at removing temporal redundancies in video sequences. The MPEG-4 AVC/H.264 video encoding standard reduces bit-rates by up to 50% compared to MPEG-2 and is the preferred standard for video compression. The high performance of H.264 is mainly due to improved motion compensation modes such as variable block-size motion compensation, quarter-pixel accuracy and multiple reference pictures [1]. However the introduction of numerous modes raises the complexity of the codec [2], [3] and makes real-time H.264 compression challenging, especially for high-definition video. Motion estimation is the most computationally intensive task of the video encoder, reaching up to 60% of the computation load of a H.264 video encoder [4].

To overcome the high processing complexity, specific circuits (ASIC: Application Specific Integrated Circuit) are designed or programmable VLSI (FPGA: Field Programmable Gate Array) is used. Computation power constraints are met thanks to high logic integration. However, from previous work described in the literature [5], [6], [7], [8], [9], none address together fractional-pixel accuracy, variable bloc-size, high-definition, and low complexity. Moreover, these hardware components provide a very low level of flexibility and require a long development time.

Programmable devices (DSP: Digital Signal Processors) provide the high flexibility of software processors but have reduced performances compared to hardware components [10]. The latest DSPs achieve high performances, but are not able to perform a motion estimator with quarter-pixel accuracy for high definition video in real-time. Although the search range of fractional-pixel refinement is small, it is the most computationally intensive part of motion estimation mainly because of interpolation and the algorithm is quite regular. Therefore it is a good candidate for execution on a hardware accelerator.

Our work proposes an efficient mixed hardware/software solution to address the challenging constraints of H.264 motion estimation. Low to medium complexity operations are handled by a DSP and an FPGA accelerates computationally intensive parts. Consequently this heterogeneous multi-component approach combines flexibility and computation efficiency. A modified distribution and scheduling of the operations allows computation parallelism. A fast Integer Motion Estimation (IME) scheme has been optimized to be executed on the DSP, and a low-complexity and low memory bandwidth VLSI coprocessor has been designed for Fractional-pixel Motion Estimation (FME). The VLSI design is scalable to match logic resource utilization with performance requirements, and supports variable block-size. Thanks to the flexibility of the DSP, smart decision algorithms can be implemented to achieve significant hardware reduction with low compression performance lost. A large range of applications can be targeted thanks to software and hardware trade-offs. The system achieves motion estimation of 720p videos at up to 60 frames per second.

This paper is laid out as follows: in section II the sub-pixel motion estimation is introduced and several approaches are discussed for embedded implementations. A DSP solution is given in section III and limitations are highlighted. Section IV proposes an efficient VLSI architecture for FME. The hybrid multi-component solution for H.264 is presented in section V and finally section VI presents our conclusion.

II. Sub-pixel accuracy motion estimation

Among the features introduced by H.264 standard to enhance coding efficiency [1], quarter-sample-accurate and vari-

F URBAN and R POULLAOUEC are with the Content Delivery and Compression lab of THOMSON RD France, e-mail: (ronan.poullaouec, fabrice.urban@thomson.net).

JF NEZAN and O DEFORGES are with the image Group lab of the Institude of Electronics and Telecomunications of Rennes, UMR CNRS 6164, France, e-mail: (jnezan, odeforge@insa-rennes.fr).

able block-size motion compensation play an important role. FME, however, involves a high computational complexity that is magnified by variable block-size. The enhancement of the reference picture resolution involves the use of interpolation filters. Computational and memory constraints are greatly increased.

A. H.264 standard interpolation filters

In the MPEG-4/AVC H.264 standard, the quarter-pixel accuracy luminance picture is interpolated with two successive filtering operations. Half-pixel samples are interpolated first using a 6-tap separable FIR filter with coefficients (1,-5, 20, 20, -5, 1). They are computed from 6 adjacent pixels horizontally and/or vertically. For example, b, h, and j in fig. 1 can be computed as follows:

$b = \frac{E-5F+20G+20H-5I+J}{32}$

$h = \frac{A-5C+20G+20M-5Q+S}{32}$

$j = \frac{aa-5bb+20b+20s-5gg+hh}{32}$

or $j = \frac{cc-5dd+20h+20m-5ee+ff}{32}$

Half-pixel samples not adjacent to 2 pixel locations are interpolated from previously computed samples using unrounded values (the value is the same whether j is interpolated vertically or horizontally). The resulting values are then rounded and clipped to a correct luminance value (i.e. between 0 and 255 for 8 bit depth).

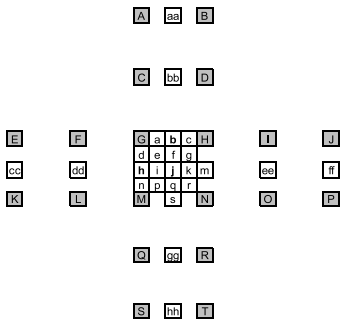


Figure 1. H.264 fractional pixel luminance interpolation filter

Once half-pixel samples are available, quarter-pixel samples are computed by averaging two adjacent samples as illustrated in fig. 2. For example, a and e are computed as follows:

$a = \frac{G+b+1}{2}$ and $e = \frac{h+b+1}{2}$

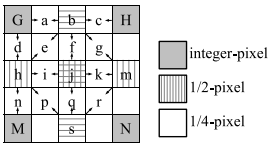


Figure 2. $\frac{1}{4}$-pixel interpolation filter

B. Implementation strategies

Improving motion vector accuracy for a block-based motion estimator obviously increases complexity. Firstly the density of candidate motion vectors is increased. To limit the number of search points a two-step approach is generally preferred. The motion is estimated at integer pixel accuracy and then refined to quarter-pixel with a limited search range (usually]-1;1[pixel) around the integer-accuracy best match. Secondly the reference image must be enlarged, involving the use of interpolation filters and introducing higher memory constraints. Two implementation strategies can be used to find a compromise between calculations and memory bandwidth: a basic data interpolation, or an on-the-fly data interpolation.

1) Basic data interpolation: The use of a basic sub-pixel motion vector refinement leads to the computation of an interpolated reference image before motion estimation. For quarter-pixel FME, the memory size and bandwidth are increased by a 16:1 ratio. Since memory constraints are already restrictive for IME [5], we can conclude that a basic data interpolation is not convenient for real-time FME implementation. Instead, on-the-fly data interpolation is preferred.

2) On-the-fly data interpolation: This is the commonly used solution. Here data interpolation is realized only when sub-pixel samples are needed in the algorithm i.e. the filters are applied on a small search window around the best integer-pixel accuracy matching block. Memory constraints are thus reduced. Hardware and software implementations can take advantage of high bandwidth local memories. External memory access concerns only integer-pixel accuracy data.

The two-step motion estimation with firstly IME and secondly FME combined with on-the-fly data interpolation appears to be the best choice for both performance and implementation constraints.

III. Motion estimation on DSP

A Digital Signal Processor (DSP) is a software processor dedicated to signal processing. Its hardware architecture is simplified to reduce its size and power consumption.

The programming is done in high level "C" language which offers high flexibility and reduced implementation time. Because of their low cost and high performance DSPs are popular for the prototyping of multi-processor signal processing applications. Depending on the device, numerous peripherals can be embedded such as a DMA controller, an Ethernet controller, a viterbi decoder, a cache controller, etc.

To get the best performance out of these processors, platform-independent optimizations such as loop unrolling and loop interchanging are usually done. Memory access can also be very time-consuming. For video processing and especially high-definition, internal memories are too small, which involves inherent external data accesses. Without an enhancement mechanism this causes performance to drop by orders of magnitude. Some DSPs integrate a cache controller that provides an automatic way to significantly reduce performance loss but it involves the user to ensure memory consistency in a multi-component context. In previous work, we developed an automatic cache management tool that uses the integrated cache controler of the device [11]. Using a prototyping methodology, inter-processor communications are automatically generated, external memory accesses are enhanced and data consistency is ensured by the tool. High definition image and video processing applications implementation on

a DSP is made fast and reliable with very limited memory constraints. The user can thus focus on application specific optimizations.

A. Integer-pixel accuracy motion estimation

Many IME have been developed to find a compromise between computation complexity and motion vector accuracy [12], [13], [14], [15], [16], [17], [18]. The well-known EPZS (Enhanced Predictive Zonal Search) [17] and hierarchical (HME) [18] algorithms have been implemented and evaluated in [19]. EPZS offers good performance in terms of both execution time and accuracy. HME is more robust but is more computationally intensive. The EPZS algorithm has been retained for the rest of this paper since it is faster and more popular for software implementation. The early stop threshold has been removed to simplify the algorithm and give a constant execution time i.e. not dependent on the sequence. Consequently, timing results are constant (close to *worst case)* regardless of the sequence, which is good design practice for real-time considerations.

B. Fractional-pixel accuracy pixel refinement

As soon as a motion vector is estimated at pixel-accuracy, it is refined to fractional-pixel accuracy. A reduced search is then performed around the best pixel-accuracy location using a two-step refinement approach. High performances are achieved thanks to two main characteristics: firstly the sub-pixel data is interpolated on-the-fly to reduce the necessary memory bandwidth and take advantage of cache architecture (data locality property), and secondly the two-step algorithm reduces the number of search points in a logarithmic search way [12]: the motion vector is first refined to half-pixel accuracy (8 neighbors), then to quarter-pixel accuracy starting from the half-pixel location (8 neighbors).

The sub-pixel search has an amplitude of $\frac{3}{4}$-pixel in each direction. The two-step technique gives one sub-pixel position out of 49 (7×7) with only 16 ($8 + 8$) search points. Quarter-pixel samples are interpolated after the half-pixel best position is known hence only necessary values are computed. This reduces execution time still further.

To perform the refinement process, four operations are executed sequentially.

- First the half-pixel filter is applied to the relevant image area. Fig. 3 shows the pixel window required for a 4×4 block and computed sub-pixel samples. Half-pixel samples of the search window are computed as well as those necessary for subsequent quarter-pixel interpolation.
- Secondly the best half-pixel location is chosen by evaluating distortion criteria for the eight neighbors. The Sum of Absolute Differences (SAD) is computed between each pixel of the source block and every other sample of the reference window with appropriate offset (the samples required to compute the same SAD are one pixel (or two half-pixels) away from each other vertically and horizontally). To take advantage of the 32-bit calculation units and vectorize the 8-bit operations, a correct data packing is performed within the SAD computation loop.

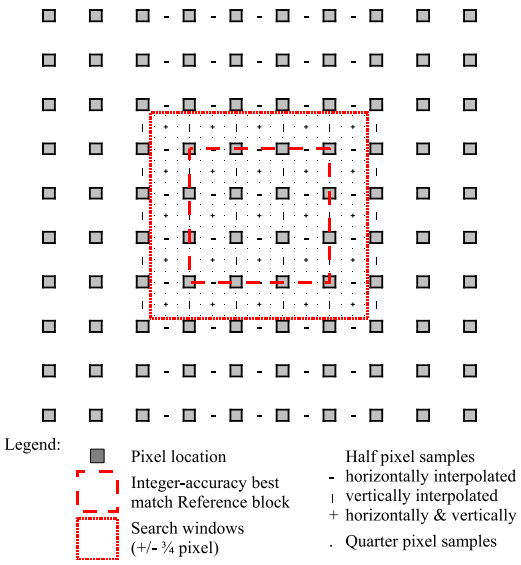


Figure 3. Sub-pixel search window and necessary data to compute quarter-pixel samples for a 4x4 block

- Thirdly quarter-pixel samples required to compute the SAD for the eight positions around the best half-pixel location are computed. There are no overlapping values between the eight search position reference samples. Consequently, quarter sample data are stored in 8 sets so that the values needed to compute one SAD are adjacent to each other.
- Finally the best quarter-pixel location is chosen from the nine positions (8 + center). The SAD calculation takes advantage of the efficient byte packing that has been undertaken in the interpolation filter. Multiple data can then be accessed simultaneously depending on the bus width, and vectorized instructions are directly used.

C. Implementation results

The quarter-pixel accuracy motion estimation comprising IME and two-step sub-pixel refinement has been implemented on a Texas Instruments C6416 DSP at 1GHz. It is one of the fastest DSP on the market [20]. Sub-pixel interpolation of the motion estimation is not standardized (unlike the case for motion compensation). Thus both H.264 and a simple bilinear filter have been implemented in order to evaluate the complexity trade-off. Table I gives execution times for 8x8 blocks. IME is performed in 900 ns. FME requires a 1200 and 630 ns overhead using an H.264 filter or bilinear filter respectively.

The sub-pixel refinement operation is the most time-consuming operation in motion estimation on a DSP. Using a bilinear filter reduces the complexity and almost halves its execution time. Section V-C discusses the impact of the choice of interpolation filter on motion estimation quality through rate-distortion encoding performance. One DSP has not enough computation power to handle real-time quarter-pixel motion estimation of high-definition video at 60 frames

filter type	H.264	Bilinear
IME	900 ns (77 fps for 720p)	
half-pixel filter	720 ns	150 ns
half-pixel refinement	210 ns	190 ns
quarter-pixel filter	140 ns	150 ns
quarter-pixel refinement	130 ns	140 ns
Total FME	1200 ns	630 ns
Total IME+FME	2100 ns (33 fps for 720p)	1530 ns (45 fps for 720p)

Table I
DSP IMPLEMENTATION TIMINGS FOR A 8X8 BLOCK

per second. The next section presents a sub-pixel refinement coprocessor to lower the computational burden on the DSP.

IV. PROPOSED VLSI ARCHITECTURE FOR FME

Despite the small search window in the sub-pixel refinement step, it is the most time-consuming part of motion estimation. The sub-pixel refinement operation is very computationally intensive and the search window is small. It could therefore benefit from the high parallelism of VLSI implementation. To meet the DSP external bus constraints, a low memory bandwidth requirement is needed. This section describes the dedicated VLSI architecture for fractional-pixel accuracy motion vector refinement. It achieves high throughput with low memory bandwidth and low resource utilization.

A. Overall VLSI architecture design

To benefit from the high degree of parallelism of VLSI implementation, the algorithm must be regular. A two-step approach, such that used in [6] requires either to save half-pixel samples for subsequent quarter-pixel interpolation, or to recompute them. To avoid implementation of memory, recomputation is needed [21]. As the filter is used twice for the same data, its efficiency is reduced and its consumption is raised. Moreover, in order to meet real-time constraints, the filter must be oversized along with memory bandwidth. Therefore a quarter-pixel accuracy full search approach with Lagrangian optimization is adopted with a search range of $\frac{3}{4}$ pixels in each direction. 48 candidates ($(2 \times 3 + 1)^2$-1) are evaluated around the IME best match. Input data required in order to refine a motion vector include the pixel-accuracy reference window, the current block and optionally the differentially coded integer-accuracy motion vector in order to compute a Lagrangian cost for each candidate.

The sub-pixel refinement coprocessor includes (fig 4) a quarter pixel interpolation module to generate sub-pixel samples. A Processor Element (PE) computes a distortion measure for each candidate displacement. The distortion measure used is the Sum of Absolute Differences (SAD) because it needs far less hardware resource than an SATD. The best candidate (i.e. with the smallest SAD) is finally selected in the decision tree. Block-size parameters can be changed without any hardware modification in order to support multiple block-size motion estimation.

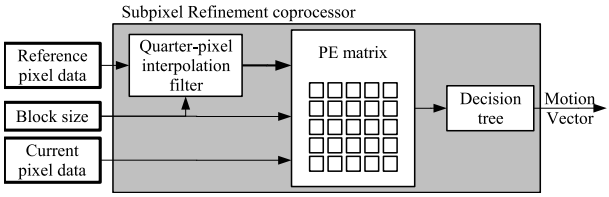


Figure 4. Overall VLSI architecture

Clock cycle & input pixel	horizontal 1/2 pel filter	Vertical 1/2 pel filter	1/4 pel filter
1(-3;-3_-2)	-	-	-
2(-3;-1_0)	-	-	-
3(-3;1_2)	-3;-1_0	-	-
4(-3;3_4)	-3;0.5_2	-	-
5(-3;5_6)	-3;2.5_4	-	-
⋮	⋮	-	-
21(1;-3_-2)	-	-	-
22(1;-1_0)	-	-1;-1	-
23(1;1_2)	1;-0.5_0	-1;-0.5_-1;0	-
24(1;3_4)	1;0.5_2	-1;0.5_-1;2	-
25(1;5_6)	1;2.5_4	-1;2.5_-1;4	-
26(2;-3_-2)	-	-	-
27(2;-1_0)	-	-0.5;-1_0;-1	-
28(2;1_2)	2;-0.5_0	-0.5;-0.5_0;0	-0.75;-0.75_0;0
29(2;3_4)	2;0.5_2	-0.5;0.5_0;2	-0.75;0.25_0;2
30(2;5_6)	2;2.5_4	-0.5;2.5_0;4	-0.75;2.25_0;4.75
31(3;-3_-2)	-	-	-
32(3;-1_0)	-	0.5;-1_1;-1	-
33(3;1_2)	3;-0.5_0	0.5;-0.5_1;0	0.25;-0.75_1;0
34(3;3_4)	3;0.5_2	0.5;0.5_1;2	0.25;0.25_1;2
35(3;5_6)	3;2.5_4	0.5;2.5_1;4	0.25;2.25_1;3.75
⋮	⋮	⋮	⋮
46(6;-3_-2)	-	-	-
47(6;-1_0)	-	3.5;-1_4;-1	-
48(6;1_2)	6;-0.5_0	3.5;-0.5_4;0	3.25;-0.75_3.75;0
49(6;3_4)	6;0.5_2	3.5;0.5_4;2	3.25;0.25_3.75;2
50(6;5_6)	6;2.5_4	3.5;2.5_4;4	3.25;2.25_3.75;3.75

Table II
H.264 INTERPOLATION FILTER DATAFLOW ($p = 2$)

The following sections details the architecture of each unit of the system.

B. Interpolation filter

Sub-pixel data is interpolated on-the-fly to eliminate the need for data memorization and access. Reference window pixels are input serially in raster-scan order beginning from the top-left corner of the reference window. Once the filter is initialized (depending on the filter size), 16 quarter-pixel samples are generated per input pixel, as shown in Fig. 5-left. To improve parallelism p input pixels are processed simultaneously per clock cycle, resulting in an output of $16 \times p$ quarter pixels per clock cycle. A state machine controls filter height and width, which ensures a full utilisation of interpolation resources. This way, no overlapping data is computed such as in [6], and the device is not oversized for small blocks, such as in [21].

Table II shows the dataflow of interpolated samples for the H.264 filter. The reference point (0;0 position) of given ranges is the top left pixel of current block. At each cyle,

two pixels of the search window are input. Once six pixels are input, the horizontal half-pixel filter can compute values. Half-pixel samples are output to the quarter pixel filter once the fifth line is available. Quarter pixel samples are computed after the input-pixel sixth line is available. The output range at each cycle corresponds to two of the squared positions of figure 5-left. Note that to make the table clear, pipeline latency is not taken into account, that is, two and three cycles for horizontally and vertically half-pixel filters respectively, and one cycle for quarter-pixel filter.

For blocks of size 4x4, only 16 out of the 25 sets is required for each SAD. Therefore the PE are enabled when appropriate data is available. It is done in four quadrants illustrated in figure 5-right.

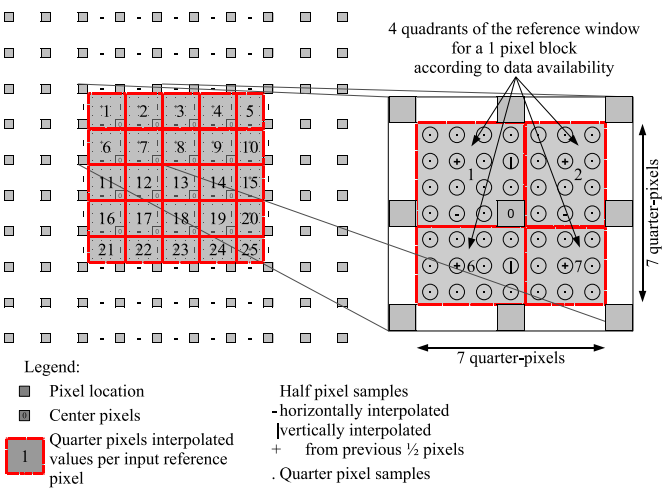


Figure 5. Scheduling of H.264 filtered sub-pixels availability for a 4x4 block (left) and distortion dependencies (right)

The design of the remainder of the architecture is dependent on the throughput of the filter only. Any interpolation filter can therefore be used: from H.264 filters for the best accuracy to bilinear filters to save on resources and time. The H.264 filter has been optimized for low resources and pipelined to support high frequency.

C. Distortion computation matrix

From the existing dedicated architectures for IME described in previous work [22], [5], [23], designs using inter-level parallelism [5] (defined as type II architecture in [22]) are best-suited to line-scan input mode and best minimize hardware resources.

In existing full-search sub-pixel refinement architectures [8], [7] the interpolation filter is not included in the study. We propose here a distortion computation matrix architecture that is adaptive to the block size and matches the interpolation filter design in order to reduce hardware requirements. In addition, the subsequent decision tree is also reduced.

1) Integer-pixel accuracy full search model: In [22] the full search IME algorithm with a search range of +/- r for a $n \times m$ block is expressed as 4 nested loops (Alg. 1).

Inter-level parallelism is obtained by unrolling Δi and Δj loops in hardware. The distortion measure is computed

Algorithm 1 IME nested loops

$for\, k = 1..m$ (block height)
 $for\, l = 1..n$ (block width)
  $for\, \Delta i = -r..r$ (vertical search range)
   $for\, \Delta j = -r..r$ (horizontal search range)
    $SAD(\Delta j, \Delta i) += |x_1(k,l) - x_2(k+\Delta i, l+\Delta j)|$
   $end\, \Delta j$
  $end\, \Delta i$
 $end\, l$
$end\, k$

x_1 denotes current picture and x_2 reference picture

simultaneously for every search point in $(2 \times r + 1)^2$ PEs. This model results from two hypotheses: firstly the current pixel $x_1(k,l)$ is broadcast to all PEs and secondly all reference pixels from $x_2(k-r, l-r)$ to $x_2(k+r, l+r)$ are available and propagated to the PEs through $(2r+1)^2$ registers.

In order to remove the registers needed to propagate reference data the architecture is inverted. Reference data is broadcast to the PEs and the current block is propagated. The associated algorithm is expressed (Alg. 2) with a variable interchange ($u = k + \Delta i$ and $v = l + \Delta j$). The latency is the same and idle initialization cycles are still needed [23]. The modification has little impact on IME, but presents several advantages when transposed to FME.

Algorithm 2 Inverted IME nested loops

$for\, u = 1 - s..m + s$ (block height)
 $for\, v = 1 - s..n + s$ (block width)
  $for\, \Delta i = -s..s$ (vertical search range)
   $for\, \Delta j = -s..s$ (horizontal search range)
    $SAD(\Delta j, \Delta i) += |x_1(u - \Delta i, v - \Delta j)$
    $-x_2(u,v)|$
   $end\, \Delta j$
  $end\, \Delta i$
 $end\, v$
$end\, u$

with $1 \leq u - \Delta i \leq m$ and $1 \leq v - \Delta j \leq n$

2) Inverted sub-pixel accuracy full search model: For FME, images x_1 and x_2 do not have the same scale. Data density is higher for the search window than for the current block. Consequently if the basic model (Alg. 1) is kept, the number of registers needed to propagate reference data ($(2ar+1)^2$, a being the accuracy factor i.e. $a = 2$ for half pixel, $a = 4$ for quarter-pixel) increases exponentially with accuracy [8].

To reduce the number of propagation registers Alg. 2 is considered and transposed to FME. Hence current block data is propagated and reference window data is spread to the appropriate PEs. Two loops (Δg and Δh) are added to handle fractional-pixel accuracy. Alg. 3 models the behavior of the PE matrix. Δi and Δj are integer-pixel displacements and Δg and Δh are fractional-pixel accuracy displacements. The number of propagation registers is then independent of the sub-pixel accuracy.

Algorithm 3 new FME nested loops

$for\, u = 1 - s..m + s$ (block height)
 $for\, v = 1 - s..n + s$ (block width)
  $for\, \Delta i = -s..s$ (integer vertical search range)
   $for\, \Delta j = -s..s$ (integer horizontal search range)
    $for\, \Delta g = -(a-1)..0$ (fractional v. search range)
     $for\, \Delta h = -(a-1)..0$ (fractional h. search range)
      $SAD(\Delta j, \Delta i) += |x_1(u - \Delta i, v - \Delta j)$
        $-x_2(au + \Delta g, av + \Delta h)|$
     $end\, \Delta h$
    $end\, \Delta g$
   $end\, \Delta j$
  $end\, \Delta i$
 $end\, v$
$end\, u$

with $-r \leq a\Delta i + \Delta g \leq r$ and $-r \leq a\Delta j + \Delta h \leq r$
and $1 \leq u - \Delta i \leq m$ and $1 \leq v - \Delta j \leq n$

Loops Δi, Δj, Δg and Δh are unrolled in hardware (e.g. for quarter-pixel accuracy this results in a 7×7 PE matrix). As a result, reference pixels from $x_2(ak - r, al - r)$ to $x_2(ak, al)$ are spread to appropriate PEs and current pixels $x_1(k, l)$ to $x_1(k - s, l - s)$ are propagated.

The inequalities $1 \leq u - \Delta i \leq m$ and $1 \leq v - \Delta j \leq n$ are ensured in hardware by propagating an "*enable*" signal to the PEs along with current block data. Consequently, not all the results are available simultaneously. The first a^2 cost results are available after $m(n + 1)$ cycles and subsequent results after appropriate delays. The new resulting systolic architecture (Fig. 6) presents several advantages:

- Propagation registers involves now the current block instead of the reference window and are thus roughly divided by a^2.
- The subsequent comparison unit is reduced (shorter pipeline and less logic) because the cost results of all the search points are not available simultaneously.
- The system works as fast as the interpolation filter can provide data because search area samples computed on the fly are used immediately.
- Interpolated data are never stored, saving bandwidth and memory/registers.

The proposed architecture reduces hardware requirements compared to previous solutions while giving optimal performances. The PE matrix and the comparison tree take account of data dependencies on the interpolation filter. As soon as the data are available, the absolute differences between the quarter-pixel samples and the current pixel values are computed, and then accumulated. Fig 6 shows the design of the SAD computation matrix. In order to prevent an overloading of the figure, only half-pixel is represented ($a = 2$ and $r = \frac{1}{2}$) and $p = 1$. The reference window is partitioned into four quadrants because of data scheduling of the interpolation filter. Each quadrant represents the unrolling of Δg and Δh loops of Alg.3. The four quadrants are the result of the unrolling of Δi and Δj loops. The top left quadrant is started first, the top right quadrant is delayed by one cycle and the delay for the bottom quadrants corresponds to the computation of one line. Dummy cycles necessary to fill the search area registers are removed from the initialization step. Instead, results are partitioned according to data availability and the decision tree resources are reduced. Indeed, comparison units are shared by the 4 quadrants; for quarter-pixel accuracy there are at most $a^2 = 16$ values (out of 48) to compare at a time.

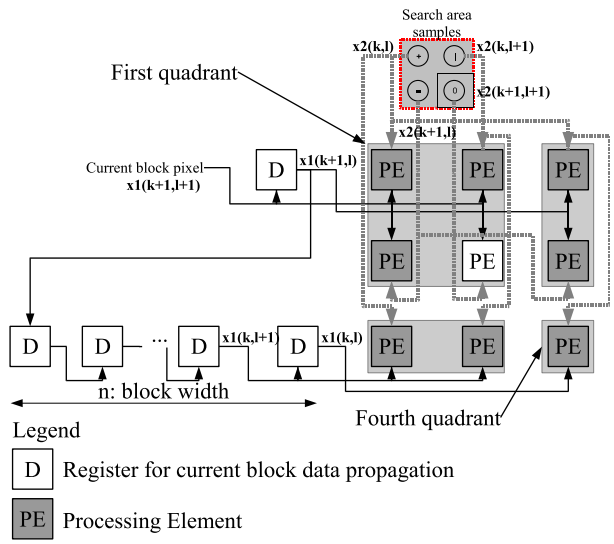


Figure 6. PE matrix design ($a = 2$ and $r = \frac{1}{2}$ and $p = 1$)

3) Increasing parallelism: This architecture makes full use of inter-level parallelism, and in addition it supports intra-level parallelism. Performances can be increased by unrolling Δl loop of Alg. 1 (or Δv loop of Alg. 3) by a factor p. Both the filter and the interpolation matrix are affected. The detail of PE implementation in fig 7 corresponds to $p = 2$. At each cycle, two absolute differences are computed and accumulated. The performances are thus roughly doubled at the cost of an increase in hardware resources (the interpolation filters and PE matrix are enlarged).

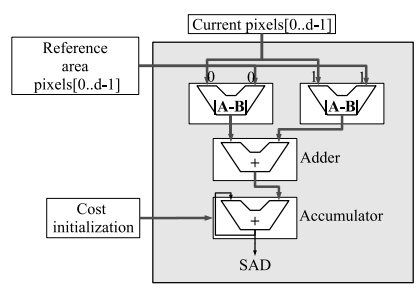


Figure 7. PE detail with $p = 2$

In order to consider rate-distortion optimization, the SAD accumulator is optionally initialized by a value corresponding to the Lagrangian cost of the vector. This value is computed with very low hardware resources during the initialization of the interpolation pipeline.

A high degree of parallelism is achieved with very low memory bandwidth. The architecture is scalable in that the

throughput can be enhanced by increasing the intra-level parallelism. The combination of two parallelism techniques (intra and inter-level) leads to high efficiency. Hardware resources are almost independent of the block size as only propagation registers are involved. Thus variable block-size is supported with no hardware modification, and the hardware is always fully utilized regardless the block-size.

D. Decision tree

To obtain the optimal motion vector, the costs computed by the PE matrix are compared to each other in a dedicated comparison tree (Fig. 8). Simultaneously available values (from the same quadrant) are compared in a binary tree structure. The lowest cost of each quadrant is then sequentially compared to the current optimum. Finally the sub-pixel motion vector is output when the 4th quadrant's costs have been processed.

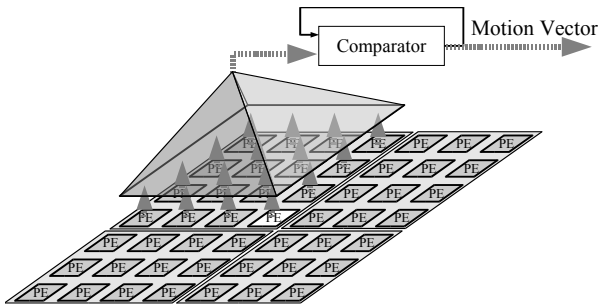


Figure 8. Decision tree

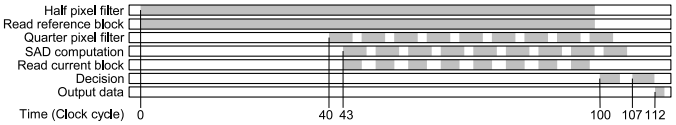
The binary tree base size is limited to a^2 thanks to the early calculation of distortion in the PEs. Consequently, its resources are shared between the 4 quadrants. In addition, the processing pipeline is shorter because there are fewer values to compare.

E. Implementation results

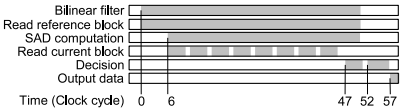
The scheduling of each unit of the architecture for an 8x8 block and an intra-level parallelism of $p = 2$ is shown in figure. 9-a for the H.264 filter and figure. 9-b for the bilinear filter. The impact of the interpolation filter on scheduling can be clearly seen. Using the 6-tap H.264 filter requires far more initialization cycles than a simple bilinear filter. This results in more idle cycles for the PE matrix. Besides, a bilinear filter can output all subpixels in one cycle.

The number of clock cycles necessary to refine an 8x8 motion vector to quarter-pixel accuracy is given in table III for several configurations. The implementation of the refinement with an intra-level parallelism of 2 pixels per cycle results in latency of 112 cycles with the H.264 interpolation filter. Comparable timing performances are achieved with a bilinear filter and $p = 1$ which results in logic savings. Section V-C discusses the impact of filter implementation on encoding performances.

The H264 filter has been retained for implementation since it is likely to achieve better motion vector field quality. For real-time performances, 720p video has to be processed at 60 Hz. Thus the FPGA must process an 8x8 block in less than



(a) H.264 filter



(b) Bilinear filter

Figure 9. Units scheduling for 8x8 block-size and $p = 2$

Input width	H.264 filter	Bilinear filter
$p = 1$ pixel	209	106
$p = 2$ pixel	112	57
$p = 4$ pixel	70	37

Table III
FPGA IMPLEMENTATION TIMINGS FOR 8X8 BLOCKS (CYCLES)

$(\frac{720\times1280\times60}{8\times8})^{-1} = 1157\,ns$. At a frequency of 133 MHz, it results in 154 cycles. The parameter $p = 2$ will be retained to perform real-time motion vector refinement of 720p 60Hz video with reasonable frequency and low logic resources.

Table IV gives a comparison of synthesis results for quarter-pixel refinement implementations. The presented coprocessor with an H.264 filter and $p = 2$ pixels per cycle had been prototyped on a Xilinx Virtex II Pro FPGA. The maximum frequency is 150 MHz. A higher value can be expected on a the latest FPGA, and even more on ASICs. The design is done in VHDL and briefly optimized, better results could be achieved with further optimizations.

The input bandwidth of two pixels per cycle per image is interesting to connect the coprocessor through a small-bandwidth bus. For example a DSP with a 32-bit bus can provide data to keep the coprocessor busy.

The system using the H.264 filter and clocked at 133MHz

	Chen's [6]	Yang's [21]	This work
cell area (Kgates)			
H.264 filter	23.9	119	14.5
PE unit	34.8	41.1	54
Decision	2.2	8.5	10.4
Total	79.3	188	94
memory bandwidth (pixels/cycle)			
Cur block	4	16	2
Ref block	10	22	2
Cycles per block			
4x4	10×2	5×2	64
8x8	28×2	14×2	112
16x16	88×2	22×2	498

Table IV
COMPARISON WITH PREVIOUS WORK

refines an 8x8 motion vector to quarter-pixel accuracy in 840 ns. The peak throughput is thus 82 frames per second for 720p video. The architecture achieves a high throughput with reduced logic. The design of each module is in adequacy with the throughput of previous one. Therefore the datapath is optimized.

External memory bandwidth is kept low compared to previous work (4 pixels/cycle, compared to 14 and 38). Moreover, data is accessed only once instead of twice because of the full-search approach . Single block-size speed performances are reduced compared to previous work, but the whole motion estimator is enhanced and the exhaustive search of all the seven block sizes is made useless.

The block-size parameters can be changed without the need for hardware modification to support variable block-size. In Chen's design, the system matches 4x4 block-size, resulting in redundant computations for bigger block-size. Yang's design eliminates redundancy with a 16 pixel-wide unit, resulting in hardware sub-utilization for smaller block-size. This work takes variable block-size into consideration to design adaptive units giving better performance.

Although the full search approach is efficient to reduce data transfers and computation time, it requires the computation of more distortions compared to a two-step approach. The gate-count and the power consumption is thus raised in the PE unit. To limit the gate-count increase, the search range can be reduced to]-0.5;0.5[, giving only 25 search points but impacting encoding performances. This will not be discussed further in this arcticle.

This architecture for fractional-pixel accuracy motion vector refinement combines efficiency and resource saving. It is scalable in that its throughput can be enhanced by increasing intra-level parallelism, involving a higher memory bandwidth and a non-negligible increase in logic resources.

V. Multi-component ME

Integer motion estimation algorithms perform well on DSPs. Fast algorithms take advantage of branching and random memory access abilities. Fractional-pixel accuracy, however, requires huge computation power and memory bandwidth due to interpolation filters and distortion evaluation. These operations take advantage of the high parallelism of VLSI implementation. The results presented in sections III-C and IV-E show that a small FPGA outperforms the latest DSPs for the sub-pixel accuracy refinement task.

A heterogeneous multi-component motion estimator has been prototyped on a platform from *Sundance (SMT395)* comprising A TI C6416 DSP at 1GHz and a Virtex-2 Pro FPGA. The inter-connection bus between the DSP and the FPGA is 32-bit wide and is clocked at 133 MHz. The DSP handles IME and allows algorithmic modifications to reduce sub-pixel refinement complexity, whereas the Virtex 2 Pro FPGA runs as a quarter-pixel motion refinement coprocessor. No memory is connected to the FPGA thus all the data has to be sent by the DSP. Therefore the coprocessor must run with low memory bandwidth. Moreover, as the FPGA already manages communications between the DSP external world. The interface with the DSP's external memory is reused to plug the FME. The coprocessor design must have reduced logic utilization to fit into the remaining space.

This section presents performance results of the multi-component design as well as a comparison with a full-DSP implementation. The motion estimator has been implemented in Thomson's H.264 video encoder to evaluate design trade-off impacts on encoding performances.

A. FPGA as a sub-pixel refinement coprocessor

The IME is based on a predictive algorithm (cf Section III-A). Previously estimated motion vectors are necessary to predict the current motion as well as to compute the Lagrangian cost. This causes data dependencies between IME and FME which result in inevitable sequential processing. To take advantage of the parallel multi-component architecture, the motion vector of the left block is input in the IME stage at integer accuracy instead of quarter-pixel accuracy. As a result the motion estimation architecture is a 2-stage pipeline. The first stage is the IME on DSP and the second stage is the FME on FPGA (Fig. 10).

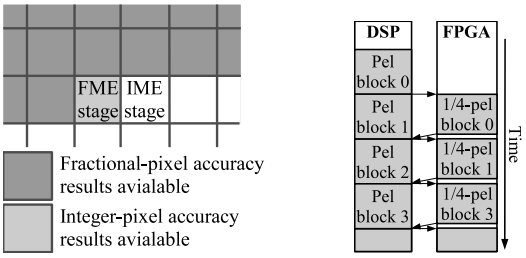


Figure 10. Block-level pipeline implementation

B. Timing results

Several configurations of the motion estimator have been benchmarked for 720p high-definition video sequences (1280x720). Execution times are given in the table V for IME only on DSP, FME only on DSP and FPGA, and the complete heterogeneous quarter-pixel accuracy motion estimator (DSP+FPGA).

The DSP runs the EPZS Integer motion estimation algorithm at 900 ns for an 8x8 block and 4000 ns for a 16x16 block. Sub-pixel refinement can be performed on the DSP with a 1200 and 4400 ns overhead for 8x8 and 16x16 block-size respectively whereas sub-pixel refinement on the FPGA takes only 842 and 1925 ns and is realized in parallel thus with no overhead.

Practical results present an execution time of 1250 and 4600 ns for 8x8 and 16x16. The small overhead is due to the prototyping platform constraints. No external memory is connected to the FPGA thus, for each block, both current block and reference window must be transferred by the DSP to input buffers on the FPGA. This increases the refinement operation time which has to take account of data transfers.

The computation time of a 1/4-pixel high-definition motion field with only one DSP is around 30 ms for both 8x8 and

Block size	8x8	16x16
DSP pel	900 ns (720p frame: 13 ms)	4000 ns (14.4 ms)
DSP 1/4 pel refinement only w H.264 filter	1200 ns (17.3 ms)	4400 ns (15.8 ms)
FPGA 1/4 pel refinement only w H.264 filter	842 ns (12 ms)	1925 ns (7 ms)
Total DSP + FPGA Pel + 1/4 pel	1250 ns **(720p frame: 18 ms)**	4600 ns **(16.5 ms)**

Table V
EXECUTION TIMES FOR 8X8 AND 16X16 BLOCKS

FME implementation	Bit-rate loss				
	city	horses	Hockey	Crowd	Spin cal.
1. Full $\frac{1}{4}$-pixel Sequential	Full quality reference encoder				
2. Full $\frac{1}{4}$-pixel block-level pipeline	0.1%	0.2%	0.1%	0%	0.2%
3. 2 step $\frac{1}{4}$-pixel H.264 filter	0.5%	0.4%	0.4%	0%	1%
4. 2 step $\frac{1}{4}$-pixel bilinear filter	2.4%	1.7%	2.6%	1%	4.1%
5. $\frac{1}{2}$ pixel accuracy	30%	11%	15%	21%	28%
6. pixel accuracy	61%	24%	32%	47%	61%

Table VI
SUB-PIXEL REFINEMENT IMPLEMENTATION TRADE-OFFS AND RELATIVE BIT-RATE LOSS

16x16 block-sizes. The addition of an FPGA at 133 MHz and an appropriate operation schedule significantly decrease running times, achieving 55 and 60 frames per second for 8x8 and 16x16 block-size respectively. These figures take data transfers between the DSP and the FPGA into account. The coprocessor design lead to low data bandwidth which is appropriate for the prototyping platform. Its performances can be enhanced with an appropriate hardware design e.g a frame memory connected to the coprocessor.

The coprocessor achieves good acceleration results with low resources and memory bandwidth. To further improve performances, intra-level parallelism can be increased. The throughput roughly doubles each time input bandwidth p doubles, but it involves a gate count increase.

C. Design trade-offs and encoding performances

In order to compare the impact of implementation trade-offs on motion field quality, the motion estimator has been implemented in an H.264 video encoder developed internally at Thomson R&D. It is a software development kit inspired by JM reference software [24]. Many 1280x720 high-definition video sequences have been encoded with variable block-size motion estimation from 8x8 to 16x16, one reference frame and constant quantification at different quantification steps and for different configurations of motion estimation. Then, each configuration is compared to the full-quality encoder in terms of bit-rate increase at constant PSNR.

The quarter-pixel accuracy motion estimation reduces the bit-rate by up to 60% over pixel-accuracy and 30% over half-pixel accuracy motion estimation. Table VI summarizes relative bit-rate loss for different sub-pixel motion refinements. Reference (1.) is a $\frac{3}{4}$-pixel range full search with sequential scheduling of IME and FME, (2.) uses block-level pipelining, (3.) and (4.) are the 2-step refinement with H.264 filter and bilinear filter respectively, (5.) and (6.) are half-pixel and integer-pixel accuracy motion estimators.

(1.) gives the best results but is not convenient in either software or hardware implementation. (2.) and (3.) have negligible loss. (2.) is more suited to hardware implementation whereas (3.) is adapted to software. The bilinear filter reduces FME cost and has little impact on encoding performances for most video sequences but results in a bit-rate increase of up to 4% for the "Spin calendar" sequence. It is a possible candidate for a low-cost motion estimator.

The quarter-pixel motion estimation property of H.264 significantly increases encoding performances. It is an important tool but is computationally intensive. Besides, its computation power constraints are magnified by variable block-size motion estimation.

D. Variable block-size considerations

Many studies have been conducted on variable block-size motion estimation [5], [25] using the full search algorithm and SAD reuse. However, for fractional-pixel accuracy motion estimation, this technique leads to a small search range for a realistic implementation [7]. SAD reuse is based on computing SADs for the smallest block-size and combining them for bigger block-sizes. Fast algorithms are not compatible with SAD reuse because the motion vectors of different sub-blocks are not usually the same.

Consequently, for a realistic implementation of variable block-size and fractional pixel motion estimation (VBSFME), a two-step approach is preferred: IME has to be conducted first followed by FME for each block-size. A smart decision algorithm can reduce the complexity by selecting the block-size mode before FME.

Two solutions have been implemented to study the complexity versus quality trade off. For high-definition video sequences, block-sizes smaller than 8x8 lead to high complexity and provide little compression gain [3], thus only 16x16, 16x8, 8x16 and 8x8 are considered for implementation. Complexity is reduced and compression loss is negligible for most video sequences.

The first solution (VBSFME1) computes IME and a quarter-pixel accuracy motion vector for each mode. Computation power for IME and FME are magnified. Because of this, either the sub-pixel refinement coprocessor has to be enlarged to handle each mode sequentially, as it is done in [6] and [21], or multiple refinement coprocessors are required to handle each mode in parallel.

The second solution (VBSFME2) is a low-cost one. IME is accelerated thanks to software trade-offs. The algorithm starts with an 8x8 block-size and bigger blocks are processed with a bottom-up merge procedure [26]. The final mode is selected from IME results before FME which is performed for only one mode, requiring a reduced quarter-pixel refinement coprocessor. It can achieve H.264 motion estimation of 720p video at up to 60 frames per seconds. The impact of the hardware trade-off on encoding performance is shown in table VII. For comparison's sake, simulations have also been performed with only 16x16 blocks and 8x8 bocks $\frac{3}{4}$-pixel motion estimation (resp. (8.) and (9.)).

VBSFME implementation	Bit-rate loss				
	city	horses	Hockey	Crowd	Spin cal.
1. Full $\frac{1}{4}$-pixel Sequential VBSFME1	Full quality reference encoder				
7. VBSFME2	1.7%	2.4%	2.2%	4.3%	4.6%
8. 16x16 $\frac{1}{4}$-pixel	4.1%	4.7%	4%	8.1%	6.6%
9. 8x8 $\frac{1}{4}$-pixel	18%	20%	20%	8.8%	21%

Table VII
VARIABLE BLOCK-SIZE IMPLEMENTATION TRADE-OFFS AND RELATIVE BIT-RATE LOSS

In addition to a high motion vector cost, motion compensation on only 8x8 blocks prevents the use of the skip mode, which operates only on 16x16 blocks. This leads to a decrease in encoding performance. 16x16 blocks (8.) provide better encoding performance than 8x8 blocks (9.) but the bit-rate loss is still high (8.1 %). Moreover, the impact of variable block-size on encoding performance is higher when B frames are used because the direct mode reduces motion cost. VBSFME2 offers a compromise between computation power and encoding performance but the best mode is difficult to estimate before FME and the bit-rate loss reaches 4.6% with a first mode selection algorithm. However, it is a good candidate for a low-cost solution and selection the algorithm can be enhanced.

For high-definition video, using fractional motion estimation and single block-size (e.g 16x16) results in significantly better encoding performance than variable block-size with integer-pixel accuracy. VBSFME2 is thus appropriate, because it is better than fixed block-size ME with the same hardware cost.

VI. CONCLUSION

Coding tools supported by the H.264 standard provide high compression performances but introduce challenging computation constraints that are magnified in a high-definition context. Fractional-pixel accuracy and variable block-size motion estimation is too computationally intensive for a real-time solution on a single DSP. The complexity of fractional-pixel accuracy motion estimation is firstly addressed and the limitations of DSP processing power for the sub-pixel motion vector refinement is highlighted. Then a scalable and flexible low-complexity VLSI architecture is designed for a sub-pixel refinement coprocessor.

Finally, we propose a flexible hybrid solution for real-time variable block-size and fractional-pixel motion estimation of high-definition video sequences. A DSP computes a fast integer motion estimation such as EPZS and an FPGA refines motion vectors to quarter-pixel accuracy. An efficient distribution and scheduling of the operations provides parallel computing. The programmability of the DSP combined with the flexibility of the coprocessor design allow various implementation trade-offs and adaptability to future standards e.g. SVC. The system achieves real-time variable block-size and quarter-pixel accuracy motion estimation of 720p video at up to 60 frames per second with reduced hardware cost.

Future work will focus on the mode selection algorithm from IME, for P pictures, and B pictures.

REFERENCES

[1] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, July 2003.

[2] J. Ostermann, J. Bormans, P. List, D. Marpe, M. N. andF. Pereira, T. Stockhammer, and T. Wedi, "Video coding with h.264/avc: tools, performance, and complexity," *Circuits and Systems Magazine*, vol. 4, pp. 7– 28, 2004.

[3] S. Saponara, K. Denolf, G. Lafruit, C. Blanch, and J. Bormans, "Performance and complexity co-evaluation of the advanced video coding standard for cost-effective multimedia communications," *EURASIP Journal on Applied Signal Processing*, vol. 2, pp. 220–235, 2004.

[4] Z. Chen, P. Zhou, and Y. He, "Fast integer pel and fractional pel motion estimation for jvt," *JVT-F017, 6th Meeting of Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG*, 2002.

[5] C.-Y. Chen, S.-Y. Chien, Y.-W. Huang, T.-C. Chen, T.-C. Wang, and L.-G. Chen, "Analysis and architecture design of variable block-size motion estimation for H.264/AVC," *IEEE Transactions on Circuits and Systems I*, vol. 53, no. 3, pp. 578 – 593, March 2006.

[6] T.-C. Chen, Y.-W. Huang, and L.-G. Chen, "Fully utilized and reusable architecture for fractional motion estimation of H.264/AVC," *Internatinal Conference on Acoustics, Speech an Signal Processing*, vol. 5, pp. 9–12, 2004.

[7] C. Rahman and W. Badawy, "A quarter pel full search block motion estimation architecture for H.264/AVC," *IEEE International Conference on Multimedia and Expo*, July 2005.

[8] T. Dias, N. Roma, and L. Sousa, "Fully parameterizable vlsi architecture for sub-pixel motion estimation with low memory bandwidth requirements," November 2005.

[9] K. Gaedke, M. Borsum, M. Georgi, A. Kluger, J.-P. Le Glanic, and P. Bernard, "Architecture and VLSI implementation of a programmable HD real-time motion estimator," *IEEE International Symposium on Circuits and Systems*, May 2007.

[10] I. Berkeley Design Technology, Ed., *FPGAs for DSP,Second Edition*, September 2006.

[11] F. Urban, M. Raulet, J. F. Nezan, and O. Déforges, "Automatic DSP cache memory management and fast prototyping for multiprocessor image applications," *14th European Signal Processing Conference*, Sept 2006.

[12] J. R. Jain and A. K. Jain, "Displacement measurement and its application in interframe coding," *IEEE Transactions on Communications*, vol. COM-29(12), pp. 1799–1808, 1981.

[13] P. Hosur and K. Ma, "Motion Vector Field Adaptive Fast Motion Estimation," *Second International Conference on Information, Communications and Signal Processing (ICICS '99)*, 1999.

[14] K. Virk, N. Khan, S. Masud, F. Nasim, and S. Idris, "Low Complexity Recursive Search Based Motion Estimation Algorithm for Video Coding Applications," in *Proceedings of 13th European Signal Processing Conference*, Antalya, Turkey, 2005.

[15] W. Li and E. Salari, "Successive elimination algorithm for motion estimation," *IEEE Transactions on Image Processing*, vol. 4, pp. 107–110, 1995.

[16] Y.-S. Chen, Y.-P. Hung, and C.-S. Fuh, "Fast Block Matching Algorithm Based on the Winner-Update Strategy," in *IEEE Transactions on Image Processing*, vol. 10, August 2001.

[17] A. M. Tourapis, "Enhanced Predictive Zonal Search for Single and Multiple Frame Motion Estimation," *Visual Communications and Image Processing*, pp. 1069–79, 2002.

[18] B. Chupeau, P. Robert, M. Pecot, and P. Guillotel, "Multiscale motion estimation," *Workshop on Advanced Matching in Vision and Artificial Intelligence*, 5th, 6th June 1990.

[19] F. Urban, R. Poullaouec, J. F. Nezan, and O. Déforges, "Real-time Multi-DSP Motion Estimator for MPEG-4 AVC/H.264 High Definition Video," *International Conference on Signals and Electronic Systems*, September 2006.

[20] K. Williston, "Microprocessors vs. DSPs," http://www.bdti.com/articles/info_articles.htm, March 2006.

[21] C. Yang, S. Goto, and T. Ikenaga, "High performance vlsi architecture of fractional motion estimation in h.264 for hdtv," in *IEEE International Symposium on Circuits and Systems*, May 2006, pp. 2605–2608.

[22] L. Vos and M. Stegherr, "Parameterizable VLSI architectures for the full-search block-matching algorithm," *IEEE Transactions on Circuits and Systems*, vol. 36 issue 10, pp. 1309–1316, 1989.

[23] H. Yeo and Y. H. Hu, "A novel modular systolic array architecture for full-search blockmatching motion estimation," *International Conference on Acoustics, Speech, and Signal Processing*, vol. 5, pp. 3303–3306, May 1995.

[24] G. Sullivan, A. M. Tourapis, and K. Sühring, *H.264/MPEG-4 AVC Reference Software Manual*, Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, october 2006.

[25] S. Y. Yap and J. V. McCanny, "A vlsi architecture for variable block size video motion estimation," *IEEE Transactions on Circuits and Systems: Expess Briefs*, vol. 51, no. 7, pp. 384–389, July 2004.

[26] Z. Zhou, M.-T. Sun, and Y.-F. Hsu, "Fast variable block-size motion estimation algorithms based on merge and split procedures for h.264/mpeg-4 avc," *International Symposium on Circuits and Systems*, vol. 3, pp. 725–8, May 2004.