J Real-Time Image Proc (2007) 2:191–205
DOI 10.1007/s11554-007-0049-6

SPECIAL ISSUE

Reconfigurable architectures and processors for real-time video motion estimation

Tiago Dias · Nuno Roma · Leonel Sousa · Miguel Ribeiro

Received: 14 May 2007 / Accepted: 12 October 2007 / Published online: 8 November 2007

Abstract With the recent proliferation of multimedia applications, several fast block matching motion estimation algorithms have been proposed in order to minimize the processing time in video coding. While some of these algorithms adopt pre-defined search patterns that directly reflect the most probable motion structures, other data-adaptive approaches dynamically configure the search pattern to avoid unnecessary computations and memory accesses. Either of these approaches leads to rather difficult hardware implementations, due to their configurability and adaptive nature. As a consequence, two different but quite configurable architectures are proposed in this paper. While the first architecture reflects an innovative mechanism to implement motion estimation processors that support fast but regular search algorithms, the second architecture makes use of an application specific instruction set processor (ASIP) platform, capable of implementing most data-adaptive algorithms that have been proposed in the last few years. Despite their different natures, these two architectures provide highly configurable hardware platforms for real-time motion estimation. By considering a wide set of fast and adaptive algorithms, the efficiency of these two architectures was compared and several motion estimators were synthesized in a Virtex-II Pro XC2VP30 FPGA from Xilinx, integrated within a ML310 development platform. Experimental results show that the proposed architectures can be easily reconfigured in run-time to implement a wide set of real-time motion estimation algorithms.

Keywords Motion estimation · Fast search algorithms · Video coding · Reconfigurable devices · Application specific instruction set processors

T. Dias
SiPS, INESC-ID/ ISEL, Rua Alves Redol 9,
1000-029 Lisbon, Portugal
e-mail: Tiago.Dias@inesc-id.pt

N. Roma · L. Sousa (✉)
SiPS, INESC-ID/ IST, Rua Alves Redol 9,
1000-029 Lisbon, Portugal
e-mail: leonel.sousa@inesc-id.pt

N. Roma
e-mail: Nuno.Roma@inesc-id.pt

M. Ribeiro
SiPS, INESC-ID, Rua Alves Redol 9,
1000-029 Lisbon, Portugal

1 Introduction

Motion estimation (ME) is one of the most important operations in video coding, to exploit temporal redundancies in sequences of images. However, it is also the most computationally costly part of a video encoder. Depending on the adopted search algorithm, up to 80% of the operations required to implement any of the recently proposed video coding standards (such as the H.26x and the MPEG-x families or the H.264/AVC), are devoted to ME [4, 10]. Despite that, most video coding applications of such standards apply the block-matching (BM) ME technique, usually over macroblocks (MBs) with 16 × 16 pixels or, more recently, by considering variable MB sizes and sub-block partitioning modes for more accurate motion estimation. Nevertheless, although the BM approach simplifies the ME operation, by considering the same translational movement for the whole block, real-time ME is usually only achievable with specialized VLSI processors. Such fact has imposed several constraints on the ME algorithms that have been adopted.

Meanwhile, the sum of absolute differences (SAD) has been the most used distortion measure, since it provides a good trade-off between the quality of the obtained motion vectors (MVs) and the involved computational cost. It is obtained as depicted in Eq. 1, where C and R denote the current and previously encoded frames, respectively.

$$\mathrm{SAD}_{(v_x,v_y)} = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} \left| C_{(x+m,y+n)} - R_{(x+v_x+m,y+v_y+n)} \right| \qquad (1)$$

Several different ME search algorithms have been proposed. The full-search BM method [16, 1] was the most adopted algorithm in the development of hardware motion estimators, due to its regularity and data independence. However, only application specific integrated circuit (ASIC) technology could provide the performance levels required for large search areas, due to its extremely high computational requirements. In the 1990s, several non-optimal but faster search BM algorithms were proposed, such as the three-step-search (3SS), the four-step-search (4SS) and the diamond search (DS). However, these algorithms have been mainly applied in pure software implementations, better suited to support the inherent data-dependency and irregular search patterns. Although some individual hardware architectures were actually proposed [6, 9], they usually resulted in complex and inefficient hardware designs, that do not offer any reconfiguration capability. Meanwhile, data-adaptive ME algorithms have also been proposed, as a result of the recent advent of the H.264/AVC encoding standard. Some examples of these algorithms are the motion vector field adaptive search technique (MVFAST), the enhanced predictive zonal search (EPZS) and the fast adaptive motion estimation (FAME). These algorithms avoid unnecessary computations and memory accesses by taking advantage of the MVs' temporal and spacial correlations, in order to adapt and optimize the search patterns. However, few hardware implementations have been presented up until now for these ME approaches [8], mainly because their operation has to adjust, itself, to the characteristics of both the video input signal and the encoding system.

Hence, the availability of such a large amount of search algorithms allows the selection of the best search strategy for a given video coding system, based on its computational complexity and efficiency (both in terms of the obtained image quality and bitrate). However, it is also possible to switch between different search strategies, in order to better adapt the coding algorithm to the instantaneous restrictions imposed by the video coding application, the terminal device or the communication channel. As an example, in mobile video coding applications a faster and less power eager algorithm, such as the 3SS, can be used whenever the system starts running out of battery energy, while a more efficient algorithm, such as the MVFAST or even the FSBM, might be selected when the available bandwidth of the communication channel is reduced, so that the video quality is maintained at an acceptable minimum level. Table 1 presents the PSNR and bitrate average values that were obtained by coding the *Table Tennis* benchmark video sequence using variable and fixed (QP = 6) quantization step sizes, in order to obtain output video streams with a constant bitrate or a constant quality level, respectively. From the presented values, in particular from those corresponding to a fixed bitrate configuration, it can be seen that the output video quality level may significantly depend on the adopted algorithm.

Table 1 Average PSNR and bitrate values obtained by coding the *Table Tennis* video sequence considering several different ME algorithms

Algorithm	Variable bitrate		Fixed bitrate	
	PSNR (dB)	Bitrate (kbps)	PSNR (dB)	Bitrate (kbps)
FSBM	35.60	15.57	42.51	31.58
3SS	34.09	56.27	29.95	31.58
DS	34.07	56.34	29.78	31.58
MVFAST	34.14	53.08	30.25	31.58

In the last few years, reconfigurable devices, such as field-programmable gate arrays (FPGAs), have proved to provide flexible and high-performance platforms to implement real-time processing systems. Recently, FPGA-based architectures were even proposed to encode video in real-time [12, 17]. However, the distinct nature of the several ME algorithms and the possible architectures that have been proposed still often pose a difficult trade-off between the provided efficiency level and the reconfigurable and programmability capabilities of each ME system, in order to easily adapt its performance to the target application or communication channel.

Hence, two new highly efficient and reconfigurable architectures for high-performance ME are presented and compared in this paper:

Architecture A: reconfigurable structure highly suited for the implementation of any regular search pattern, such as any of the previously referred fast sub-optimal search algorithms or even the optimal full-search;
Architecture B: programmable specialized and configurable architecture, capable of implementing any of the previously referred class of ME algorithms, but particularly well suited and optimized to the implementation of ME algorithms with irregular execution patterns, such as the recently proposed data-adaptive search algorithms, adopted by the H.264/AVC coding standard.

Besides the high performance and run-time reconfigurable characteristics of these architectures, they focus different optimization trade-offs: while architecture A mainly exploits the inherent parallelism by using a priori information about the regular search patterns; architecture B provides a significantly broader programmable capability, at the cost of a slightly slower processing rate. Consequently, architecture A is more suitable for implementations of video encoding systems with reduced complexity, and that therefore require the use of more efficient ME units to execute fast sub-optimal search strategies. On the other hand, architecture B is better suited to more advanced systems with increased programability and with more hardware resources. Furthermore, it also provides the means to implement more complex ME algorithms, which may take advantage of advanced prediction techniques that significantly accelerate the search procedure.

After a brief discussion of other architectures that have been proposed in literature, presented in Sect. 2, these two distinct approaches will be extensively described in Sects. 3 and 4. Section 5 presents their integration with the remaining video encoding system, as well as a thorough comparison in terms of their efficiency, flexibility and execution performance, in order to better demonstrate the application area of each of these two processing alternatives. Finally, Sect. 6 will conclude the paper.

2 Related work

The vast majority of the hardware motion estimators that have been proposed in the literature [11] consist of custom ASIC implementations of the full-search BM (FSBM) algorithm, mostly due to its regularity and data independence. The need for faster search algorithms to achieve real-time ME has also led to the proposal of a few architectures implementing sub-optimal search strategies [6, 9]. However, the highly irregular control-flow that characterizes such algorithms tends to compromise the efficiency of such architectures and therefore has limited its application to generic programmable systems.

With the advent of new video coding standards, such as the H.264/AVC, and with the proliferation of stream-based applications and of mobile devices, the limitations of conventional processors, both in terms of performance and of power consumption, have become more evident. As a result, a new class of programmable processors and coarse-grain reconfigurable architectures has emerged in the latter years to speed up computationally intensive and performance critical applications, such as real-time ME. These architectures combine the performance frequently offered by ASICs with the flexibility of reconfigurable and programmable processors, by embedding instruction set processors, usually with RISC or VLIW architectures, in coarse-grain reconfigurable arrays. These architectures allow a significant increase of the processing efficiency, by improving the performance without increasing the power consumption. Nevertheless, until recently none of these architectures has been widely adopted in industry, due to the novelty of its programming models, the difficult interface between the reconfigurable array and the host processor and the lack of an efficient compiler technology supporting such structures. However, this scenario has recently changed, with the proposal of some new coarse-grain architectures, such as PACT XPP [14], Silicon Hive [5], IMEC ADRES [13] and ARM OptimoDE [2].

The PACT XPP consists of a coarse-grain reconfigurable array, using a packet-oriented communication network with powerful run-time reconfiguration mechanisms. The XPP array includes only three different types of functional blocks: I/O units, RAM and ALU processing array elements. The ALU processing elements can only execute logic and arithmetic operations. This architecture is therefore most suitable to implement co-processor cores for the processing of high bandwidth datastreams in very demanding applications, such as multimedia processing and wireless.

The Silicon Hive architecture presents significant differences regarding the PACT XPP. It consists of a hierarchical array composed of multiple cells, where each cell has its own thread of control. These cells have more complex processing elements than the PACT XPP architecture, allowing the execution of more complex operations such as load/store, branch or multiply and accumulate (MAC) instructions. Silicon Hive has also implemented a complete development framework, which allows the design of customized, programmable, VLIW cores with ultralong instructions, capable of efficiently executing compute- and data-intensive tasks.

Unlike the PACT XPP and the Silicon Hive architectures, the IMEC ADRES and the ARM OptimoDE architectures combine a coarse-grain reconfigurable array and a VLIW processor, to allow the complete execution of an application. By using such organization, the computational-intensive kernels are mapped into the array and the VLIW processor controls its processing, while performing concurrent processing. Moreover, in these architectures the array functional units are also capable of executing more complex operations, such as multiplications. Like the Silicon Hive architecture, both architectures also present powerfull development frameworks that greatly simplify their programming model.

Contrasting with all these approaches, the two architectures proposed in this paper are characterized by fine-grain structures specialized for high-performance ME.

They both present a simple programming model and allow a straightforward integration with any generic instruction set processor. By using this approach, any of these architectures can be used either as a ME co-processor in video coding systems, or as a specialized functional unit of one of the above-mentioned coarse-grain architectures.

3 *Architecture A*-reconfigurable structure for regular search patterns

This reconfigurable structure represents a quite innovative mechanism to implement highly flexible and programmable ME units. The simple and efficient hardware architecture, composed of an adaptable controller and a single data-path, is based on a reconfigurable mechanism that expresses any considered regular search pattern, as well as its initial search locations, through well-defined data-structures. Such data is tightly integrated in hardwired control units and is used to guide the search procedure [15].

3.1 Search algorithm configuration mechanism

By analyzing the characteristics of most fast search algorithms, it can be observed that they usually evolve iteratively, depending on the value of the distance measures computed in run-time. Even so, a certain level of regularity can still be identified and used to represent the search patterns of these algorithms through simple data structures. Hence, the main idea behind this run-time reconfiguration mechanism is to fill small lookup tables with these data structures and integrate them in the controller component, in order to guide the search procedure in hardware motion estimators. These structures will contain all the data required to implement the reconfiguration mechanism, namely:

Cartesian displacements (Δx, Δy)—accumulated in each search step to compute the output MV;
Raster displacement—provides the address offset within a frame that corresponds to the displacement of the current MV; although redundant, it provides a significant reduction in the required hardware;
Next pattern address (NPA)—every time a given search step finishes, it is used to address the next pattern of candidate MBs that will be considered in the following search step;
Step end (StE)—flag that signals the end of the current search step; it also causes the loading of the address corresponding to the MB with the smallest error;
Search end (SeE)—flag that signals the end of the search process.

As an example, the search patterns of the well known 3SS and DS fast ME algorithms are illustrated in Fig. 1. The corresponding data structures are represented in Table 2. As it can be seen, for each MB of the current image the 3SS algorithm considers 25 candidate MBs. On the other hand, the DS algorithm visits a variable number of candidate MBs that depend on the image data corresponding to the frames being processed (on average, only 17 different candidate blocks are evaluated). By using this configuration for the required data structures, conditional jumps can easily be implemented and programmed, so that a given section of the pattern memory can be efficiently reused. This fact keeps the storage requirements for the whole data structure reasonably

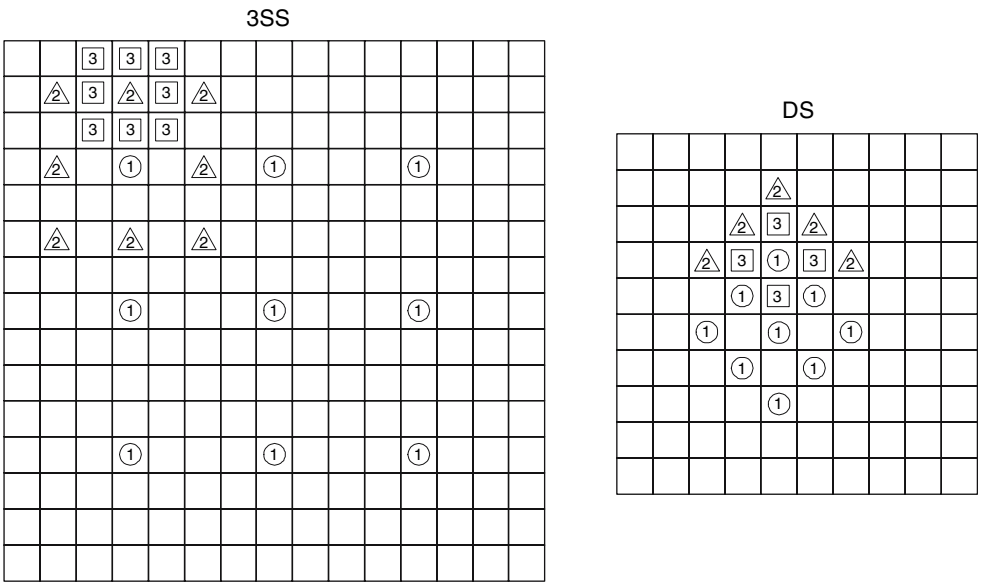


Fig. 1 3SS and DS fast ME patterns

small, while fairly complex algorithms can still be implemented. For a given search area composed of $N \times N$ pixels, a reasonable worst case estimate for the size of this data structure is N^2. As an example, for $N = 16$, 256 different pattern addresses will be used. However, significantly less positions are usually required, in practice, for the most used classes of ME algorithms.

The flowchart of the overall mechanism for searching the "best" candidate block for a given MB is presented in Fig. 2. The system starts by fetching the current search pattern from memory and uses the raster displacement to compute the base address of the candidate MB. The candidate MB is then fetched from the frame memory and the SAD cost function is computed and compared with the stored smallest SAD value that was found so far. If the current cost measure is smaller than the stored one, the candidate MB becomes the best known match. This causes the SAD output value to be stored in the *smallest-SAD* register and the corresponding pattern address to be stored in the *smallest-SAD-address* register. This procedure is successively repeated until the *StE flag* is activated. As soon as this flag is set, the values stored in the *smallest-SAD* registers are used to start the following search step. This centers the next search process in the candidate MB that yielded the smallest SAD value and addresses the following search pattern corresponding to the next search step. Simultaneously, the *StE flag* also triggers the accumulation of the partial displacements corresponding to the smallest SAD pattern (Δx and Δy) in the MV output accumulator. At any time, the output MV accumulator holds the sum of the Cartesian displacements of the points that produced the smallest SAD:

$$\mathrm{MV} = \sum_{s=1}^{K} \left(\mathrm{MV}(P_s^I) \mid \mathrm{SAD}(P_s^I) \leq \mathrm{SAD}(P_s^i) \right) \tag{2}$$

for $(0 \leq i, I \leq n_s)$, where K represents the number of search steps, MV(P) denotes the Cartesian displacements corresponding to pattern P, P_s^i represents the ith candidate MB of the sth search step and n_s is the number of candidate MBs checked in the sth search step.

In Fig. 3, a simple example that tests nine candidate blocks in three search steps is shown. In this example, all the NPAs of the first step point to the same pattern address-4. The same happens during the second step, where the NPA also assumes the single value-7.

3.2 Architecture

The proposed ME architecture for regular search patterns can be represented by three main blocks (see Fig. 4) that are connected to a pair of frame memories:

Table 2 Data structures corresponding to the 3SS and the DS fast ME algorithms

PA	3SS algorithm					DS algorithm				
	SeE	StE	NPA	Δx	Δy	SeE	StE	NPA	Δx	Δy
0			9	0	0			49	0	0
1			9	−4	−4			9	0	−2
2			9	0	−4			15	1	−1
3			9	4	−4			19	2	0
4			9	−4	0			25	1	1
5			9	4	0			29	0	2
6			9	−4	4			35	−1	1
7			9	0	4			39	−2	0
8		1	9	4	4		1	9	−1	−1
9			18	0	0			49	0	0
10			18	−2	−2			39	−2	0
11			18	0	−2			45	−1	−1
12			18	2	−2			9	0	−2
13			18	−2	0			15	1	−1
14			18	2	0		1	19	2	0
15			18	−2	2			49	0	0
16			18	0	2			9	0	−2
17		1	18	2	2			15	1	−1
18				−1	−1		1	19	2	0
19				0	−1			49	0	0
20				1	−1			9	0	−2
21				−1	0			15	1	−1
22				1	0			19	2	0
23				−1	1			25	1	1
24				0	1		1	29	0	2
25	1	1		1	1			49	0	0
...			...			...	...	...	...	...
...			...			...	...	...	...	...
49									0	−1
50									1	0
51									0	1
52						1			−1	0

Address generation unit (AGU)—responsible for the generation of the several addresses that are required to access each frame memory;

SAD unit (SADU)—computes the disparity measure between each pair of MBs; it also contains a scratchpad memory [3] whose function is, besides storing the current MB, gathering and alignment of the data from the candidate MB;

Search decision unit (SDU)—chooses the best candidate block for each search step; it contains a comparator and all the registers and accumulators that are required to hold the several key values throughout the search

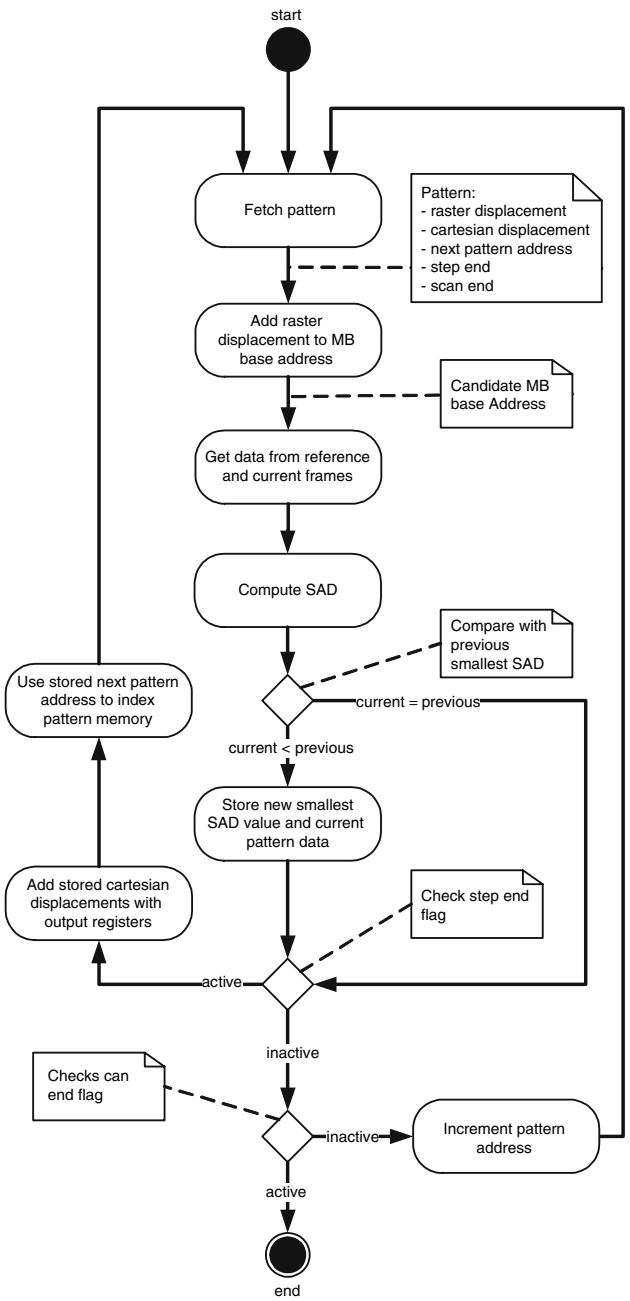


Fig. 2 Search mechanism flowchart

process; it is also capable of determining whether the current search coordinates are valid for the MB under processing.

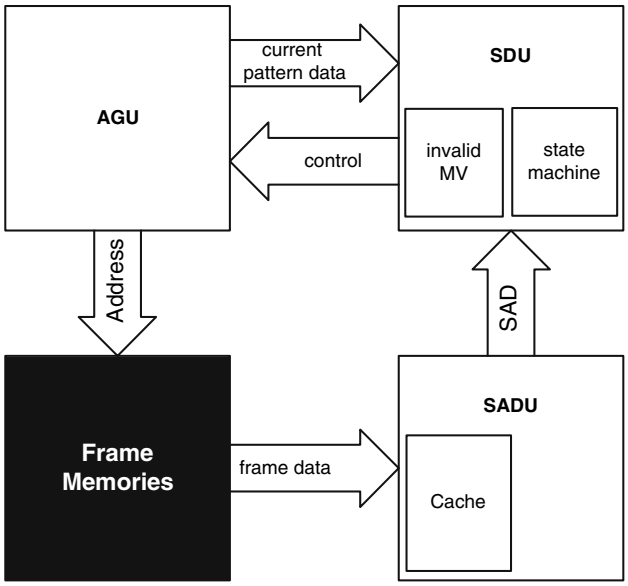


Fig. 4 Top level block diagram of architecture A

A more detailed description of all these units is found in Fig. 5 and in the next subsections, considering MBs with 16 × 16 pixels.

3.2.1 *Address generation unit (AGU)*

The previously described pattern data structures are stored in a small dedicated memory located in the AGU. By using the raster displacement information of each pattern, the AGU is capable of addressing all the pixels for each MB.

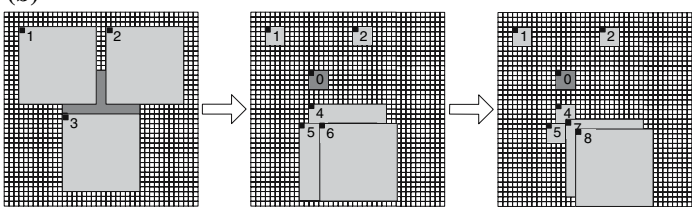
In the block diagram of the AGU, represented in Fig. 5, all the addresses that are supplied to the frame memories are expressed in the raster format. For each step of the search procedure, the incremental displacement is added to the step base address. Hence, the result of this sum will represent the top-left corner address of the target candidate MB. This value is used by the MB pattern scan block to generate all the pixel addresses for each MB. These addresses are directly used to access both the reference and the search frame memories. Moreover, the first time when the reference frame memory is read, the corresponding pixels are also copied into the scratchpad memory of the

Fig. 3 Search process example, assuming that pattern addresses 3 and 6 yield the smallest error for the corresponding search step

(a)

PA	ScE	StE	NPA	Δx	Δy
0			4	0	0
1			4	-9	-9
2			4	9	-9
3		1	4	0	9
4			7	0	-2
5			7	-2	2
6		1	7	2	2
7				-1	-1
8	1			1	1

(b)



SADU, so that latter accesses to this data are performed from this faster and local memory.

On the other hand, the *Pattern Generator* block is responsible for computing the several addresses that index the *Pattern Memory*. This pattern address is incremented each time the SDU provides an impulse in the *inc_patt* output signal, meaning that the last address of a given MB has been reached. Hence, during the processing of the candidate blocks within a given search step, the AGU is able to continuously output one new address per clock cycle. The pattern address is also incremented when an invalid search step is detected, which only occurs for pixels laying outside the frame borders. However, in this particular situation the AGU will output invalid addresses for the two subsequentclock cycles. The *ld_patt* signal is active whenever a search step finishes and a new next pattern address has to be loaded from the pattern memory.

Finally, a quite important role is played by the *sel_pat* input signal, which is used to select the data structure corresponding to the selected search algorithm. By reserving a fixed number of memory positions to each search algorithm (in particular, one that is an integer power of two), the *sel_pat* input signal actually corresponds to just some extra addressing bits.

3.2.2 Search decision unit (SDU)

The search decision unit operates at three different levels. In the first level, it controls the updating of the MV that is used in each step. For each pattern, the MV accumulated in the previous steps is added to the relative MV of the current candidate MB. These values, along with the base address and the next pattern address, are stored until the SADU completes the computation. As soon as the SAD output is available, it is compared to the smallest SAD value. If the current SAD output is smaller than the stored one, this new SAD value and the corresponding relative coordinates are stored in the registers depicted in Fig. 5.

In the second operation level, the decision block detects the patterns that hold invalid MVs for the current MB, due to restrictions imposed by the frame borders.

In the third and last level, the search decision unit controls the motion estimator based on a finite state machine (FSM). This FSM is composed of seven different states and was directly implemented in hardware. The first state ("load" state) activates the *ld_patt* input line of the AGU, which causes the AGU to load a new pattern and to clear all the counters. It also causes the MV registers to load the currently accumulated MVs. During the "running" state, the system tests different search locations, according to the specific ME algorithm that is being used, looking for the one yielding the smallest SAD. The processor will remain in this state to process the current pattern, until an active *StE* signal is received and the last MB line is reached, which causes the transition into the "last" state.

3.2.3 SAD unit (SADU)

At each clock cycle, the arithmetic part of the SAD unit performs, in parallel, eight absolute differences and

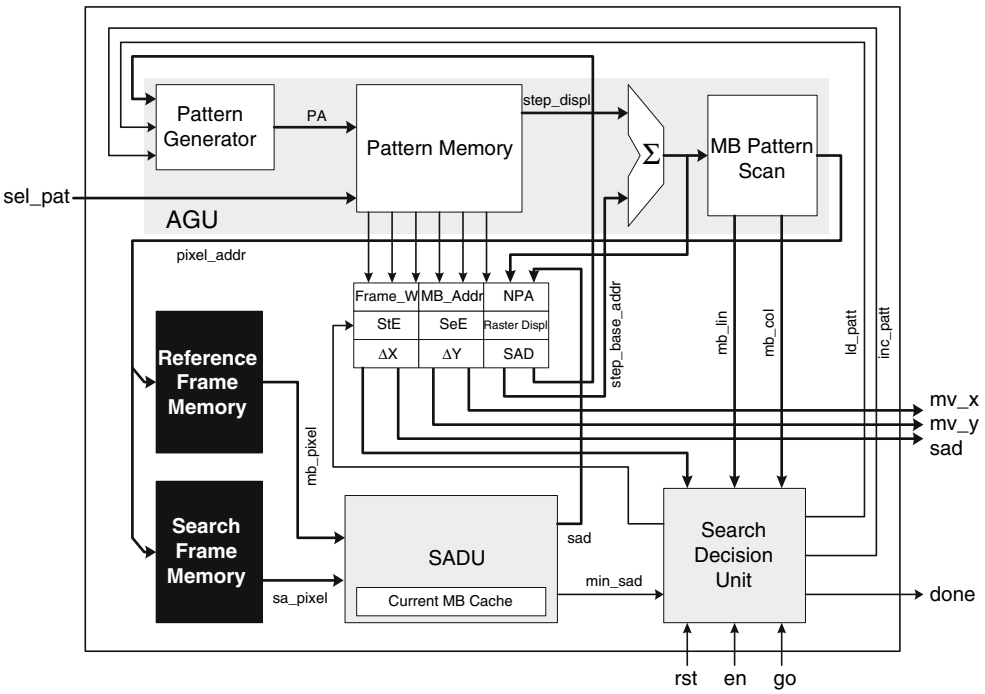


Fig. 5 Reconfigurable architecture A

accumulates the obtained results. Therefore, 32 clock cycles are required to compute the SAD value for a 16 × 16 pixels MB. In order to provide the set of eight pairs of operands to the arithmetic part, the SADU also stores part of the candidate MB in its scratchpad memory. Hence, while the current MB is completely buffered in the scratchpad memory, because it is used throughout the whole searching process, the candidate buffer only holds a single line of the MB under processing.

3.3 Limitations

Despite the high performance levels provided by this quite efficient architecture and of its reconfigurable properties, it is worth noting that it may still present some limitations, mainly, in what concerns the selection of the adopted search procedure. In fact, although the AGU and the SADU of the proposed architecture can be run-time or offline reconfigured to better adapt the processors' structure to the requirements of the target video coding platform (i.e., in terms of the motion estimation algorithm, hardware resources, power consumption, memory bandwidth, channel bandwidth, etc.), for specific classes of more complex ME algorithms with irregular search patterns or search procedures that also exploit temporal correlation, other programmable and more flexible structures are required. In the next section, one of such structures will be proposed.

4 *Architecture B*-programmable specialized architecture for motion estimation

The presented programmable and specialized architecture for ME was tailored to efficiently program and implement a broad class of powerful, fast and/or adaptive ME search algorithms. This architecture supports the most used MB structures, such as the traditional fixed 16 × 16 pixels block size, adopted in the H.261/H.263 and the MPEG-1/MPEG-2 video coding standards, or even any other variable block-size structures, adopted in the H.264/AVC coding standards. Such flexibility was attained by developing a simple and efficient micro-architecture to support a minimum and specialized instruction set, composed of only eight different instructions, that was specifically defined for ME. In the core of this architecture, a data-path was specially designed around a low-power arithmetic unit that efficiently computes the SAD function. Furthermore, the several control signals are generated by a quite simple and hardwired control unit [8].

4.1 Instruction set

The instruction set architecture (ISA) of the proposed application specific instruction set processor (ASIP) was designed to meet the requirements of most ME algorithms, including some recent approaches that adopt irregular and unpredictable search patterns, such as the data-adaptive ones. Such ISA is based on a register–register architecture and provides a quite reduced number of different instructions (8), that focus on the set of operations that are most widely used in ME algorithms. The selection of this register–register approach comes as a direct consequence of the simplicity and efficiency that it offers, allowing the design of simpler and less hardware consuming circuits.

On the other hand, it also offers an increased efficiency level, mainly due to its large number of general purpose registers (GPRs), which provides a reduction of the memory traffic and, consequently, a decrease of the program execution time. The amount of registers that compose the register file therefore results as a trade-off between hardware resources, memory traffic and size of the program memory. For the proposed ASIP, the register file consists of 24 GPRs and 8 special purpose registers (SPRs), capable of storing one 16-bit word each. This configuration optimizes the ASIP efficiency since: (1) the amount of GPRs is enough to allow the development of efficient, yet simple, programs for most ME algorithms; (2) the 16-bit data type covers all the possible values that are commonly assigned to variables in ME algorithms; and (3) the 8 SPRs are efficiently used as configuration parameters for the implemented ME algorithms and for data I/O.

The operations supported by the proposed ISA are grouped in four different categories of instructions, as it can be seen in Table 3, and were obtained as the result of a thorough analysis of the execution of several different ME algorithms. The encoding of these instructions into binary representation was performed using a fixed 16-bit format. For each instruction, it is specified an opcode and up to three operands, depending on the instruction category. Such encoding scheme therefore provides a minimum bit wasting for the instruction encoding and eases the corresponding decoding, thus allowing a good trade-off between the program size and the efficiency of the architecture. A brief description of all the operations of the proposed ISA is presented in the following sections.

4.1.1 Control operation

The jump control operation, `J`, introduces a change in the control-flow of a program, by updating the program counter with an immediate value that corresponds to an effective address. This instruction has a 2-bit condition

Table 3 Instruction-set architecture of the proposed ASIP

Opcode	Mnemonic	Instruction Category	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
000	LD	Memory data transfer	opcode			t	-											
001	J	Control	opcode			cc		-	address									
010	MOVR	Register data transfer	opcode			Rd					-			Rs				
011	MOVC	Register data transfer	opcode			t	Rd				constant							
100	SAD16	Graphics	opcode			-	Rd				Rs1			Rs2				
101	DIV2	Arithmetic	opcode			-	Rd				Rs			-				
110	ADD	Arithmetic	opcode			-	Rd				Rs1			Rs2				
111	SUB	Arithmetic	opcode			-	Rd				Rs1			Rs2				

field (cc) that specifies the condition that must be verified for the jump to be taken: always or in case the outcome of the last executed arithmetic or graphics operation (`SAD16`) is positive or zero, or even negative for the case of arithmetic operations. Hence, not only is this instruction quite important for algorithmic purposes but also for improving the code density, since it allows a minimization of the number of instructions required to implement a ME algorithm and therefore a reduction of the required capacity of the program memory.

4.1.2 Register data transfer operations

The register data transfer operations allow the loading of data into a GPR or SPR of the register file. Such data may be the content of another register, in the case of a simple move instruction, `MOVR`, or an immediate value for constant loading, `MOVC`. Due to the adopted instruction coding format, the immediate value is only 8-bit width. A control field (t) sets the loading of the 8-bit literal into the destination register upper or lower byte.

4.1.3 Arithmetic operations

The `ADD` and `SUB` instructions support the computation of the coordinates of the MBs and of the candidate blocks, as well as the updating of control variables used in loops. On the other hand, the `DIV2` instruction (integer division by two) allows, for example, to dynamically adjust the search area size, which is most useful in adaptive ME algorithms. Moreover, these three instructions also provide some extra information about their outcome, which can be used by the jump (`J`) instruction to conditionally change the control-flow of a program.

4.1.4 Graphics operation

The `SAD16` operation allows the computation of the SAD similarity measure between a MB and a candidate block. To do so, this operation computes the SAD value considering two sets of 16 pixels (the minimum amount of pixels for a MB in the MPEG-4 video coding standard) and accumulates the result to the contents of a GPR. The computation of the SAD value for a given (16 $\times$ 16) pixels candidate MB therefore requires the execution of sixteen consecutive `SAD16` operations. To further improve the algorithm efficiency and reduce the program size, both the horizontal and vertical coordinates of the line of pixels of the candidate block under processing are also updated with the execution of this operation. Likewise the previously described arithmetic operations, the outcome of this operation also provides some extra information that may be used by the jump (`J`) instruction to conditionally change the control-flow of a program.

4.1.5 Memory data transfer operation

The processor comprises two fast and small scratchpad local memories to store the pixels of the MB under processing and of its corresponding search area. To improve the processor performance, a memory data transfer operation (`LD`) was also included to load the pixels data into these memories. Such operation is carried out by means of an AGU, which generates the set of addresses of the corresponding internal memory, as well as of the external frame memory, that are required to transfer the pixels data. The selection of the target memory device is carried out by means of an 1-bit control field (t), which is used to specify the type of image data that is loaded into the local memory: either corresponding to the current MB or to the corresponding search area.

4.2 Micro-architecture

The proposed ISA is supported by a specially designed micro-architecture, following strict power and area driven policies to allow its implementation in a wide range of target implementation platforms, such as portable and mobile devices. This micro-architecture presents a modular

structure and is composed of simple and efficient units that optimize the data processing, as it can be seen in Fig. 6.

4.2.1 Control unit

The control unit is characterized by its low complexity, due to the adopted fixed instruction encoding format and a careful selection of the opcodes for each instruction. This not only provided the implementation of a very simple and fast hardwired decoding unit, which enables almost all instructions to complete in just one clock cycle, but also allowed the implementation of effective power saving policies within the processor functional units, such as clock gating and operating frequency adjustment.

4.2.2 Datapath

For the particular case of the most complex and specific operations, such as the `LD` and the `SAD16` instructions, the datapath also includes specialized units to improve the efficiency of these operations: an AGU and a SADU, respectively.

The `LD` operation is executed by a dedicated AGU, which is capable of fetching all the pixels of both a MB or an entire search area. To maximize the efficiency of the data processing, this unit can work in parallel with the remaining functional units of the micro-architecture. By using such feature, programs can be highly optimized, by rescheduling the `LD` instructions in order to allow data fetching from the external memory to occur simultaneously with the execution of other parts of the program that do not depend on this data.

The SADU can execute the `SAD16` operation in up to sixteen clock cycles and is capable of using the ALU to update the line coordinates of the candidate block. The number of clock cycles that is required for the computation of a SAD value is imposed by the type of architecture that is selected in the configuration of this unit, which depends on the performance and hardware resources constraints specified at design time. Thus, applications imposing more severe constraints in the amount of used resources may adopt a configuration based on a serial processing architecture, that reuses hardware but takes more clock cycles to compute the `SAD16` operation, while others without so strict requisites may adopt a configuration based on a parallel processing architecture that is able to compute the `SAD16` operation in only one single clock cycle.

5 Prototype and experimental results

5.1 Prototype

To validate the functionality of the two proposed reconfigurable architectures for ME in a practical realization, a video encoder was developed and implemented in a Xilinx ML310 development platform [20], making use of a 100 MHz 256 MB DDR memory and of a Virtex-II Pro XC2VP30 FPGA device from Xilinx. Besides all the implementation resources offered by this reconfigurable device, this platform also provides two Power-PC processors, several Block RAM (BRAM) modules and high speed on-chip bus-communication links, to enable the interconnection of the Power-PC processors with the user developed hardware circuits.

The implemented video encoding system makes use of some of these resources. Its base configuration consists of a software implementation of the H.263 video encoder provided by Telenor R&D (TMN5) [18], running on one of the Power-PC 405 D5 processors at 300 MHz and built into the FPGA Block RAM (BRAM) and the DDR memory module. The encoder implementation presents some optimizations, regarding to the original version provided by Telenor R&D, to make its implementation more efficient in

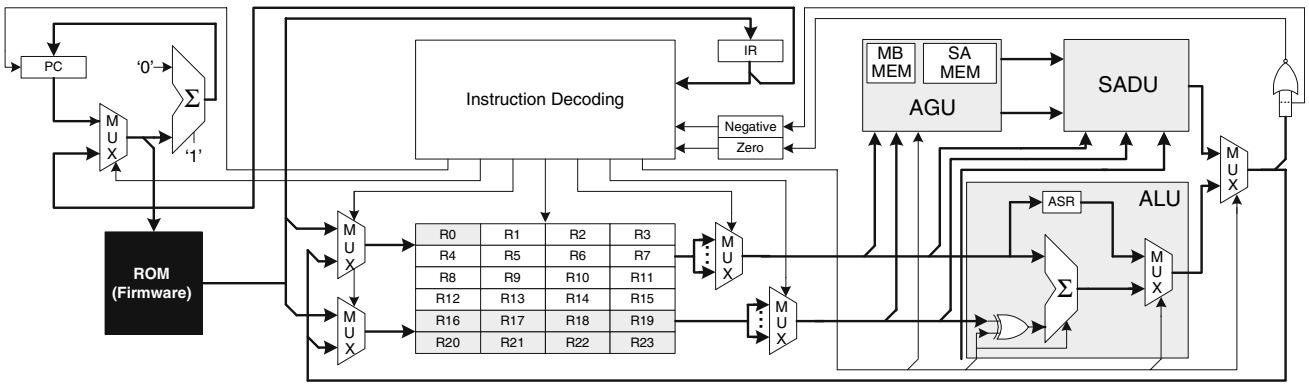


Fig. 6 Architecture of the proposed ASIP (SPRs are shaded in the register file)

embedded systems, namely, the redesign of all the functions used in ME and the declaration of all variables in these functions using the prefix *register*, to optimize the processing time. To maximize the performance of the encoder, the linker script of the video encoding system was also modified. In particular, both the application section *.text* and section *.me* (a special segment in the system memory address space that holds the pixels of the reference block and of its corresponding search area candidate blocks) were located in two distinct 128 kB FPGA BRAM modules, while the *.data*, *stack* and *heap* sections were located in the DDR memory module, due to its large size (more than 256 kB).

The proposed ME configurable structures perform as a co-processor of the main video encoder system, by computing, in parallel, the several MVs that are required by the encoder to compute the prediction error of the video signal. To do so, the interconnection between the Power-PC processor and the proposed reconfigurable architectures for ME is implemented by using both the high-speed 64-bit processor local bus (PLB) and the general purpose 32-bit on-chip peripheral bus (OPB), where the Power-PC is connected as the master device. Such interconnect buses are used not only to exchange the control signals between the Power-PC and the ME co-processor, but also to send all the required data to the adopted ME structure, namely, the pixels of the candidate and reference blocks and the ME algorithm search pattern or program code.

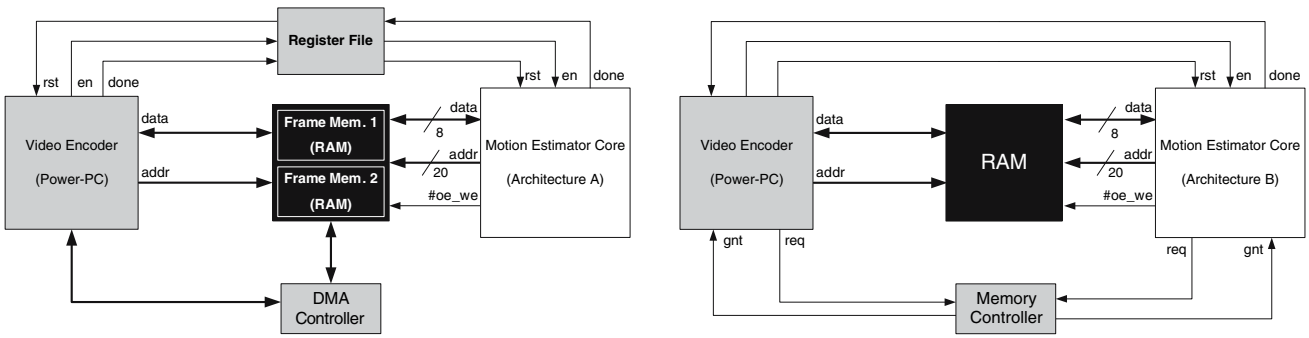
5.2 Integration with the video encoding system

The integration of the two proposed reconfigurable architectures for ME with the video encoding system is quite similar for the two structures, as it can be seen in Fig. 7. Such integration was designed both to allow efficient data transfers from the external frame memories and to efficiently export the coordinates of the computed best matching MVs to the video encoder.

In the proposed reconfigurable architecture for regular search patterns (architecture A), data transfers with the video encoding system are mainly performed through a register file and two frame memories, as it is illustrated in Fig. 7. The register file is composed of four 64-bit registers, which are used not only to configure and control the overall operation of the motion estimator, but also to store the computed MV at the end of the search procedure. In this architecture, data is transferred between the video encoding system and the register file/frame memories/pattern memory using 64-bit width communication channels, which allow DMA data transfers into the two frame memories. Moreover, since the frame memories are only used to store the reference and current frames, there is no need for control signals to implement the handshaking with the bus master.

Unlike the proposed reconfigurable structure, in the programmable architecture (architecture B) the interface with the external frame memory was designed to allow 8-bit data transfers from a 1 MB memory address space. In fact, the pixel values for ME are usually represented using only 8-bits and MVs are estimated using pixels from the current and previous frames (each frame consists of 704×576 pixels in the 4CIF image format). Consequently, three I/O ports are used for the data transfers, as it can be seen in Fig. 7: (1) a 20-bit output port, that specifies the memory address for the data transfers (*addr*); (2) an 8-bit bidirectional port, for transferring the data (*data*); and (3) a 1-bit output port, that sets whether it is a load or store operation (*#oe_we*). Moreover, two extra 1-bit control ports (*req* and *gnt*) are also used to implement the required handshake protocol with the bus master, since the external frame memory is to be shared with the video encoder and with the ME circuit.

To minimize the number of required I/O connections, the coordinates of the best matching MVs are also outputted through the *data* port. Nevertheless, such operation requires two distinct clock cycles for its completion: a first one to output the 8-bits of the MV horizontal coordinate



(a) Architecture A **(b) Architecture B**

Fig. 7 Interface with the video encoding system

and a second one to output the 8-bits of its vertical coordinate value. In addition, every time a new value is outputted through the *data* port, the status of the *done* output port is toggled, in order to signal the video encoder that new data awaits to be read at the *data* port of the co-processor.

The co-processor's firmware, corresponding to the compiled assembly code of the considered ME algorithm, is also uploaded into the program RAM through the *data* port. To do so, the co-processor must be in the programming mode, which it enters whenever a high level is simultaneously set into the *rst* and *en* input ports. In this operating mode, as soon as the master processor acquires the bus ownership, it supplies memory addresses through the *addr* port and uploads the corresponding instructions into the internal program RAM. The co-processor exits this programming mode as soon as the last memory position of the 2 kB program memory is filled in. Once again, each 16-bit instruction takes two clock cycles to be loaded into the program memory, which is organized in the little-endian format.

Despite the configurable ME architecture that is chosen to implement the ME co-processor, the operating principle of the developed prototyping video encoding system is similar for the two proposed structures. It only consists of three different tasks: (1) configuration of the adopted ME co-processor (MB size, search area size, image width and image height); (2) data transfers from the Power-PC to the ME co-processor, which occur on demand by the motion estimator; and (3) data transfers from the ME co-processor to the Power-PC, mainly used to output the coordinates of the best-matching MV and of the corresponding SAD value.

5.3 Experimental results

To assess the performance of the proposed reconfigurable architectures for ME, several different ME algorithms, such as the FSBM, the 3SS, the DS and the adaptive MVFAST, were implemented using these structures. These algorithms not only were programmed with the proposed instruction set for the presented programmable architecture, but also had their search patterns properly generated and programmed into the AGU of the reconfigurable architecture for regular search patterns.

To carry out these implementations, the two proposed reconfigurable architectures for ME were described using both behavioral and fully structural parameterizable IEEE-VHDL and implemented in the Virtex-II Pro XC2VP30 FPGA device of the ML310 board, using the EDK 8.1i and ISE 8.1i tools from Xilinx. The considered configurations adopted different SADU modules for the two architectures. The proposed reconfigurable architecture for regular search patterns (architecture A) simultaneously computes the absolute difference values for 8 pixels, thus only requiring 32 clock cycles to compute the SAD value for an entire macroblock. On the other hand, the presented programmable architecture (architecture B) makes use of a power efficient serial processing structure for the SADU module [7]. Such architecture was selected as the result of a compromise between the amount of hardware resources required for its implementation, the circuit power consumption and its usability for real-time operation. Based on previous research work [7], we have concluded that for ME based on a single reference frame and for image formats up to CIF, a serial structure for the SADU presents the best trade-off between the parameters formerly described. However, as it was previously mentioned, depending on the target application the proposed architectures can be reconfigured to use other SADU structures that represent different trade-offs.

Table 4 presents the experimental results that were obtained with the implementation of the developed video encoding system in the Virtex-II Pro XC2VP30 FPGA device using the considered ME algorithms for the two proposed ME architectures. The obtained results, in what concerns the used BRAM modules, demonstrate that the two proposed structures have identical memory requirements. In both architectures, one dual-port BRAM module is used to store the pixel data for both the reference and search areas, while the other module is used to store either the pattern data structures of the implemented ME algorithm (architecture A) or the corresponding compiled assembly code (architecture B). Nevertheless, to optimize the data transfers, the organization of these BRAMs is quite different for the two architectures.

The results presented in Table 4 evidence other main differences between the two architectures. Although it is not evident at a first sight, by carefully analyzing these results it is possible to demonstrate that architecture A provides higher efficiency levels when implementing ME algorithms characterized by regular search patterns. In fact, although the two structures present quite similar hardware requirements, most of the resources used by architecture A are allocated to implement the 32-stage pipelined SADU, which allows a fast computation of the SAD value for a

Table 4 Experimental results obtained with the implementation of the video encoder in the Xilinx ML310 development board

Unit	# Slices		# LUTs		# BRAMs	F (MHz)
Architecture A	2213	16%	2031	7%	2	66.19
Architecture B	2046	14%	1844	6%	2	85.84

Table 5 Average time to estimate a MV considering several different algorithms and video sequences, by using, as reference, the software implementation

Algorithm	Software		Using architecture A				Using architecture B			
	Time (ms)		Time (ms)		Speed-up		Time (ms)		Speed-up	
FSBM	70.98	0.11	0.78	0.00	91	0.1	1.32	0.00	54	0.1
3SS	7.21	0.27	0.16	0.01	44	1.6	0.20	0.00	34	1.3
DS	7.13	0.53	0.18	0.01	39	2.6	0.21	0.02	31	2.3
MVFAST	1.16	0.26	-	-	-	-	0.08	0.02	12	2.6
	Avg	StDev	Avg	StDev	Avg	StDev	Avg	StDev	Avg	StDev

given macroblock in only 32 clock cycles. On the other hand, the fewer hardware resources required by architecture B are mainly due to the usage of a serial SADU, which needs 256 clock cycles to compute the SAD value for a 16×16 pixels MB. Even so, more resources still need to be allocated to implement the datapath and the control unit. Consequently, despite its lower maximum allowed operating frequency, architecture A actually presents a higher execution performance when regular search patterns are adopted to estimate the MVs. In fact, the main drawback of this architecture arises from its reduced flexibility, which does not allow the implementation of more complex or adaptive ME algorithms. Such fact is often a decisive requirement in the implementation of the most modern video coding systems, based on the H.264/AVC encoding standard. Hence, the higher flexibility provided by architecture B, mostly owed due to its programmable capability, makes it more suitable for the implementation of such algorithms, as it can be seen from the experimental results presented in Table 5.

Two different scenarios, that only differ in the implementation of the ME technique, were considered to obtain the experimental results presented in Table 5 with the proposed video coding system. In the first scenario, the ME algorithms were entirely implemented in software, just like the rest of the video coding system. Then, for the other scenario, the ME procedure was performed in hardware, by using the two proposed configurable architectures. In both cases, one timer module of the Power-PC processor was used to assess the time elapsed in the ME procedure. This operation was performed by considering search areas with 16×16 candidate locations and a small set of frames (the first 20) of the following benchmark video sequences: *Mobile, Carphone, Table Tennis*, and *Bream / Bus*. These sequences are well known test video sequences with quite different characteristics, both in terms of the amount of movement and spatial detail.

The results presented in Table 5 represent average values that were obtained by encoding the five referred benchmark video sequences using the selected ME algorithms. As it was predicted before, they evidence that architecture A is more efficient than architecture B to implement fast ME algorithms with regular search patterns. However, this architecture does not allow the implementation of more complex and efficient algorithms, capable of further decreasing the estimation time. Such fact is clearly seen in Table 5, where it is shown that the implementation of the motion vector field adaptive search technique (MVFAST) provides an average speed-up of over 2.5 times when compared with the other fast search algorithms implemented in architecture A. This advantage is even greater when the implementation of the MVFAST adaptive algorithm, in architecture B, is compared with the several fast sub-optimal search algorithms also implemented in architecture B (the obtained speed-up is over 3.2 times). In addition, the usage of such more complex ME algorithms also often provide an increase of the PSNR of the encoded video sequences [19], at the cost of an inherent usage of higher hardware resources.

6 Conclusions

This paper presents two classes of highly efficient reconfigurable architectures for ME with different optimization trade-offs. While the first class consists of a high performance reconfigurable structure optimized for regular search patterns, the second one makes use of a programmable specialized structure that allows the implementation of any ME algorithm, although it is particularly well suited for the implementation of more complex and data adaptive search patterns.

The performance of these architectures was thoroughly assessed by embedding them as ME co-processors in a video encoding system, implemented in a Xilinx ML310 prototyping environment. Experimental results obtained with the implementation of several different ME algorithms (FSBM, 3SS, DS and MVFAST) in these co-processors have shown that the two classes of architectures allow the estimation of MVs in real-time (above 25 fps) for both the QCIF and CIF image formats. Furthermore, such results proved the higher efficiency of the reconfigurable

structure for the implementation of regular search patterns, typically used in low complexity video encoding systems. The programmable architecture has shown to be more suitable for the execution of the most irregular or data adaptive search patterns. In fact, despite the slightly lower performance levels provided by the programmable architecture, it allows the implementation of more complex and/or adaptive search procedures, required for the implementation of the newest video coding standards. Such algorithms not only involve a smaller amount of search steps, but also lead to an inherent increase of the overall PSNR of the encoded video sequences [19]. Moreover, by operating at lower clock frequencies, this architecture may still provide a further reduction of the power consumption of the ME co-processor.

Acknowledgments This work has been supported by the POSI program and the Portuguese Foundation for Science and for Technology (FCT) under the research project Adaptive H.264/AVC Motion Estimation Processor for Mobile and Battery Supplied Devices (AMEP) POSI/EEA-CPS/60765/2004.

References

1. Ang, S., Constantinides, G., Luk, W., Cheung, P.: The cost of data dependence in motion vector estimation for reconfigurable platforms. In: Proceedings of the International Conference on Field Programmable Technology—FPT'2006, IEEE, pp. 333–336 (2006)
2. ARM: OptimoDE Data Engines. http://www.arm.com/products/DataEngines. ARM Ltd (2007)
3. Banakar, R., Steinkeand, S., Lee, B., Balakrishnan, M., Marwedel, P.: Scratchpad memory: a design alternative for cache on-chip memory in embedded systems. In: 10th International Symposium on Hardware/Software Codesign—CODES 2002, IEEE, pp. 73–78 (2002)
4. Bhaskaran V., Konstantinides K.: Image and Video Compression Standards: Algorithms and Architectures, 2nd edn. Kluwer, Dordrecht (1997)
5. Burns, G., Jacobs, M., Lindwer, M., Vandewiele, B.: Silicon Hive's Scalable and Modular Architecture Template for High-Performance Multi-Core Systems. Silicon Hive (2006)
6. Chao, W., Hsu, C., Chang, Y., Chen, L.: A novel hybrid motion estimator supporting diamond search and fast full search. In: IEEE International Symposium on Circuits and Systems 2002—ISCAS 2002, IEEE, pp. 492–495 (2002)
7. Dias, T., Roma, N., Sousa, L.: Low power distance measurement unit for real-time hardware motion estimators. In: International Workshop on Power and Timing Modeling, Optimization and Simulation—PATMOS'2006. Lecture Notes in Computer Science, vol. 4148, pp. 247–255. Springer, Berlin (2006)
8. Dias, T., Momcilovic, S., Roma, N., Sousa, L.: Adaptive motion estimator for autonomous video devices. EURASIP J. Embed. Syst. (2007)
9. Jehng, Y., Chen, L., Chiueh, T.: An efficient and simple VLSI tree architecture for motion estimation algorithms. IEEE Trans. Signal Process. **41**(2), 889–900 (1993)
10. Joint Video Team of ITU-T and ISO/IEC JTC1: ITU-T Recommendation H.264, "Advanced video coding for generic audiovisual services". ITU-T (2003)
11. Komarek, T., Pirsch, P.: Array architectures for block matching algorithms. IEEE Trans. Circuits Syst. **36**(10), 1301–1308 (1989)
12. Kuzmanov, G., Gaydadjiev, G., Vassiliadis, S.: The molen media processor: design and evaluation. In: Proceedings of the International Workshop on Application Specific Processors—WASP'2005, pp. 26–33 (2005)
13. Mei, B., Vernalde, S., Verkest, D., Man, H., Lauwereins, R.: ADRES: an architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix. In: 3rd International Conference on Field Programmable Logic Applications—FPL 2003, IEEE, pp. 61–70 (2003)
14. PACT (2006) XPP-III Processor Overview: White Paper. PACT XPP Technologies
15. Ribeiro, M., Sousa, L.: A run-time reconfigurable processor for video motion estimation. In: 17th International Conference on Field Programmable Logic and Applications (FPL), IEEE (2007)
16. Roma, N., Sousa, L.: Efficient and configurable full search block matching processors. IEEE Trans. Circuits Syst. Video Technol. **12**(12), 1160–1167 (2002)
17. Sima, M., Cotofana, S., Eijndhoven, J., Vassiliadis, S., Vissers, K.: An 8×8 IDCT implementation on an FPGA-augmented trimedia. In: Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2001) (2001)
18. Telenor: TMN (test model near term)—(H.263) encoder/decoder, version 2.0, source code. Telenor Research and Development, Norway (1996)
19. Tourapis, A., Au, O., Liou, M.: Predictive motion vector field adaptive search technique (PMVFAST)—enhancing block based motion estimation. In: Proceedings of SPIE—Visual Communications and Image Processing (VCIP), San Jose, CA, pp 883–892 (2001)
20. Xilinx ML310 User Guide for Virtex-II Pro Embedded Development Platform v1.1.1. Xilinx, Inc. (2004)

Author Biographies

Tiago Dias received the M.Sc. degree on Electrical and Computers Engineering from Instituto Superior Tecnico (IST), Technical University of Lisbon, Portugal, in 2004. He is currently a lecturer at the Department of Electronics, Telecomunications and Computer Engineering at Instituto Superior de Engenharia de Lisboa (ISEL) of Instituto Politécnico de Lisboa. He has been with the Signal Processing Systems Group (SiPS) of Instituto de Engenharia de Sistemas e Computadores R&D (INESC-ID), where he has been developing his research work in the area of specialized architectures for video coding.

Nuno Roma received the M.Sc. degree on Electrical and Computers Engineering from Instituto Superior Técnico (IST), Technical University of Lisbon, Portugal, in 2001. He is currently a lecturer at the Department of Information Systems and Computer Engineering at IST and a researcher of the Signal Processing Systems Group (SiPS) of Instituto de Engenharia de Sistemas e Computadores R&D (INESC-ID), where he has been pursuing his Ph.D. studies in the area of video coding and transcoding algorithms in the compressed DCT-domain. He has also maintained a long term research interest in the area of dedicated and specialized circuits for digital signal processing, with a special emphasis on image processing and video coding.

Leonel Sousa received the Ph.D. degree in electrical and computer engineering from the Instituto Superior Técnico (IST), Universidade Técnica de Lisboa, Lisbon, Portugal, in 1996. He is currently an Associate Professor of the Electrical and Computer Engineering Department at IST and a senior researcher at the Instituto de Engenharia de Sistemas e Computadores- R&D. His research interests

include VLSI architectures, computer architectures, parallel and distribuded computing, and multimedia systems. He has contributed to more than 100 papers to journals and international conferences and he is currently a member of HiPEAC European Network of Excellence. He is a senior member of IEEE and the IEEE Computer Society and a member of ACM.

Miguel Ribeiro received his M.Sc. degree in electrical and computer engineering from the Instituto Superior Técnico (IST), Universidade Técnica de Lisboa, Lisbon, Portugal, in 2007. His research interests include parallel and distribuded computing and specialized architectures for digital signal processing.