Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2007, Article ID 57234, 10 pages
doi:10.1155/2007/57234

Research Article

Adaptive Motion Estimation Processor for Autonomous Video Devices

T. Dias, S. Momcilovic, N. Roma, and L. Sousa

INESC-ID/IST/ISEL, Rua Alves Redol 9, 1000-029 Lisboa, Portugal

Received 1 June 2006; Revised 21 November 2006; Accepted 6 March 2007

Recommended by Marco Mattavelli

Motion estimation is the most demanding operation of a video encoder, corresponding to at least 80% of the overall computational cost. As a consequence, with the proliferation of autonomous and portable handheld devices that support digital video coding, data-adaptive motion estimation algorithms have been required to dynamically configure the search pattern not only to avoid unnecessary computations and memory accesses but also to save energy. This paper proposes an application-specific instruction set processor (ASIP) to implement data-adaptive motion estimation algorithms that is characterized by a specialized datapath and a minimum and optimized instruction set. Due to its low-power nature, this architecture is highly suitable to develop motion estimators for portable, mobile, and battery-supplied devices. Based on the proposed architecture and the considered adaptive algorithms, several motion estimators were synthesized both for a Virtex-II Pro XC2VP30 FPGA from Xilinx, integrated within an ML310 development platform, and using a StdCell library based on a 0.18 μm CMOS process. Experimental results show that the proposed architecture is able to estimate motion vectors in real time for QCIF and CIF video sequences with a very low-power consumption. Moreover, it is also able to adapt the operation to the available energy level in runtime. By adjusting the search pattern and setting up a more convenient operating frequency, it can change the power consumption in the interval between 1.6 mW and 15 mW.

1. INTRODUCTION

Motion estimation (ME) is one of the most important operations in video encoding to exploit temporal redundancies in sequences of images. However, it is also the most computationally costly part of a video codec. Despite that, most of the actual video coding standards apply the block matching (BM) ME technique on reference blocks and search areas of variable size [1]. Nevertheless, although the BM approach simplifies the ME operation by considering the same translation movement for the whole block, real-time ME with power consumption constraints is usually only achievable with specialized VLSI processors [2]. In fact, depending on the adopted search algorithm, up to 80% of the operations required to implement a MPEG-4 video encoder are devoted to ME, even when large search ranges are not considered [3].

The full-search block-matching (FSBM) [4] method has been, for several years, the most adopted method to develop VLSI motion estimators, due to its regularity and data independency. In the 90s, several nonoptimal but faster search block-matching algorithms were proposed, such as the three-step search (3SS) [5], the four-step search (4SS) [6], and the diamond search (DS) [7]. However, these algorithms have been mainly applied in pure software implementations, better suited to support data-dependency and irregular search patterns, which usually result in complex and inefficient hardware designs, with high power consumption.

The recent demand for the development of portable and autonomous communication and personal assistant devices imposes additional requirements and constraints to encode video in real time but with low power consumption, maintaining a high signal-to-noise ratio for a given bitrate. Recently, the FSBM method was adapted to design low-power architectures based on a ± 1 full-search engine that implements a fixed 3×3 square search window [8], by exploiting the variations of input data to dynamically configure the search-window size [9], or to guide the search pattern according to the gradient-descent direction.

Moreover, new data-adaptive efficient algorithms have also been proposed, but up until now only software implementations have been presented. These algorithms avoid unnecessary computations and memory accesses by taking

advantage of temporal and spacial correlations, in order to adapt and optimize the search patterns. These are the cases of the motion vector field adaptive search technique (MVFAST), the enhanced predictive zonal search (EPZS) [10, 11], and the fast adaptive motion estimation (FAME) [12]. In these algorithms, the correlations are exploited by carrying information about previously computed MVs and error values, in order to predict and adapt the current search space, namely, the start search location, the search pattern, and the search area size. These algorithms also comprise a limited number of different states. Such states are selected according to threshold values that are dynamically adjusted to adapt the search procedure to the video sequence characteristics.

This paper proposes a new architecture and techniques to implement efficient ME processors with low power consumption. The proposed application-specific instruction set processor (ASIP) platform was tailored to efficiently program and implement a broad class of powerful, fast and adaptive ME search algorithms, using both the traditional fixed block structure (16×16 pixels), adopted in H.261/H.263 and MPEG-1/MPEG-2 video coding standards, or even variable-block-size structures, adopted in H.264/AVC coding standards. Such flexibility was attained by developing a simple and efficient microarchitecture to support a minimum and specialized instruction set, composed by only eight different instructions specifically defined for ME. In the core of this architecture, a datapath has been specially designed around a low-power arithmetic unit that efficiently computes the sum of absolute differences (SAD) function. Furthermore, the several control signals are generated by a quite simple and hardwired control unit.

A set of software tools was also developed and made available to program ME algorithms on the proposed ASIP, namely, an assembler and a cycle-based accurate simulator. Efficient and adaptive ME algorithms, that also take into account the amount of energy available in portable devices at any given time instant, have been implemented and simulated for the proposed ASIP. The proposed architecture was described in VHDL and synthesized for a Virtex-II Pro FPGA from Xilinx. An application-specific integrated circuit (ASIC) was also designed, using a $0.18\,\mu$m CMOS process. Experimental results show that the proposed ASIP is able to encode video sequences in real time with very low power consumption.

This paper is organized as follows. In Section 2, ME algorithms are described and adaptive techniques are discussed. Section 3 presents the instruction set and the microarchitecture of the proposed ASIP. Section 4 describes the software tools that were developed to program and simulate the operation of the ASIP with cycle level accuracy, as well as other implementation aspects. Experimental results are provided in Section 5, where the efficiency of the proposed ASIP is compared with the efficiency of other motion estimators. Finally, Section 6 concludes the paper.

2. ADAPTIVE MOTION ESTIMATION

Block matching algorithms (BMA) try to find the best match for each macroblock (MB) in a reference frame, according to a search algorithm and a given distortion measure. Several search algorithms have been proposed in the last few years, most of them using the SAD distortion measure, depicted in (1), where F_{curr} and F_{prev} denote the current and previously coded frames, respectively,

$$\begin{aligned}\text{SAD}(v_x, v_y) &= \sum_{m=0}^{(N_1-1)} \sum_{n=0}^{(N_2-1)} \big| F_{\text{curr}}(x+m, y+n) \\ &\qquad - F_{\text{prev}}(x+v_x+m, y+v_y+n) \big|. \end{aligned} \tag{1}$$

The well-known FSBM algorithm examines all possible displaced candidates within the search area, providing the optimal solution at the cost of a huge amount of computations. The faster BMAs reduce the search space by guiding the search pattern according to general characteristics of the motion, as well as the computed values for distortion. These algorithms can be grouped in two main classes: (i) algorithms that treat each macroblock independently and search according to predefined patterns, assuming that distortion decreases monotonically as the search moves in the best match direction; (ii) algorithms that also exploit interblock correlation to adapt the search patterns.

The 3SS, 4SS, and DS are well-known examples of fast BMAs that use a square search pattern. Their main advantage is their simplicity, being the *a priori* known possible sequence of locations that can be used in the search procedure. The 3SS algorithm examines nine distinctive locations at 9×9, 5×5, and 3×3 pixel search windows. In 4SS, search windows have 5×5 pixels in the first three steps and 3×3 pixels in the fourth step. If the minimal distortion point corresponds to the center in any of the intermediate steps, this algorithm goes directly to the fourth and last step. On the other hand, the DS algorithm performs the search within the limits of the search area until the best matching is found in the center of the search pattern. It applies two diamond-shaped patterns: large diamond search pattern (LDSP), with 9 search points, and small diamond search pattern (SDSP) with 5 search points. The algorithm initially performs the LDSP, moving it in the direction of a minimal distortion point until it is found in the center of a large diamond. After that, the SDSP is performed as a final step.

The other class of more powerful and adaptive fast BMAs exploits interblock correlation, which can be in both the space and time dimensions. With this approach, information from adjacent MBs is potentially used to obtain a first prediction of the motion vector (MV). The MVFAST and the FAME are some examples of these algorithms.

The MVFAST is based on the DS algorithm, by adopting both the LDSP and the SDSP along the search procedure (see Figure 1). The initial central search point as well as the following search patterns are predicted using a set of adjacent MBs, namely, the left, the top, and the top-right neighbor MBs depicted in Figure 1(a). The selection between LDSP and SDSP is performed based on the characteristics of the motion in the considered neighbor MBs and on the values of two thresholds, L1 and L2. As a consequence, the algorithm performs as follows: (i) when the magnitude of the largest

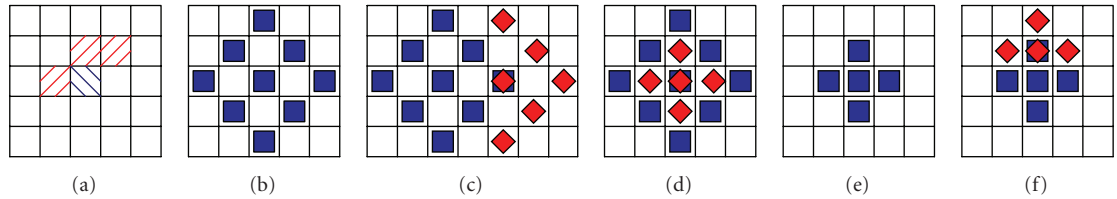

Figure 1: MVFAST algorithm: (a) neighbor MBs considered as potential predictors; (b-c) large diamond patterns; (d) switch from large to small diamond patterns; (e-f) small diamond patterns.

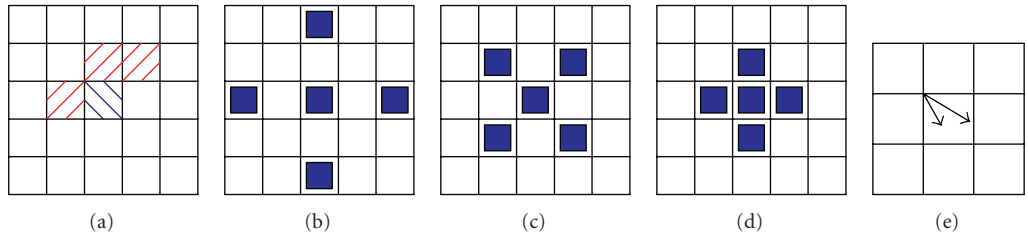

Figure 2: FAME algorithm: (a) neighbor MBs considered as potential predictors; (b) large diamond pattern; (c) elastic diamond pattern; (d) small diamond pattern; (e) considered motion vector predictions: average value and central value.

MV of the three neighbor MBs is below a given threshold L1, the algorithm adopts an SDSP, starting from the center of the search area and moving the small diamond until the minimum distortion is found in the center of the diamond; (ii) when the largest MV is between L1 and L2, the algorithm uses the same central point but applies the LDSP until the minimal distortion block is found in the center; an additional step is performed with the SDSP; (iii) when the magnitude is greater than L2, the minimum distortion point among the predictor MVs is chosen as the central point and the algorithm performs the SDSP until the minimum distortion point is found in the center.

Meanwhile, the predictive motion vector field adaptive search technique (PMVFAST) algorithm has been proposed. It incorporates a set of thresholds in the MVFAST to trade higher speedup at the cost of memory size and memory bandwidth. It computes the SAD of some highly probable MVs and stops if the minimum SAD so far satisfies the stopping criterion, performing a local search using some of the techniques of MVFAST.

More recently, the FAME [12] algorithm was proposed, claiming very accurate MVs that lead to a quality level very close to the FSBM but with a significant speedup. The FAME algorithm outperforms MVFAST by taking advantage of the correlation between MVs in both the spatial (see Figures 2(b)–2(d)) and the temporal (see Figure 2(e)) domains, using adaptive thresholds and adaptive diamond-shape search patterns to accelerate ME. To accomplish such objective, it features an improved control to confine the search pattern and avoid stationary regions.

When compared in terms of computational complexity, all these algorithms are widely regarded as good candidates for software implementations, due to their inherent irregular processing nature. It is proved in this paper that by adopting the proposed ASIP approach, it is possible to develop hardware processors to efficiently implement not only any adaptive ME algorithm of this class, but also any other fast BMA. In fact, the FSBM, 3SS, 4SS, DS, MVFAST, and FAME algorithms have been implemented with the proposed ASIP, in order to evaluate the performance of the processor.

3. ASIP INSTRUCTION SET AND MICROARCHITECTURE

3.1. *Instruction set*

The instruction set architecture (ISA) of the proposed ASIP was designed to meet the requirements of most ME algorithms, including adaptive ones, but optimized for portable and mobile platforms, where power consumption and implementation area are mandatory constraints. Consequently, such ISA is based on a register-register architecture and provides only a reduced number of different operations (eight) that focus on the most widely executed instructions in ME algorithms. This register-register approach was adopted due to its simplicity and efficiency, allowing the design of simpler and less hardware consuming circuits. On the other hand, it offers increased efficiency due to its large number of general purpose registers (GPRs), which provides a reduction of the memory traffic and consequently a decrease in the program execution time. The amount of registers that compose the register file therefore results as a tradeoff between the implementation area, memory traffic, and the size of the program memory. For the proposed ASIP, the register file consists of

Table 1: Instruction-set architecture of the proposed ASIP. Categories of instruction operators and operation codes.

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 Rs —

24 GPRs and eight special purpose registers (SPRs) capable of storing one 16 bits word each. Such configuration optimizes the ASIP efficiency since: (i) the amount of GPRs is enough to allow the development of efficient, yet simple, programs for most ME algorithms; (ii) the 16 bits data type covers all the possible values assigned to variables in ME algorithms; and (iii) the eight 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 can be seen from Table 1, and were obtained as the result of the analysis of the execution of several different ME algorithms [10, 11, 13]. The encoding of the instructions into binary representation was performed using 16 bits and a fixed format. For each instruction it is specified an opcode and up to three operands, depending on the instruction category. Such encoding scheme therefore provides minimum bit wasting for instruction encoding and eases the decoding, thus allowing a good tradeoff between the program size and the efficiency of the architecture. In the following, it is presented a brief description of all the operations of the proposed ISA.

3.1.1. 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. The instruction has a 2 bits condition 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 negative, positive or zero. Not only is this instruction important for algorithmic purposes, but also for improving code density, since it allows a minimization of the number of instructions required to implement an ME algorithm and therefore a reduction of the required capacity of the program memory.

3.1.2. 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 can 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 bits width, but a control field (t) sets the loading of the 8 bits literal into the destination register upper or lower byte.

3.1.3. Arithmetic operations

In what concerns the arithmetic operations, while 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 in loops, 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 its outcome that can be used by the jump (J) instruction, to conditionally change the control flow of a program.

3.1.4. Graphics operation

The SAD16 operation allows the computation of the SAD similarity measure between an MB and a candidate block. To do so, this operation computes the SAD value considering two sets of sixteen pixels (the minimum amount of pixels for an MB in the MPEG-4 video coding standard) and accumulates the result to the contents of a GPRs. The computation of a SAD value for a given (16×16)-pixel 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 arithmetic operations, the outcome of this operation also provides some extra information that can be used by the jump (J) instruction to conditionally change the control flow of a program.

3.1.5. Memory data transfer operation

The processor comprises two small and fast 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 pixel data into these memories. Such

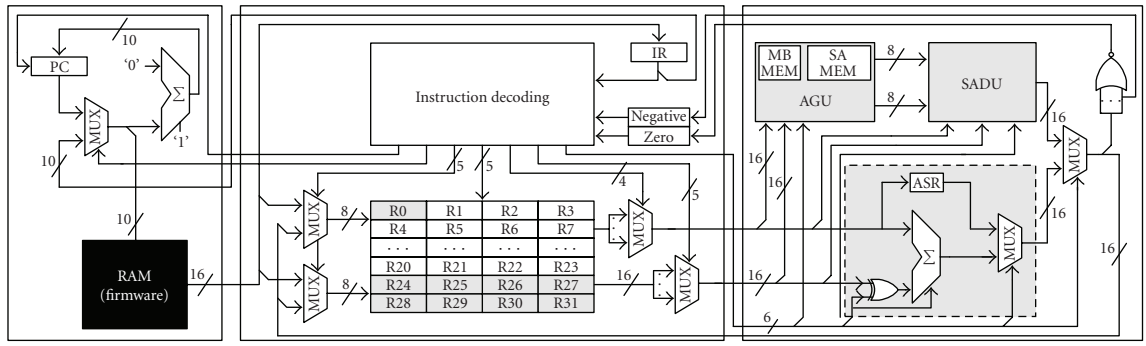

Figure 3: Architecture of the proposed ASIP.

operation is carried out by means of an address generation unit (AGU), which generates the set of addresses of both the corresponding internal memory as well as of the external frame memory, that are required to transfer the pixel data. The selection of the target memory is carried out by means of a 1-bit control field, which is used to specify the type of image area that is loaded into the local memory. As a consequence, this operation is performed independently for the data concerning a given MB and for the corresponding search area.

3.2. Micro architecture

The proposed ISA is supported by a specially designed micro-architecture, following strict power and area driven policies to support its implementation in portable and mobile platforms. This micro-architecture presents a modular structure and is composed by simple and efficient units to optimize the data processing, as it can be seen from Figure 3.

3.2.1. 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 processors functional units, such as clock gating and operating frequency adjustment. The former technique is applied to control the switching activity at the function unit level, by inhibiting input updates to functional units whose outputs are not required for a given operation, while the latter adjusts the operating frequency according to the programmed algorithm and the current available energy level.

3.2.2. Datapath

For more complex and specific operations, like the LD and SAD16 instructions, the datapath also includes specialized units to improve the efficiency of such operations: the AGU and the SAD unit (SADU), respectively.

The LD operation is executed by a dedicated AGU optimized for ME, which is capable of fetching all the pixels for both an MB and 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. Using such feature, programs can be optimized by rescheduling the LD instructions to allow data fetching from memory to occur simultaneously with the execution of other parts of the program that do not depend on this data. For implementations imposing strict constraints in the power consumption, memory accesses can be further optimized by using efficient data reuse algorithms and extra hardware structures [4, 14]. This not only significantly reduces the memory traffic to the external memory, but also provides a considerable reduction in the power consumption of the video encoding system.

The SADU can execute the SAD16 operation in up to sixteen clock cycles and is capable of using the arithmetic and logic unit (ALU) to update the coordinates of the candidate block line of pixels. The number of clock cycles required for the computation of a SAD value is imposed by the type of architecture adopted to implement this unit, which depends on the power consumption and implementation area constraints specified at design time. Thus, applications imposing more severe constraints in power or area can use a serial processing architecture, that reuses hardware but takes more clock cycles to compute the SAD value, while others without so strict requisites may use a parallel processing architecture that is able to compute the SAD value in only one single clock cycle. Pipelined versions of the SADU are also supported to allow better tradeoffs between the latency, the power consumption, and the required implementation area, thus providing increased flexibility for different implementations of the proposed ASIP.

Despite all these different alternatives in what concerns the SADU architecture to meet the desired performance level, the implemented SADU also adopted an innovative and efficient arithmetic unit to compute the minimum SAD distance [15] that allows the proposed processor to better comply with the low-power constraints usually found in autonomous and portable handheld devices. Such unit not only avoids the usage of carry-propagate cells to compute and compare the SAD metric, by adopting carry-free arithmetic,

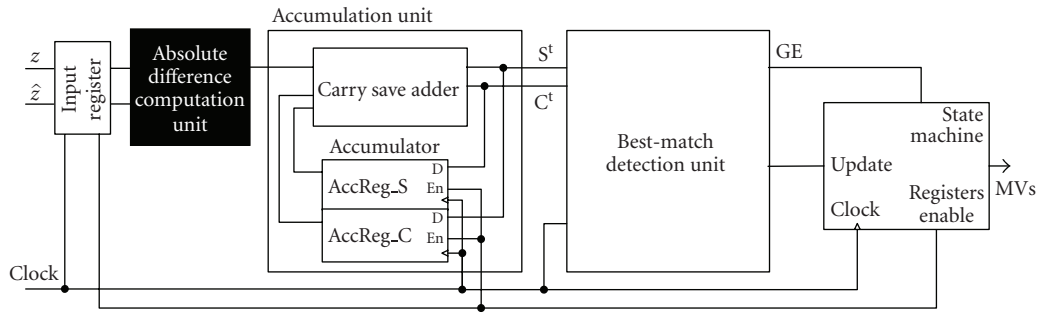

Figure 4: Low-power serial SADU block diagram.

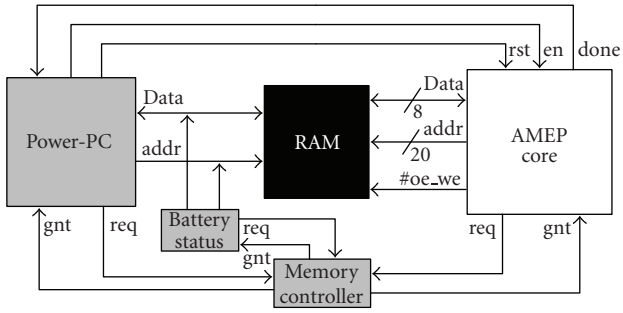

Figure 5: Interface of the proposed ASIP.

but it also generates a "greater or equal" (GE) signal, issued by the best-match detection unit (see Figure 4). This signal is obtained from the partial values of the SAD measure, by comparing the current metric value with the best one previously obtained. It can be used by the main state machine to update the output register corresponding to the current MV. Due to its null latency, this GE signal can also be used to apply the set of power-saving techniques that have been proposed in the last few years [16]. In fact, it is used as a control mechanism to avoid needless calculations in the computation of the best match for a macroblock, by aborting the ME procedure as soon as the partial value of the distance metric for the candidate block under processing exceeds the one already computed for the current block [16]. Such computations can be avoided by disabling all the logic and arithmetic units used in the computation of the SAD metric, thus providing significant power saving ratios. On average, this technique allows to avoid up to 50% of the required computations [16], giving rise to a reduction of up to 75% of the overall power consumption [15].

3.2.3. External interface

The proposed ASIP presents an external interface with a quite reduced pin count, as shown in Figure 5, that allows an easy embedding of the presented micro-architecture in both existing and future video encoders. Such interface was designed not only to allow efficient data transfers from the external frame memory, but also to efficiently export the coordinates of the best matching MVs to the video encoder. In addition, it also provides the possibility to download the processor's firmware, that is, the compiled assembly code of the desired ME algorithm.

Since pixels for ME are usually represented using 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), the interface with the external frame memory was designed to allow 8 bits data transfers from a 1 MB memory address space. Thus, the proposed interface with such external memory bank is done using three I/O ports: (i) a 20 bits output port that specifies the memory address for the data transfers (*addr*); (ii) an 8 bits bidirectional port for transferring the data (*data*); and (iii) a 1-bit output port that sets whether it is a load or store operation (*#oe_we*). Since the external frame memory is to be shared between the video encoder and the ME circuit, the proposed ASIP interface has two extra 1-bit control ports to implement the required handshake protocol with the bus master: the *req* port allows requesting control of the bus to the bus master, while the *gnt* port allows the bus master to grant such control.

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 low-order 8 bits of the MV coordinate and a second one to output its high-order 8 bits. 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.

This port is also used to dynamically aquire the energy level that is available to compute the motion estimation at any instant (see Figure 5). Such level may be used by adaptive algorithms to adjust the overall computational cost of the ME procedure.

The processor's firmware, corresponding to the compiled assembly code of the considered ME algorithm, is also downloaded into the program RAM through the *data* port. To do so, the 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, after having acquired the bus ownership, the master processor supplies memory addresses through the *addr* port and loads the corresponding instructions into the internal program RAM. The

```
...
0042h 81efh  SAD16 R1, R14, R15
0043h 81efh  SAD16 R1, R14, R15
0044h 81efh  SAD16 R1, R14, R15
0045h 81efh  SAD16 R1, R14, R15
0046h eb21h  SUB R11, R2, R1
0047h 684bh  J.N NEXT_POS
0048h 2041h  MOVR R2, R1
0049h 20c5h  MOVR R6, R5
004ah 20e4h  MOVR R7, R4
004bh        NEXT_POS:
004bh c448h  ADD R4, R4, R8
004ch eb94h  SUB R11, R9, R4
004dh 6850h  J.Z NEWLINE
004eh c338h  ADD R3, R3, R8
004fh 680eh  J.U LOOP
0050h        NEWLINE:
0050h 2091h  MOVR R4, R17
0051h e404h  SUB R4, R0, R4
0052h c558h  ADD R5, R5, R8
0053h eb95h  SUB R11, R9, R5
0054h 6858h  J.Z END
0055h 2170h  MOVR R11, R16
0056h c33bh  ADD R3, R3, R11
0057h 680eh  J.U LOOP
0058h        END:
 ...
```

Figure 6: Fraction of one of the output files obtained with the assembly compiler.

processor exits this programming mode as soon as the last memory position of the 1 kB program memory is filled in. Once again, each 16 bits instruction takes two clock cycles to be loaded into the program memory, which is organized in the little-endian format.

4. SOFTWARE TOOLS

To support the development and implementation of ME algorithms using the proposed ASIP, a set of software tools was developed and made available, namely, an assembly compiler and a cycle-based accurate simulator.

Since the proposed ASIP architecture and the considered instruction set do not support subroutine calls nor make use of an instruction/data stack, the implementation of the compiler consists of a straightforward parsing of the assembly instruction directives (as well as their register operands), followed by a corresponding translation into the appropriate opcodes, in order to translate the sequence of assembly instructions into a series of 16 bits machine code words of program data. The exception to this direct translation occurs whenever a *jump* instruction has to be compiled. A two-step strategy was adopted to compile these control flow instructions, in order to determine the target address of each *jump* invoked within the program.

In Figure 6 it is presented a fraction of one of the output files (code.lst) that are generated during this translation process. This file presents three different sorts of information, disposed in three columns (see Figure 6). While the first column presents the effective address of each instruction (or label), the second column presents the instruction code of the assembly directive presented in the third column. In the illustrated case, it is presented a fraction of an implementation of the FSBM algorithm (used as reference in the considered algorithm comparisons). As it can be seen in Figure 6, the resulting SAD value, accumulated in R1 register after a sequence of 16 SAD16 instructions (one for each row of the macroblock), is compared with the best SAD value (stored in R2) that was found in previous computations. Depending on the difference between these values, the current SAD value, as well as the corresponding MV coordinates (R5, R4), will be stored in R2, R6, and R7 registers, in order to be considered in the next searching locations. In the remaining instructions, the MV coordinates are incremented and it is checked if the last column and line of the considered search area were already reached, respectively.

The implementation and evaluation of the ME algorithms were supported by implementing an accurate simulator of the proposed ASIP. It provides important information about: the number of clock cycles required to carry out the ME of a given macroblock, the amount of memory space required to store the program code, the obtained motion vector and corresponding SAD value, and so forth.

5. IMPLEMENTATION AND EXPERIMENTAL RESULTS

To assess the performance provided by the proposed ASIP, the microarchitecture core was implemented by using the described serial processing architecture for the SADU module (see Figure 4) and a simplified AGU that does not allow data reusage. This microarchitecture was described using both behavioral and fully structural parameterizable IEEE-VHDL. The ASIP was firstly implemented in a FPGA device, in order to proof the concept. Later, an ASIC was specifically designed in order to evaluate the efficiency of the proposed architecture and of the corresponding ISA for motion estimation.

The performance of the proposed ASIP was evaluated by implementing several ME algorithms, such as the FSBM, the 4SS, the DS, and the MVFAST and FAME adaptive ME algorithms. These algorithms were programmed with the proposed instruction set and the ASIP operation was simulated by using the developed software tools (see Section 4). Such simulation phase was fundamental to obtain the number of clock cycles required to implement the algorithms, which implicitly defines the minimum clock frequency for real-time processing, as well as the size of the memory required to store the programs.

Table 2 provides the average number of clock cycles per pixel (CPP) required to implement the several considered algorithms, using the following benchmark video sequences: *mobile*, *carphone*, *foreman*, *table tennis*, *bus,* and *bream*. These are well-known video sequences with quite different characteristics, in terms of both movement and spacial detail. The presented results were obtained for a search area with 16×16 candidate locations and for the first 20 frames of each video

Table 2: Required clock cycles to process each pixel considering several different algorithms and video sequences.

Video seq.	FSBM	4SS	DS	MVFAST	FAME
Mobile	265	19	15	9	8
Carphone	265	21	18	13	9
Foreman	265	21	18	13	9
Table tennis	265	19	15	8	6
Bream	265	19	15	8	8
Bus	265	24	21	18	8
Maximum	265	24	21	18	9

Table 3: Required operating frequencies to process QCIF and CIF video sequences in real time.

Format	FSBM	4SS	DS	MVFAST	FAME
QCIF	200 MHz	20 MHz	18 MHz	15 MHz	8 MHz
CIF	800 MHz	75 MHz	65 MHz	55 MHz	28 MHz

Table 4: Code size of the proposed algorithms (words of 16 bits).

Algorithm	FSBM	4SS	DS	MVFAST	FAME
Code size	56	365	460	744	917

sequence. Moreover, redundancy was eliminated in both the 4SS and the MVFAST algorithms, by avoiding the computation of SAD more than once for a single location.

The results presented in Table 2 evidence the huge reduction of the number of performed computations that can be achieved when fast search algorithms are applied. The MVFAST and FAME adaptive algorithms allow to significantly reduce the CPP even further, when compared with the 4SS and the DS fast algorithms. By considering the maximum value for the obtained CPPs (CPP_M) and a real-time frame rate of 30 Hz for an $H \times W$ image format, the required minimum operating frequency (ϕ) can be calculated for each class of algorithms using (2),

$$\phi = (H \times W) \times \text{CPP}_M \times 30\,\text{Hz}. \tag{2}$$

By considering the quarter common intermediate format (QCIF) and the common intermediate format (CIF) image formats, as well as the values presented in Table 2 and (2), the required minimum clock frequencies were computed and are presented in Table 3. The obtained operating frequencies of the proposed motion estimators for fast adaptive search algorithms are significantly lower than the operating frequency of the ± 1 full-search-based processor presented in [8].

In Table 4, it is represented the size of the memory required to store the programs corresponding to the considered algorithms. As it can be seen, the adaptive algorithms require significantly more memory for storing the program than the 4SS. The memory requirements of the FAME algorithm are even greater than the MVFAST, due to the need to keep in memory more past information to achieve significantly better predictions. In fact, it requires approximately 13 times more memory than the FSBM. This is the price to pay for the irregularity and also for the adaptability of the MVFAST and FAME algorithms (744×16 bit). However, since most of the portable communication systems already provide nonvolatile memories with significant capacity, the power consumption gain due to the reduction of the operating frequency can supersede this disadvantage.

Table 5: Experimental results of the implementation of the video encoder in the Xilinx ML310 development board.

Unit	Slices		LUTs		BRAMs	F (MHz)
AMEP core	2052	14%	2289	8%	208	67.18
Interface	207	1%	382	1%	0	156.20

5.1. FPGA implementation for proof of concept

To validate the functionality of the proposed ASIP in a practical realization, a hybrid video encoder was developed and implemented in a Xilinx ML310 development platform, making use of a Virtex-II Pro XC2VP30 FPGA device from Xilinx embedded in the board [17]. Besides all the implementation capabilities offered by such configurable device, this platform also provides two Power-PC processors, several block RAMs (BRAMs), and high speed on-chip bus-communication links to enable the interconnection of the Power-PC processors with the developed hardware circuits.

The prototyping video encoding system was implemented by using these resources. It consists of the developed ASIP motion estimator, a software implementation of an H.263 video encoder, built into the FPGA BRAMs and running on a 100–300 MHz Power-PC 405D5 processor, and of four BRAMs to implement the firmware RAM and the local memory banks in the AGU of the proposed ASIP. Furthermore, the Power-PC processor and the developed motion estimator were interconnected according to the interface scheme described in Figure 5, using both the high-speed 64 bits processor local bus (PLB) and the general purpose 32 bits on-chip peripheral bus (OPB), where the Power-PC was connected as the master device. Such interconnect buses are used not only to exchange the control signals between the Power-PC and the proposed ASIP, but also to send all the required data to the proposed motion estimator, namely, the ME algorithm program code and the pixels for both the candidate and reference blocks. Moreover, a simple handshake protocol is used in these data transfers to bridge the different operating frequencies of the two processors.

The operating principle of the proposed prototyping hybrid video encoder consists only of three different tasks related to motion estimation: (i) configuration of the ME coprocessor, by downloading an ME algorithm and all the configuration parameters (MB size, search area size, image width, and image height) into the code memory and the SPRs of the proposed ASIP; (ii) data transfers from the Power-PC to the proposed ASIP, which occur on demand by the motion estimator and are used either to download the MB and the search area pixels into the AGU local memories or to supply additional information required by adaptive ME algorithms, depending on the memory position addressed by

Table 6: Experimental results of the synthesized ASIP components for the maximum frequencies and 0.18 μm CMOS technology.

Unit	Area (μm^2)	Max. freq.	Power at max. freq.
AMEP core	128625	144 MHz	48 mW
AGU	28889	154 MHz	49 mW
ALU	3496	481 MHz	13 mW
SADU	16961	275 MHz	22 mW
Best-match DU	9489	500 MHz	14 mW

Table 7: Experimental results of the synthesized ASIP components operating at 100 MHz and 0.18 μm CMOS technology.

Unit	AGU	ALU	SADU	BMDU	AMEP core
Power (mW)	8.29	1.92	3.40	1.26	19.96

Table 8: Estimated power consumption of the ASIP for different frequencies and 0.18 μm CMOS technology.

Freq. (MHz)	8	15	18	20	28	55	65	75	100
Power (mW)	1.6	3	3.5	4	5.5	11	13	15	20

the ASIP; and (iii) data transfers from the proposed ASIP to the Power-PC, that are used to output the coordinates of the best-match MV and the corresponding SAD value, as well as the current configuration parameters of the motion estimator, since some adaptive ME algorithms change these values during the video coding procedure.

Table 5 presents the experimental results that were obtained with the implementation of the proposed video coding system in the Virtex-II Pro XC2VP30 FPGA device. Such results show that by using the proposed ASIP, it is possible to estimate MVs in real time (30 fps) for the QCIF and CIF image formats by using any fast or adaptive search algorithms, except the 4SS for CIF images (see Table 3). Moreover, the minimum throughput achieved for the considered algorithms (4SS) is about 2.8 Mpixels/s, corresponding to a relative throughput per slice of about 1.36 kpixels/s/slice.

The operating frequency of the ASIP can be changed in the FPGA by using the digital clock managers (DCMs). In this case, the DCMs were used to configure setup pairs of algorithms/formats-frequencies depicted in Table 3. However, in an ASIC implementation, an additional input is required in the ASIP in order to sense, at any time, the amount of energy that is still available; and an extra programmable divisor to adjust the clock frequency. The control of this dynamic adjustment can be done by the ASIP and the programming of the divisor can be done through an extra output register.

5.2. Standard-cell implementation

The proposed motion estimator was implemented using the *Synopsis* synthesis tools and a high-performance StdCell library based on a 0.18 μm CMOS process from *UMC* [18]. The obtained experimental results concern an operating environment imposing typical operating conditions: $T = 25^\circ$C, $V_{dd} = 1.8$ V, the "suggested_20 k" wire load model, and some

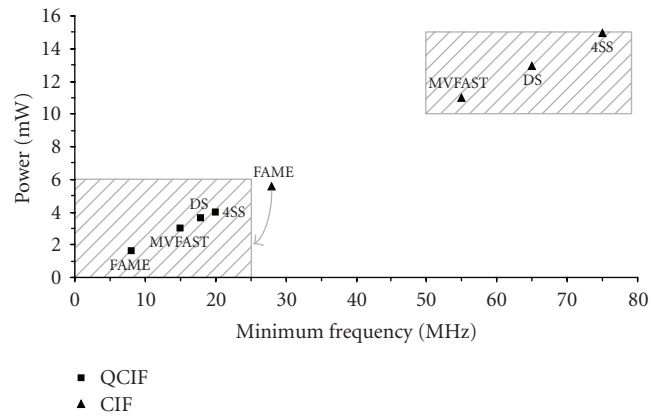

Figure 7: Power consumption corresponding to each of the considered algorithms and image formats.

constraints that lead to an implementation with minimum area. Typical case conditions have been considered for power estimation, and prelayout netlist power dissipation results are presented.

The first main conclusion that can be drawn from the synthesis results presented in Tables 6, 7, and 8 is that the power consumption of the proposed ASIP for ME with the adaptive ME algorithms is very low. Operating at a frequency of 8 MHz, it only consumes about 1.6 mW, which does not imply any significant reduction of the life time of our actual batteries (typically 1500 mAh batteries). For the 4SS algorithm, the operating frequency increases to about 20 MHz but the power consumption is kept low, about 3.9 mW. The setup corresponding to the FSBM algorithm for the CIF image format was not fully synthesized, since the required operating frequency is beyond the technology capabilities. The maximum operating frequency obtained with this architecture and with this technology is about 144 MHz, as it can be seen in Table 6. Near this maximum frequency, which corresponds to having the components of the processor operating at 100 MHz, the power consumption becomes approximately 20 mW (see Table 7).

Tables 7 and 8 present the power consumption values estimated for the required minimum operating frequencies. Two main clusters of points can be identified in the plot of Figure 7: the one for the QCIF and the one for the CIF format. The former format requires operating frequencies below 25 MHz and the corresponding power consumption is below 6 mW, while for the CIF format the operating frequency is above 50 MHz and the power consumption is between 10 mW and 15 mW. The exception is the FAME algorithm, for which the operating frequency (28 MHz) and the power consumption (5.5 mW) values for the CIF format are closer to the QCIF values.

Common figures of merit for evaluating the energy and the area efficiencies of the video encoders are the number of Mpixels/s/W and the number of Mpixels/s/mm^2. For the designed VHDL motion estimator, the efficiency figures are, on average, 23.7 Mpixels/s/mm^2 and 544 Mpixels/s/W. These

values can be compared with the ones that were presented for the motion estimator ASIP proposed in [19], after normalizing the power consumption values to a common voltage level: 22 Mpixels/s/mm^2 and 323 Mpixels/s/W. Hence, it can be concluded that the proposed motion estimator is more efficient in terms of both power consumption and implementation area. In fact, the improvements should be even greater, since the proposed circuit was designed with a 0.18 μm CMOS technology, while the circuit in [19] was designed with a 0.13 μm CMOS technology.

6. CONCLUSIONS

An innovative design flow to implement efficient motion estimators was presented here. Such approach is based on an ASIP platform, characterized by a specialized datapath and a minimum and optimized instruction set, that was specially developed to allow an efficient implementation of data-adaptive ME algorithms. Moreover, it was also presented a set of software tools that were developed and made available, namely, an assembler compiler and a cycle-based accurate simulator, to support the implementation of ME algorithms using the proposed ASIP.

The performance of the proposed ASIP was evaluated by implementing a hybrid video encoder using regular (FSBM), irregular (4SS and DS), and adaptive (MVFAST and FAME) ME algorithms using the developed software tools and a Xilinx ML310 prototyping environment, that includes a Virtex-II Pro XC2VP30 FPGA. In a later stage, the performance of the developed microarchitecture was also assessed by synthesizing it for an ASIC using a high-performance StdCell library based on a 0.18 μm CMOS process.

The presented experimental results proved that the proposed ASIP is capable of estimating MVs in real time for the QCIF image format for all the tested fast ME algorithms, running at relatively low operating frequencies. Furthermore, the results also showed that the power consumption of the proposed architecture is very low: near 1.6 mW for the adaptive FAME algorithm and around 4 mW for the remaining irregular algorithms that were considered. Consequently, it can be concluded that the low-power nature of the proposed architecture and its high performance make it highly suitable for implementations in portable, mobile, and battery-supplied devices.

REFERENCES

[1] F. C. N. Pereira and T. Ebrahimi, *The MPEG4 Book*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 2002.

[2] V. Bhaskaran and K. Konstantinides, *Image and Video Compression Standards: Algorithms and Architectures*, Kluwer Academic Publishers, Boston, Mass, USA, 2nd edition, 1997.

[3] P. Pirsch, N. Demassieux, and W. Gehrke, "VLSI architectures for video compression—a survey," *Proceedings of the IEEE*, vol. 83, no. 2, pp. 220–246, 1995.

[4] T. Dias, N. Roma, and L. Sousa, "Efficient motion vector refinement architecture for sub-pixel motion estimation systems," in *Proceedings of IEEE Workshop on Signal Processing Systems Design and Implementation (SIPS '05)*, pp. 313–318, Athens, Greece, November 2005.

[5] R. Li, B. Zeng, and M. L. Liou, "A new three-step search algorithm for block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 4, no. 4, pp. 438–442, 1994.

[6] L.-M. Po and W.-C. Ma, "A novel four-step search algorithm for fast block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 3, pp. 313–317, 1996.

[7] S. Zhu and K.-K. Ma, "A new diamond search algorithm for fast block-matching motion estimation," *IEEE Transactions on Image Processing*, vol. 9, no. 2, pp. 287–290, 2000.

[8] S.-Y. Huang and W.-C. Tsai, "A simple and efficient block motion estimation algorithm based on full-search array architecture," *Signal Processing: Image Communication*, vol. 19, no. 10, pp. 975–992, 2004.

[9] S. Saponara and L. Fanucci, "Data-adaptive motion estimation algorithm and VLSI architecture design for low-power video systems," *IEE Proceedings Computers & Digital Techniques*, vol. 151, no. 1, pp. 51–59, 2004.

[10] A. M. Tourapis, O. C. Au, and M. L. Liou, "Predictive motion vector field adaptive search technique (PMVFAST): enhancing block-based motion estimation," in *Proceedings of Visual Communications and Image Processing (VCIP '01)*, vol. 4310 of *Proceedings of SPIE*, pp. 883–892, San Jose, Calif, USA, January 2001.

[11] A. M. Tourapis, "Enhanced predictive zonal search for single and multiple frame motion estimation," in *Proceedings of Viual Communications and Image Processing (VCIP '02)*, vol. 4671 of *Proceedings of SPIE*, pp. 1069–1079, San Jose, Calif, USA, January 2002.

[12] I. Ahmad, W. Zheng, J. Luo, and M. Liou, "A fast adaptive motion estimation algorithm," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 3, pp. 420–438, 2006.

[13] S. Momcilovic, T. Dias, N. Roma, and L. Sousa, "Application specific instruction set processor for adaptive video motion estimation," in *Proceedings of the 9th Euromicro Conference on Digital System Design: Architectures, Methods and Tools (DSD '06)*, pp. 160–167, Dubrovnik, Croatia, August-September 2006.

[14] J.-C. Tuan, T.-S. Chang, and C.-W. Jen, "On the data reuse and memory bandwidth analysis for full-search block-matching VLSI architecture," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 1, pp. 61–72, 2002.

[15] T. Dias, N. Roma, and L. Sousa, "Low power distance measurement unit for real-time hardware motion estimators," in *Proceedings of International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS '06)*, pp. 247–255, Montpellier, France, September 2006.

[16] L. Sousa and N. Roma, "Low-power array architectures for motion estimation," in *Proceedings of IEEE International Workshop on Multimedia Signal Processing (MMSP '99)*, pp. 679–684, Copenhagen, Denmark, September 1999.

[17] Xilinx Inc., "User Guide. v1.1.1.," *ML310 User Guide for Virtex-II Pro Embedded Development Platform*, October 2004.

[18] Virtual Silicon Technology Inc., "eSi-Route/11^{TM} high performance standard cell library (UMC 0.18μm)," Tech. Rep. v2.4., November 2001.

[19] A. Berić, R. Sethuraman, H. Peters, J. van Meerbergen, G. de Haan, and C. A. Pinto, "A 27 mW 1.1 mm^2 motion estimator for picture-rate up-converter," in *Proceedings of the 17th International Conference on VLSI Design (VLSI '04)*, vol. 17, pp. 1083–1088, Mumbai, India, January 2004.