Application Specific Instruction Set Processor for Adaptive Video Motion Estimation

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

INESC-ID, R. Alves Redol, 9 1000-029 Lisboa – PORTUGAL

Email: {sveta,tdias}@sips.inesc-id.pt, {Nuno.Roma, Leonel.Sousa}@inesc-id.pt

Abstract

Motion estimation is the most demanding operation of a video encoder, corresponding to at least 80% of the overall computational cost. With the proliferation of 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 data-path and minimum and optimized instruction set. Due to its low-power nature, this architecture is specially adequate to develop motion estimators for portable, mobile and battery supplied devices. A cycle-based accurate simulator was also developed for the proposed ASIP and fast and data-adaptive search algorithms have been implemented, namely, the four-step search and the motion vector field adaptive search algorithms. Based on the proposed ASIP and the considered adaptive algorithms, several motion estimators were synthesized in $0.13\mu m$ CMOS technology. Experimental results show that very-low power adaptive motion estimators have been achieved to encode QCIF video sequences.

1. Introduction

Motion Estimation (ME) is widely recognized as a fundamental operation in video encoding, to exploit temporal correlation in sequences of images. However, it is also the most computationally costly part of a video codec. In fact, if the optimal Full-Search Block-Matching (FSBM) method is considered, more than 80% of the operations required to implement a MPEG-4 video encoder are devoted to ME, even if large search ranges are not considered. Although FSBM [6] has been, for several years, the main adopted method to develop VLSI motion estimators, due to its regularity and data independency, in the 90s several non-optimal but faster search block-matching algorithms have been proposed. Some examples of those algorithms are the Three-Step-Search (3SS) [4], the Four-Step-Search (4SS) [5] and the Diamond Search (DS) [11]. Up until now, all these algorithms have been mainly applied in pure software implementations, since they are better suited to support data-dependency and irregular search patterns, which usually results in complex and inefficient hardware designs.

The recent trend to make communication and personal assistant devices portable and autonomous imposes additional requirements and constraints to encode video in real time but with low power consumption, achieving, at the same time, a high signal-to-noise ratio for a given bit-rate. Following the lines of the two previously stated and distinct approaches for ME, recent research has been performed to adapt the FSBM method in order to develop low-power hardware motion estimators. At the same time, new data-adaptive fast algorithms have also been proposed for software ME. On the hardware side, some of the proposed architectures exploit the input data variations to dynamically configure the search-window size of the FSBM [7] algorithm, while other approaches allow to guide the search pattern according to the gradient-descent direction, based on a ± 1 full-search that implements a fixed 3×3 square search window [2]. For software implementations, efficient search algorithms have been developed, by taking advantage of temporal and spacial correlations of the motion vectors in order to adapt and optimize the search pattern, thus avoiding unnecessary computations and memory accesses. Examples of these recent fast search techniques are the Motion Vector Field Adaptive Search Technique (MVFAST) and the Enhanced Predictive Zonal Search (EPZS) [9, 8]. In these algorithms, the correlations are exploited by carrying information about motion vectors and previously computed error values, in order to predict and adapt the actual search space, namely, the initial search location, the search pattern and the search area dimension. Moreover, these algorithms comprise a limited number of different states that are selected according to threshold values that are dinamicaly adjusted to adapt the search procedure to the video sequence characteristics.

In this paper, an entirely new approach to implement efficient motion estimators is proposed. This approach is based on a new Application Specific Instruction Set Processor (ASIP) platform that was tailored to efficiently program and implement powerful and adaptive ME search algorithms. A minimum and specialized instruction set, specific for ME and composed by only eight different instructions, was entirely defined. To support this instruction set, a simple and efficient microarchitecture was also designed and implemented. In the core of this simple data-path it was incorporated a specialized arithmetic unit to efficiently compute the Sum of Absolute Differences (SAD) function. Furthermore, control signals are generated by a quite simple and hardwired control unit. A set of software tools was 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 instant, have been implemented and simulated for the proposed ASIP. The proposed architecture was described in VHDL and synthetised for the UMC $0.13\mu m$ standard cell library. 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 motion estimation algorithms are described and adaptive techniques are discussed. Section 3 presents the instruction set and the microarchitecture of the proposed ASIP, as well as the software tools that were developed to program and simulate, with accuracy at cycle level, the operation of the ASIP. Finally, experimental results are provided in section 4 and section 5 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 and the SAD has been the most used matching distortion measure (see eq. 1).

$$SAD(v_x, v_y) = \sum_{m,n=0}^{N-1} |F_{curr}(x+m, y+n) - F_{prev}(x+v_x+m, y+v_y+n)|, \quad (1)$$

where F_{curr} represents the current frame and F_{prev} denotes the previously coded frame.

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. Although this algorithm is not often used in practice, it is commonly adopted as a reference for evaluating other faster search algorithms, which are alternatively solutions to overcome the computational burden and related power consumptions of FSBM algorithm. These fast BMAs, which reduce the search space by guiding the search pattern according to general characteristics of the motion as well as the computed values for distortion, can be grouped in two main classes: *i*) algorithms that treat each macroblock independently and search according to pre-defined patterns; *ii*) algorithms that also exploit inter-block correlation to adapt the search patterns.

Fast BMAs use a fixed set of search patterns, assuming that distortion decreases monotonically as the search moves in the best match direction. Therefore, the search procedure always starts from a fixed and pre-defined location (usually the origin) and evolves in the direction of the minimum computed SAD, according to pre-defined search patterns. Algorithms usually finish when the SAD value does not decrease when new search locations are considered.

Examples of well known fast BMAs that use a square search pattern are the 4SS and the 3SS. The 3SS algorithm pattern examines nine characteristic locations (center, 4 angular and 4 between them) at 9x9, 5x5 and 3x3 pixels search windows. In 4SS, search windows have 5x5 pixels in the first three steps, and 3x3 pixels in the fourth step. In the 4SS algorithm, if the minimal distortion point corresponds to the center in any of the intermediate steps, the algorithm goes directly to the fourth and last step. The main advantage of these algorithms is their simplicity, being an *a priori* known the possible sequence of locations that can be used in the search procedure.

The DS algorithm is accepted in the MPEG4 Verification Model. It performs search within the limits of the search area until the best matching is found in the center of the search pattern, giving best results for large motion and with less regularity. Additionally, DS proposes 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 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 main idea is to take advantage of the center-based Motion Vector (MV) distribution, typically found in real-world frame sequences. In these algorithms, the inherent redundancy on computing more than once the same location can be avoided, by not considering already checked locations, but the corresponding processing becomes a little more irregular.

The other class of more powerful adaptive fast BMAs exploits inter-block correlation, which can be both in space

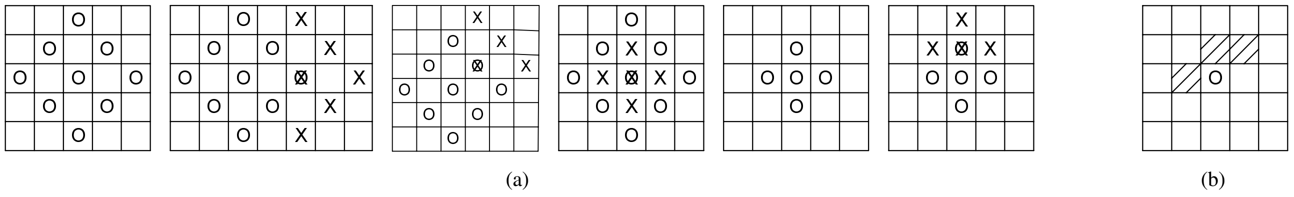
Figure 1. (a) Examples of the application of both the large and small diamond search patterns adopted by the MVFAST algorithm; (b) Neighbor MBs considered as potential predictors.

and time dimensions. With this approach, information from adjacent blocks is potentially used to obtain a first prediction of the MV.

One of the most robust adaptive algorithms is MVFAST. This algorithm is based on the diamond search algorithm, adopting both the LDSP and the SDSP along the search procedure (see fig. 1(a)). According to this algorithm, the initial central search point, as well as the search patterns, are predicted using the direct neighbor MBs, namely, the left, top, and top-right neighbor MBs depicted in fig. 1(b).

The selection between small and large diamond search patterns is performed based on the characteristics of the motion in the considered neighbor MBs and on two threshold values, L1 and L2. Hence, motion is characterized: *i*) as *low* when the magnitude of the largest MV of the three MBs is below a threshold L1; *ii*) as *medium* when this value is between L1 and L2, and *iii*) as *high* when the magnitude is greater than L2. When the motion is characterized as *low*, the algorithm adopts a SDSP with the (0,0) point as a central search point and the small diamond is moved until the minimum distortion is found in the center of the diamond. If the motion is characterized as *medium*, the algorithm uses the same central point, but with LDSP. The large diamond search pattern is applied till the minimal distortion block is found in the center and an additional step is performed with the SDSP. Finally, when motion activity is considered *high*, the central point is chosen as the minimum distortion point among the predictor MVs, as well as the location with coordinates (0,0). The algorithm performs the SDSP until the minimum distortion point is found in the center.

This procedure corresponds to a typical adaptive ME algorithm, that only considers few search locations, because most of the times the best match is found around the central point of the search area. Nevertheless, the algorithm performs a large pattern searching or a central point replacement if the motion is characterized as *high*.

In what concerns the complexity, although all these algorithms are good candidates for software implementations, this paper proves that it is possible to develop hardware processors to efficiently implement any adaptive algorithm of this class, as well as any fast block matching algorithm, by adopting the ASIP approach. The proposed ASIP was specified by analyzing the characteristics of adaptive motion estimation algorithms, but it is able to efficiently implement any fast motion algorithm of both classes referred above. In fact, the FSBM, 3SS, 4SS, DS and MVFAST algorithms have been implemented in the proposed ASIP, in order to evaluate the performance of the processor.

3. ASIP instruction-set, microarchitecture and software tools

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 operations (eight) that focus the most widely executed instructions in ME algorithms. This register-register approach was adopted due to its simplicity and efficiency. On the one hand, it allows 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 Register (GPR)s, 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 trade-off between the implementation area, memory traffic and the size of the program memory. For the proposed ASIP, the register file consists of 24 GPRs and eight Special Purpose Register (SPR)s capable of storing one 16-bit 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 half-word 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 [9, 8]. The encoding of the instructions into binary

Table 1. Instruction-set architecture of the proposed ASIP.

Instruction	15 13	12	11 8	7 5	4 3 0
LD	`000`	t	-		
J	`001`	cc	-	#addr	
MOVR	`010`	Rd		-	Rs
MOVC	`011`	t	Rd	#const	
SAD16	`100`	-	Rd	Rs1	Rs2
DIV2	`101`	-	Rd	Rs	-
ADD	`110`	-	Rd	Rs1	Rs2
SUB	`111`	-	Rd	Rs1	Rs2

representation was performed using 16-bit 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 trade-off between the program size and the efficiency of the architecture. In the following, it is presented a brief description for the operating mode of all the operations of the proposed ISA.

3.1.1 Control operation

The jump control operation, `J`, introduces changes 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-bit 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 to improve code density, since it allows to minimize the number of instructions required to implement a ME algorithm and therefore to reduce the capacity of the required 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-bit width, which restricts the range of constants that can be used. To overcome this problem, the `MOVC` instruction has also a control field (t) that sets the loading of the 8-bit 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 on 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 similarity measure between a MB and a candidate block. To do so, this operation computes the SAD value for two sets of sixteen 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 SPRs. The computation of a SAD value for a given candidate (16×16) pixels 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 proposed ISA does not provide a register file with enough capacity to store all the data required for the implementation of an efficient `SAD16` operation. To overcome this drawback, it comprises a small and fast local memory, that stores all the pixels of a MB and of its corresponding search area, and includes a memory data transfer operation, `LD`, that loads the data into this local memory. Nevertheless, the loading of the data concerning a MB and a corresponding search area is performed independently. As a result, the `LD` operation has a 1-bit control field to specify the type of image area to be loaded into the local memory.

3.2. Microarchitecture

The proposed ISA is supported by a specially designed microarchitecture, following strict power and area driven policies to support its implementation in portable and mobile platforms. This microarchitecture presents a modular structure and is composed by simple and efficient units to optimize the data processing, as it can be seen from fig. 2.

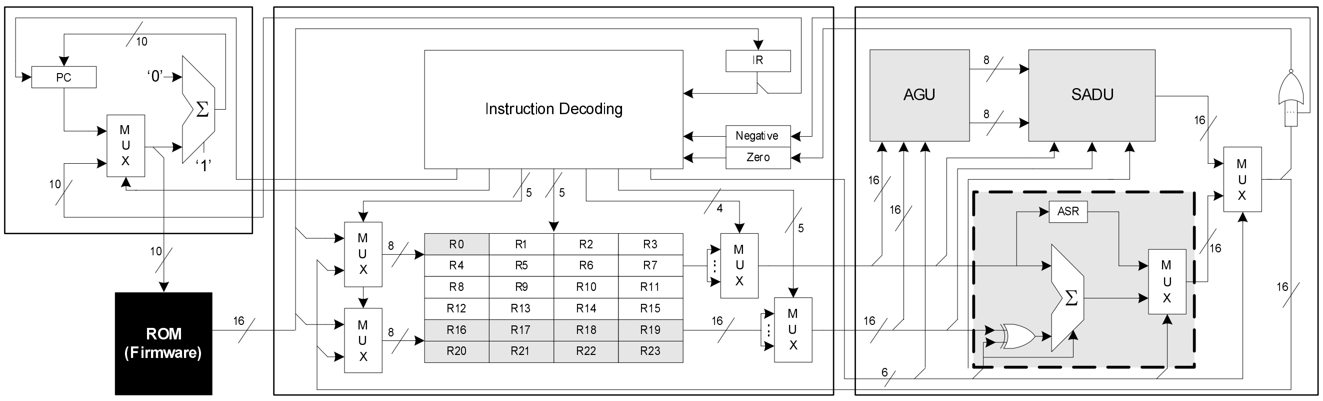

Figure 2. Architecture of the proposed ASIP.

The control unit is also characterized by its low complexity, due to both the adopted fixed instruction encoding format and to 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 gatting. This 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.

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

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 for implementing 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 with no 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 trade-offs between the latency, the power consumption and the required implementation area, thus providing increased flexibility for different implementations of the proposed ASIP.

The `LD` operation is executed by a dedicated AGU optimized for ME, which is capable of fetching all the pixels for both a MB and an entire search area. To maximize the efficiency of data processing, this unit can work in parallel with the remaining functional units of the microarchitecture. 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 implementation area, memory accesses can be further optimized by using efficient data reuse algorithms and extra hardware structures [3, 1]. 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.

3.3. Software tools

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

Since the proposed ASIP architecture and the considered instruction set do not support subroutine calls neither makes use of an instruction/data stack, the implementation of the compiler consists in a straightforward parsing of the assembly instruction directives as well as their register operands, and a corresponding translation into the appropriate opcodes, in order to translate the sequence of assembly instructions into a series of 16-bit machine code words of program data. The exception to this direct translation occurs whenever a *jump* instruction has to be compiled. As a consequence, the implemented compiler adopted a two-step strategy to compile these control flow instructions, in order to determine the target address of each *jump* invoked within the program.

To illustrate the operation of the conceived compiler, it is presented in fig. 3 a fraction of one of the output files (code.lst) that are generated during this translation process. In particular, this file presents three different sorts of information, disposed in three columns (see fig. 3). 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 for the considered algorithm comparisons). As it can be seen, the resulting SAD value, accumulated in register R1 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 registers R2, R6 and R7, in order to be considered in the next searching locations. In the remaining instructions, the motion vector coordinates are incremented and it is checked if the last column and line of the considered search area were already reached, respectively.

To support the implementation and evaluation of the ME algorithms, an accurate simulator of the proposed ASIP was also implemented. This tool provides the means to closely simulate the operation of the processor architecture in a cycle by cycle basis, in order to assess the efficiency of the considered algorithm, by providing important information such as: the number of clock cycles required to carry-out the motion estimation of a given macroblock, the amount of memory space required to store the program code, the obtained MV and corresponding SAD value, etc.

Besides this instruction level simulation, the results obtained both from the processor implementation, as well as from the processor simulation, were validated by implementing each of the considered ME algorithms using a pure software approach, by adopting the C programming language.

```
 ...
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 3. Fraction of one of the output files obtained with the assembly compiler.

4. Experimental Results

To assess the performance provided by the proposed ASIP, a simpler version of the microarchitecture was developed by using a completely serial processing architecture for the SADU (see fig. 4) and a simplified AGU that does not allow data re-usage. This microarchitecture was described using both behavioral and fully structural parameterizable IEEE-VHDL and implemented using *Synopsis* synthesis tools and a high density StdCell library from *UMC*, which is based on a $0.13\mu m$ CMOS process from *Virtual Silicon Technology Inc.* [10]. The obtained experimental results concern an operating environment imposing typical operating conditions: $T = 25^oC$, $V_{dd} = 1.2V$, the "suggested_10k" wire load model and some constraints that lead to an implementation with minimum area.

The performance of the proposed ASIP was evaluated by using the FSBM, the 4SS, the DS and the adaptive MV-FAST motion estimation algorithms. These algorithms have been programmed for the proposed instruction set and the ASIP operation was simulated by using the developed software tools (see section 3.3). 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.

Tables 2 and 3 provide the average number of clock Cycles Per Frame (CPF) required to implement the algorithms using the Quarter Common Intermediate Format (QCIF) (QCIF – 176×144 pixels) and the Common Intermediate Format (CIF) (CIF – 352×288 pixels) image format, respectively, for the following benchmark video sequences: *Mobile*, *Carphone*, *Foreman*, *Table Tennis*, *Bus* and *Bream*.

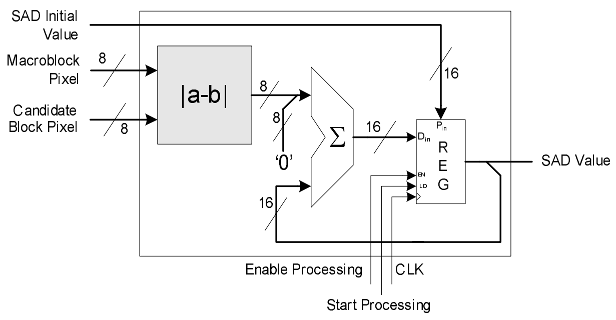

Figure 4. Block diagram of the SADU.

Table 2. CPF for different algorithms and QCIF video sequences.

Video Seq.	FSBM		4SS	DS	MVFAST
mobile	4268632	100%	6.74%	5.18%	2.18%
carphone	4272068	100%	7.34%	6.10%	3.85%
foreman	4270610	100%	7.71%	6.47%	4.47%
table tennis	4267524	100%	6.99%	5.57%	2.79%
bream	4271976	100%	6.98%	5.54%	2.89%

Table 3. CPF cycles for different algorithms and CIF video sequences.

Video Seq.	FSBM		4SS	DS	MVFAST
mobile	21683460	100%	6.90%	5.58%	3.29%
carphone	21694705	100%	7.83%	6.62%	4.65%
foreman	21684114	100%	7.90%	6.54%	4.76%
table tennis	21673815	100%	7.03%	5.58%	2.88%
bus	21678763	100%	8.79%	7.72%	6.61%

These are well known video sequences with quite different characteristics, both in terms of movement and spacial detail. Average results are given for a search area with 16×16 candidate locations and for the first 20 frames of each video 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 in tables 2 and 3 show the huge reduction that was achieved in the amount of computations, when fast search algorithms were applied. The MVFAST adaptive algorithm allows to significantly reduce the CPF even further, when compared with the 4SS and the DS fast algorithms. By considering the worst case for the CPF (CPF$_{wc}$) and a frame rate of 30 Hz, the required maximum operating frequencies (ϕ) can be calculated for each class of algorithms using eq. 2.

$$\phi = CPF_{wc} \times 30 \text{ Hz} \qquad (2)$$

Using the values in table 2 and eq. 2, the ASIP was synthesized for the clock frequencies of 3 MHz for the MVFAST algorithm, 5 MHz for the 4SS algorithm and 50 MHz for the bottom line case of the FSBM algorithm. Nevertheless, the maximum operating frequency of the proposed motion estimators is significantly lower than the frequency of the ± 1 full-search based processor presented in [2].

Although the evaluation of the quality of the encoded video sequences is out of the scope of this paper, the average SAD values obtained with the various algorithms are presented in tables 4 and 5. As it is already known [9], the minimum value of the SAD only increases significantly when the fast search algorithms are applied to the *Bus* sequence, which is a sequence with a lot of non-central based movement. Moreover, in most of the cases the DS an MVFAST algorithms give similar results in terms of block matching

Table 4. Relative increase of SADmin when fast algorithms are used for QCIF video sequences.

Video Seq.	FSBM	4SS	DS	MVFAST
mobile	1907	+0.05%	+0.05%	+0.00%
carphone	774	+6.97%	+2.03%	+3.73%
foreman	607	+15.34%	+7.04%	+5.89%
table tennis	641	+5.04%	+3.75%	+8.03%
bream	925	+4.05%	+2.12%	+2.94%

Table 5. Relative increase of SADmin when fast algorithms are used for CIF video sequences.

Video Seq.	FSBM	4SS	DS	MVFAST
mobile	2335	+2.26%	+1.56%	+0.89%
carphone	781	+7.35%	+3.34%	+4.17%
foreman	676	+6.76%	+11.05%	+6.50%
table tennis	779	+3.47%	+3.71%	+4.06%
bus	1789	+46.11%	+54.88%	+48.65%

distortion, whereas the MVFAST significantly reduces de CPF and consequently the operating frequency of the ASIP and thus its power consumption.

The first main conclusions that can be drawn from the results in table 7 is that the power consumption of the proposed ASIP for motion estimation with the adaptive MVFAST algorithm is very low. Operating at a frequency of 3MHz, it only requires about 60μW, which does not imply any significant reduction of the life time of our actual batteries (typically 1500mAh batteries). For the 4SS algorithm, the operating frequency increases to about 5MHz but the power consumption is kept low, about 100μW. At the bottom line, the FSBM 50MHz processor consumes about 1mW, which is not a very high frequency neither a high consumption, for exhaustively searching all the candidate blocks in a 16×16 search area. The maximum operating frequency obtained with this architecture and this technology is 307MHz, much higher that the highest frequency required for FSBM. At this maximum frequency, which corresponds to having the components of the processor operating near its maximum speed, as depicted in table 6, the consumption becomes 8mW.

In what concerns the code size (see table 8), the MVFAST requires significantly more memory for storing the program than the 4SS, and approximately 13 times more memory than the FSBM. This is the price to pay for irregularity and also for the adaptability of the MVFAST algorithm (744×16 bit). However, since most of the portable communication systems already provide memories with relatively high capacity, this should not be considered as an additional cost for the design of a complete video encoder system.

Table 6. Experimental results for the components of ASIP synthesized for the maximum frequencies and $0.13\mu\mathrm{m}$ CMOS technology.

Unit	Area	Max. Freq.	Power
ALU	1493 μm^2	833 MHz	2 mW
AGU	27541 μm^2	571 MHz	6 mW
SAD	4598 μm^2	606 MHz	3 mW

Table 7. Experimental results for the ASIP synthesized at four different frequencies and $0.13\mu\mathrm{m}$ CMOS technology.

Setup	Area	Max. Freq.	Power
Max. Freq.	64541 μm^2	307 MHz	8 mW
FSBM	67540 μm^2	66 MHz	1 mW
3SS, 4SS	67540 μm^2	5 MHz	0.1 mW
MVFAST	67540 μm^2	3 MHz	0.06 mW

Table 8. Code size for proposed algorithms.

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

5. Conclusions

This paper presents an innovative design flow to implement efficient motion estimators. Such approach is based on an ASIP platform characterized by a specialized datapath and minimum and optimized instruction set, that was specially developed to allow an efficient implementation of data-adaptive motion estimation algorithms.

Moreover, the paper also presented the set of software tools that was 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 simulating its operation for implementations of regular (FSBM), irregular (4SS and DS) and adaptive (MVFAST) ME algorithms using the developed software tools. Furthermore, the performance of the developed microarchitecture was also assessed by implementing it in ASIC using a high density StdCell library based on a $0.13\mu m$ CMOS process.

The experimental results showed that the proposed ASIP allowed the estimation of MVs in real time for the QCIF image format for all the tested ME algorithms, when operating at relatively low operating frequencies. Furthermore, the results also showed that the power consumption of the proposed architecture is very low: about 60μW for the adaptive algorithm, around 100μW for the irregular algorithms and near 1mW for the regular one. Consequently, it can be concluded that the low-power nature of the proposed architecture and its high performance makes it highly suitable for implementations in portable, mobile and battery supplied devices.

References

[1] T. Dias, N. Roma, and L. Sousa. Efficient motion vector refinement architecture for sub-pixel motion estimation systems. In *IEEE Workshop on Signal Processing Systems - SiPS'2005*, pages 313–318. IEEE, November 2005.

[2] S.-Y. Huanga and W.-C. Tsai. A simple and efficient block motion estimation algorithm based on full-search array architecture. *Signal Processing: Image Communication*, 19:975–992, 2004.

[3] T.-S. C. Jen-Chieh Tuan 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*, 12(1):61–72, Jan. 2002.

[4] 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*, 4(4):438–442, Aug. 1994.

[5] 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*, 6(3):313–317, June 1996.

[6] N. Roma and L. Sousa. Efficient and configurable full search block matching processors. *IEEE Transactions on Circuits and Systems for Video Technology*, 12(12):1160–1167, Dec. 2002.

[7] S. Saponara and L. Fanucci. Data-adaptive motion estimation algorithm and VLSI architecture design for low-power video systems. *IEE Proc.-Comput. Digit. Tech.*, 151(1):51–59, Jan. 2004.

[8] A. Tourapis. Enhanced predictive zonal search for single and multiple frame motion estimation. In *Proceedings of SPIE - Visual Communications and Image Processing (VCIP)*, pages 1069–1079, San Jose, CA, Jan. 2002.

[9] A. Tourapis, O. Au, and M. Liou. Predictive motion vector field adaptive search technique (PMVFAST) - enhancing block based motion estimation. In *Proceedings of SPIE - Visual Communications and Image Processing (VCIP)*, pages 883–892, San Jose, CA, Jan. 2001.

[10] Virtual Silicon Technology Inc. *UMC High Density Standard Cells Library - 0.13μm CMOS process*, v2.3 edition, Dec. 1999.

[11] S. Zhu and K.-K. Ma. A new diamond search algorithm for fast block-matching motion estimation. *IEEE Transactions on Image Processing*, 9(2):287–290, Feb. 2000.