Programmable IP core for motion estimation: comparison of FPGA and ASIC based implementations

Tiago Dias *INESC-ID / ISEL*
Nuno Sebastião *INESC-ID*
Nuno Roma *INESC-ID / IST*
Paulo Flores *INESC-ID / IST*
Leonel Sousa *INESC-ID / IST*

Abstract

A performance analysis of two distinct implementations of a recently proposed quite efficient motion estimation co-processor is presented. This comparison considers two distinct implementation technologies: a high performance FPGA device, from Xilinx Virtex-II Pro family, and an ASIC based implementation, using a $0.18\mu m$ *CMOS standard cells library. Experimental results have shown that the two considered implementations present quite similar performance levels and allow the estimation of motion vectors in real-time. Nevertheless, the reconfigurability properties of the FPGA implementation allow the motion estimator to dynamically adapt the video encoder to the characteristics of the target application and/or of the communication channel.*

1. Introduction

In the last few years there has been a growing trend to design very complex and efficient processing systems by integrating already developed and dedicated cores which implement, in a particularly efficient way, certain specific and critical parts of the main system. Such design approach can either be conducted in order to obtain very complex and autonomous processing architectures, or to implement specific and dedicated processing structures that will be integrated with other larger scale processing modules, in the form of co-processors, to alleviate the computational burden. As a consequence, a significant amount of quite different processing modules have been proposed and made available, providing an easy integration with the target processing systems and a substantial reduction of the design effort. To attain such objective, these processing cores have to follow strict design methodologies, in order to provide an easy and efficient implementation in a broad range of target implementation technologies (e.g.: FPGA, ASIC, etc.).

One of such modules that has deserved a particular attention in the scope of digital video coding is the motion estimator. In fact, although this block is often regarded as one of the most important operations in video coding to exploit temporal redundancies in sequences of images, it often represents most of the computation cost of these systems [1]. As a consequence, real-time Motion Estimation (ME) is usually only achievable by adopting specialized VLSI structures to implement this processing task.

Most of the processing cores of hardware motion estimators that have been proposed in the literature [2, 3] consist of custom ASIC implementations of the Full-Search Block-Matching (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 that implement sub-optimal search strategies, such as the Three-Step-Search (3SS), the Four-Step-Search (4SS) and the Diamond Search (DS) [4, 5]. However, the highly irregular control-flow that characterizes such algorithms tends to compromise the efficiency of such architectures and has therefore limited their application to general purpose programmable systems. Although some individual hardware architectures were actually proposed [4, 5], they usually resulted in complex and inefficient hardware designs, that do not offer any reconfiguration capability. Meanwhile, data-adaptive ME algorithms have been proposed, as a result of the recent advent of the H.264/AVC encoding standard [6]. 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 temporal and spacial correlations of the Motion Vectors (MVs), in order to adapt and optimize the search patterns. However, few hardware implementations have been presented for these new ME approaches, mainly because of the inherent computational complexity of their operation to adjust, themselves, to the characteristics of both the video input signal and the encoding system. Even so, a highly efficient processing core of an Application Specific Instruction Set Processor (ASIP) that is capable of implementing any of these complex ME algorithms, was recently proposed [7].

Independently of the application scenario, the advent of the most recent generations of FPGAs has proved that these devices can be regarded as feasible alternatives to other more costly implementation platforms, such as ASICs. In fact, due to their highly reconfigurable nature and their capability to implement the highest demanding processing applications, FPGAs not only offer significant advantages in what concerns the design and implementation time, but they also represent a cost effective means to implement a given circuit. In fact, not only do they provide an easy integration with other larger scale processing structures, but they also may be easily applied to implement external processing structures (e.g.: within PCMCIA cards [8]) that accelerate the execution of a given part of the processing algorithm. Nevertheless, the quite distinct implementation technologies that are inherent to these two platforms often

Table 1: Instruction-set architecture of the motion estimation co-processor.

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				Rs1			-				
110	ADD	Arithmetic	opcode	-	Rd				Rs1			Rs2				
111	SUB	Arithmetic	opcode	-	Rd				Rs1			Rs2				

demand an effective comparison in what concerns the performances of the obtained circuit. In this paper, it is presented a performance analysis of two distinct implementations of the ME co-processor circuit recently proposed in [7]. This comparison considers two distinct implementation technologies: a high performance FPGA device, from Xilinx Virtex-II Pro family, and an ASIC based implementation, using a $0.18\mu m$ CMOS standard cells library.

2. Motion estimator architecture

The programmable and specialized architecture for ME proposed in [7] 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 Macroblock (MB) structures, such as the traditional fixed 16×16 pixels block size, adopted in the H.261/H.263 and in the MPEG-1/MPEG-2 video coding standards, or even any other variable block-size structures, adopted in the H.264/AVC coding standard. This flexibility was attained by developing a simple and efficient micro-architecture, that supports a minimum and specialized instruction set. The data-path was also designed around a specialized arithmetic unit that efficiently computes the Sum of Absolute Differences (SAD) similarity function. Furthermore, the several control signals are generated by a quite simple and hardwired control unit [7].

2.1. Instruction Set

The Instruction Set Architecture (ISA) of the proposed 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 (eight), that focus on the set of operations that are most widely used in ME algorithms.

It also offers an increased efficiency level, mainly due to its large and configurable number of General Purpose Registers (GPRs), which provides a reduction of the memory traffic and a subsequent decrease of the program execution time. The amount of registers that compose the register file therefore results as a trade-off between the hardware resources, the memory traffic and the size of the program memory. For the proposed configuration, the register file consists of 24 GPRs and 8 Special Purpose Registers (SPRs), capable of storing one 16-bit word each. The GPRs sub-bank is split in two different groups. The first group comprises registers R0-R15 and can be used to hold the source and destination operands of all the instructions of the proposed architecture. On the other hand, all the registers of the other group, consisting of registers R16-R24, can only be accessed using two specific instructions: `MOVR`, which allows data exchange between the two groups, and the specialized graphics operation `SAD16`, whose destination operand can be written in any of these registers. The SPRs can also be accessed using these two specific instructions or even directly accessed by the specialized functional units of the proposed architecture, to optimize the micro-architecture performance.

The set of operations supported by the proposed ISA is divided in five different categories of instructions, as it can be seen in Table 1, and was obtained as the result of an exhaustive 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.

2.1.1. Control operation

The jump 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 3-bit condition field (cc) that specifies the condition that must be verified for the jump to be taken: always; 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; or in case the Address Generation Unit (AGU) or the SAD Unit (SADU) are operating.

2.1.2. Register data transfer operations

These 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.

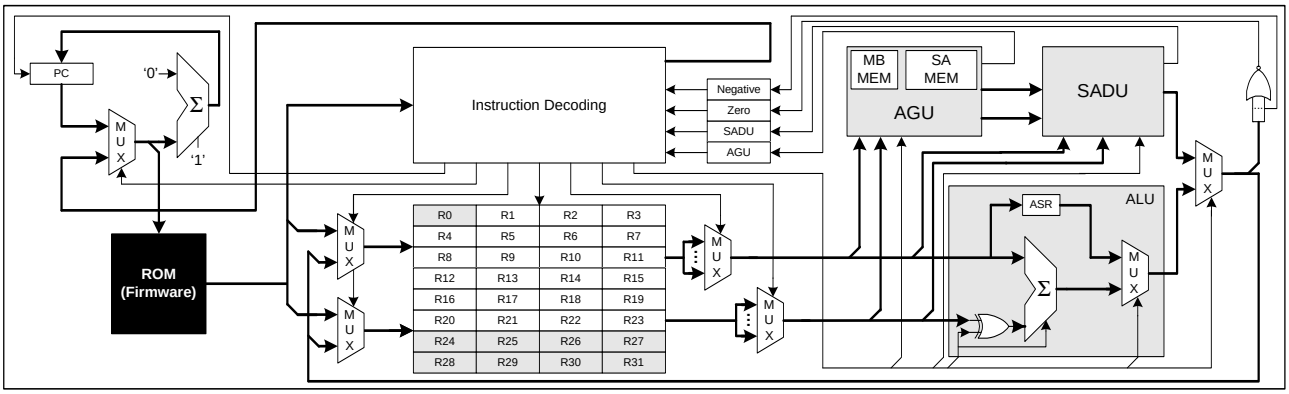

Figure 1: Architecture of the proposed ASIP (SPRs are shaded in the register file).

2.1.3. 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. 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.

2.1.4. Graphics operation

The `SAD16` instruction provides 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 (16×16) pixels candidate MB therefore requires the execution of sixteen consecutive `SAD16` operations. Likewise the previously described arithmetic operations, the outcome of this operation also provides some extra information that may be used by the jump (`J`) control instruction to conditionally change the control-flow of a program.

2.1.5. Memory data transfer operation

The processor comprises two fast and small scratch-pad local memories to store the pixels of the MB under processing and 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 Address Generation Unit (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.

2.2. Micro-architecture

The proposed ISA is supported by a specially designed micro-architecture. It presents a modular structure and is composed by simple and efficient units that optimize the data processing, as it can be seen in Fig. 1.

2.2.1. Control unit

The control unit is characterized by its extremely low complexity, due to the adopted fixed instruction encoding format and a careful selection of the opcodes for each instruction. This design option provided the implementation of a very simple and fast hardwired decoding unit, which enables almost all instructions to complete in just one clock cycle.

2.2.2. Datapath

The datapath includes specialized units to increase the efficiency of the most complex and specific operations, such as the `LD` and the `SAD16` instructions: 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. Furthermore, to ease the programmer task, the micro-architecture also makes available a specific flag in its program status register that reflects the operation state of this unit.

The SADU can execute the `SAD16` operation in up to sixteen clock cycles. It is also capable of operating the Arithmetic and Logic Unit (ALU) in parallel with the execution of the `SAD16` operation, 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 specific type of architecture that is selected in the configuration of this unit at design time, which depends on the specified performance and hardware resources constraints. Thus, applications imposing more severe con-

straints 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. Despite the type of architecture adopted for the SADU, the proposed micro-architecture makes always available a specific flag in its program status register that indicates if the result of the last computed SAD16 operation is a minimum SAD value.

3. Integration with the video encoding system

To embed the proposed ASIP core as a ME co-processor in a video encoding system, a simple and efficient interface for both the data and control signals must be made available. In addition, the co-processor must also use simple and efficient protocols, to exchange the control commands and data with the main processing unit of the video encoding system.

3.1. Interface

The proposed programmable and configurable architecture for ME presents a simple and reduced pin count interface, as it can be seen in Fig. 2. Such interface was designed to allow the fetching of the pixel data required for the ME task from the main frame memory of the video encoding system, i.e., the pixels of the reference macroblock and of its corresponding search area. In addition, the proposed interface is also able to efficiently export the configuration parameters and the output results of the ME operation to the video encoding system main processing unit, i.e., the coordinates and the SAD value for the computed best matching MVs.

The data transfers with the video encoder frame memory are mostly supported through five I/O ports, as it can be seen in Fig. 2. The 1-bit port, *#oe_we*, is used to set the type of external memory operation: a load or store. The *addr* port is 20-bit width and is used to select the position of the frame memory from which the pixels of a reference macroblock, or those of a search area, are to be retrieved by the load operation. Since pixel values for ME are usually represented using only 8-bits, an 8-bit width signal is used to exchange data with the frame memory. Such signal is available at the *data* port of the proposed structure. Thus, the total memory address space provided by this programmable architecture is 1 MB. Considering that MVs are estimated using pixels from two different frames, the reference and search frames, such address range therefore allows the computation of MVs for the most used image formats (e.g.: in the 4CIF image format each frame consists of 704×576 pixels). In what concerns the store operation, it also makes use of the *data* port to transfer, to the video encoder, the result of the ME operation, as well as the configuration parameters of the ME co-processor used in such computation. These parameters consist of the horizontal and vertical coordinates of the computed best matching MV, its corresponding SAD value, the MB size, the search area size and the image width and height, which can be dynamically adjusted by the ME algorithm implemented in the ME co-processor. The video encoder accesses these parameters by reading a reserved memory region of the frame memory, beginning at memory address 0xFFFF0 and encompassing 16 memory locations, as depicted in Fig. 3. Consequently, the number of required I/O connections is minimized.

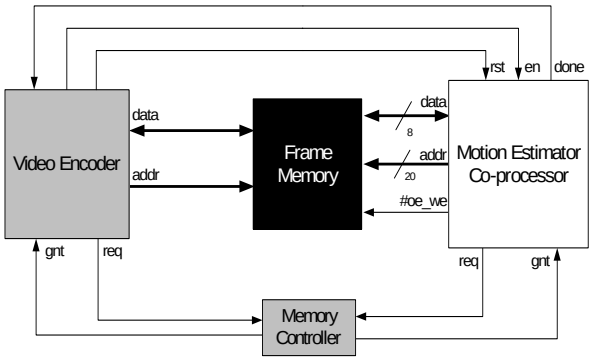

Figure 2: Interface with the video encoding system.

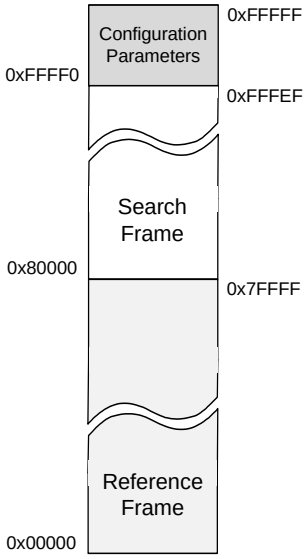

Figure 3: Memory map of the proposed ASIP.

The two remaining I/O ports provided by the proposed architecture, *req* and *gnt*, are used to implement the required handshake protocol with the bus master of the video encoding system. Such control task is required not only because the frame memory bank is shared between the ME co-processor and the main processing unit of the video encoder, but also to optimize the memory usage and minimize the memory bandwidth requirements of the frame memory.

The *en* input port is used to control the co-processor operation, while the *rst* input port is used to set the processor into its startup state. The *done* output port is used to signal the video encoder that the ME co-processor has completed the estimation of a new MV.

3.2. Communication protocols

The communication between the proposed programmable architecture and the video encoder is achieved

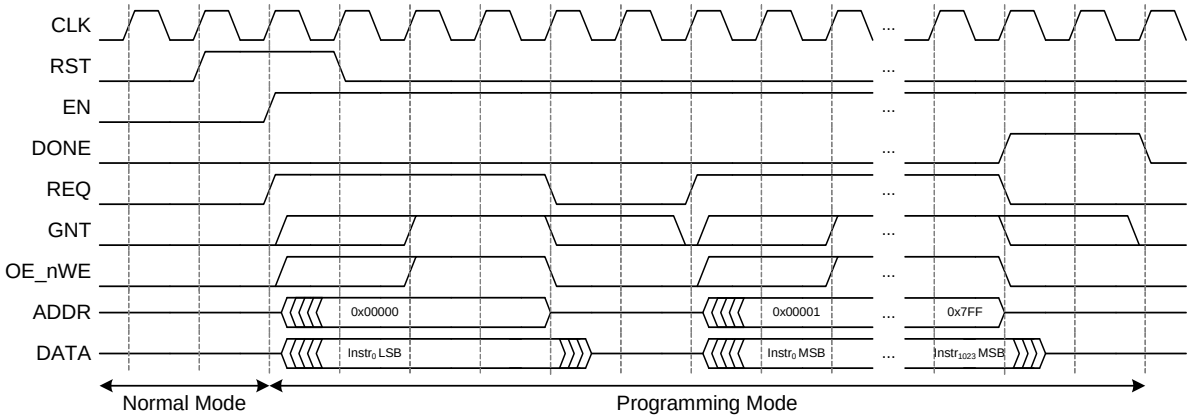

Figure 4: Temporal diagram concerning the loading of the firmware into the proposed ASIP.

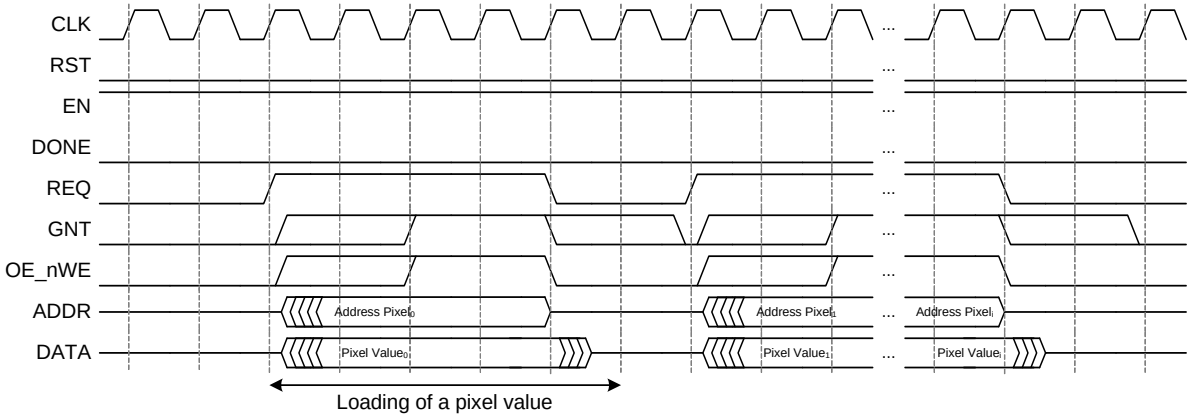

Figure 5: Temporal diagram concerning the loading of MB/SA pixels into the proposed ASIP.

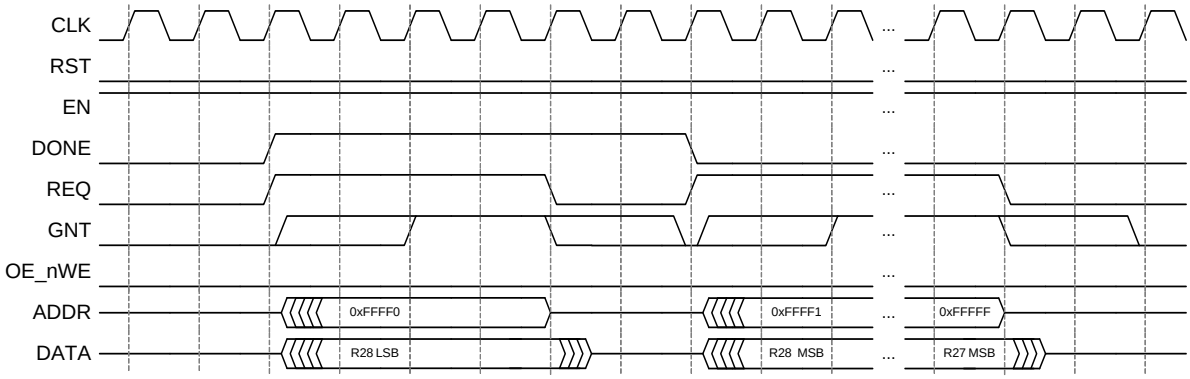

Figure 6: Temporal diagram concerning the output of the result of a ME operation.

through the interface signals described in section 3.1 by using three distinct simple and efficient protocols. Such protocols are aimed to support the operating principle of the video encoding system, consisting of only three different tasks.

The first task consists in the configuration of the ME co-processor, by downloading the compiled assembly code of the considered ME algorithm to the co-processor (i.e., the co-processor's firmware). Both the firmware and the ME configuration parameters are uploaded into the co-processor's program RAM through the *data* port. The configuration of the co-processor is therefore achieved by first storing the required data in the frame memory of the video encoder system and by setting the co-processor into its *programming mode*. The co-processor enters in this mode when both signals, *rst* and *en*, are high, as it can be seen in Fig. 4. In this *programming mode*, the co-processor firstly acquires the bus ownership. Then, it supplies memory addresses through the *addr* port to the frame memory, in order to download the corresponding instructions into its internal program RAM, organized in the little-endian format. Since each instruction is 16-bit width, two memory access cycles are required to load an instruction into the program memory. The co-processor exits the *programming mode* as soon as the last memory position of its 2 kB program memory is filled in. Such approach allows to minimize the number

of required I/O connections of the ME co-processor without degrading its efficiency, since the downloading of a ME algorithm into the ME co-processor is not very often executed.

The second task consists in all data transfers concerning not only the pixels of a given reference macroblock and of the corresponding search area, from the video encoder frame memory to the ME co-processor, but also all the control parameters required to the ME operation, namely, the MB size, the search area size, the image width and the image height. An entirely similar protocol is used to support this task, as depicted in Fig. 5. This task occurs on the co-processor's demand and is controlled by the AGU of the proposed programmable architecture. Consequently, the AGU must firstly generate the required control signals for the co-processor to acquire the bus ownership before initiating the pixel data transfer. Then, the AGU supplies the correct memory addresses to the frame memory through the *addr* port, so that all the pixels of the macroblock, or of the search area, are retrieved from the external frame memory and loaded into the local scratch-pad memories of the ME co-processor. Since each pixel is represented using 8-bits, a single memory access cycle is required to transfer a pixel value from the external frame memory into the local memories of the ME co-processor.

The third task consists in transferring, to the video encoder, both the result of the ME operation, as well as the ME configuration parameters updated by the co-processor during its operation. A different protocol is used, but again, such data transfer occurs on demand by the co-processor. However, it is now controlled by the main control unit of the proposed programmable architecture. The controller starts the output operation by requesting the bus ownership, as it can be seen in Fig. 6. Then, it enters in a loop that outputs the contents of all the co-processor's SPRs through the *data* port. Two memory access cycles are required for this operation, since the SPRs are 16-bit width and the output *data* port is only 8-bit width. 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 was uploaded into the reserved memory region of the video encoding system. The memory position used to store the data is selected according to the value being outputted at the *addr* port.

4. Prototyping platform

To validate the functionality of the proposed programmable architecture for ME in a practical realization, a complete video encoding system was developed and implemented. Two different platforms were considered for the prototyping, in order to evaluate the performance of the proposed ME core when implemented using both FPGA and ASIC target technologies. Nevertheless, the base configuration of such system is the same for both cases. It consists of a general purpose processor that executes all the video encoder operations, except for the ones concerning ME. The ME operations are executed by the proposed programmable architecture, acting as a specialized co-processor of the main processing unit of the video encoder, i.e., the general purpose processor. This co-processor computes, in parallel, the several MVs that are required by the encoder to compute the prediction error of the video signal.

The implemented video encoder consists of a software implementation of the H.263 encoding standard, provided by Telenor R&D (TMN5) [9]. Such implementation includes some optimizations of the original version, provided by Telenor R&D, in order to make its use more efficient in embedded systems. The modifications include the redesign of all the functions used in ME and the declaration of all variables in these functions using the prefix *register*, so as to optimize the program execution time. To maximize the performance of the encoder, the linker script of the video encoding system was also adapted to the target embedded system. Such modifications aimed at optimizing the data transfers from the video encoder main memory module to the ME co-processor, concerning the pixels of the reference blocks and of its corresponding search area.

4.1. FPGA based prototype

The implementation of the proposed video encoding system using an FPGA device was realized using a Xilinx ML310 development platform [10], which includes a 100MHz 256MB DDR memory bank and a Virtex-II Pro XC2VP30 FPGA device from Xilinx. Besides all the implementation resources that are offered by this reconfigurable device, it also provides two Power-PC processors, several Block RAM (BRAM) modules and high speed on-chip bus-communication links. Such communication links 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. The programmable architecture for ME is implemented using the configurable logic blocks provided by the Virtex-II Pro XC2VP30 FPGA device, while the main processing unit of the video encoder consists of a Power-PC 405 D5 processor, operating at 300MHz. Such processor runs the optimized software implementation of the H.263 video encoder [9], which is built into the FPGA BRAMs and the ML310 DDR memory bank. The linker script used for this implementation maximizes the performance of the encoder by taking into account the significantly different access times provided by these two memory banks. To do so, both the *text* and the *me* sections of the application were located in two distinct 128kB FPGA BRAM modules, while the *data*, *stack* and *heap* sections were located in the DDR memory module, due to its large size (more than 256kB). The interconnection between the Power-PC processor and the ME co-processor 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 ME structure.

4.2. ASIC based prototype

The implementation of the proposed programmable architecture for ME in an ASIC was carried out using an AT91SAM9263-EK evaluation kit [11] from ATMEL. This development board includes an AT91SAM9263 [12] processor, which is based on the ARM926EJ-S core, widely adopted by the latest generation of mobile phones and PDAs, and an extensive set of peripherals for control, communication and data storage purposes. This set of peripherals also includes all the components required to implement a modern video encoding system, i.e., a graphical 1/4 VGA TFT LCD module, an ISI connector that provides interface to video cameras and a 100 MHz 64MB SRAM memory bank. In addition, this development board also offers the possibility to embed user-developed peripherals into the prototyping platform, by making available some connectors to the External Bus Interface (EBI) of the processor.

The main processing unit of the implemented video encoder consists of the AT91SAM9263 processor, operating at 99.33MHz. Just as the FPGA prototyping system, such processing unit runs the optimized software implementation of the H.263 video encoder [9]. In this prototyping platform, all program code and data sections of the application are located in the 64MB SRAM memory bank. An expansion slot is used to connect the AT91SAM9263-EK prototyping platform to a daughter board, with the ASIC implementation of the proposed ME programmable architecture, by making use of the processor's EBI. Similarly to the FPGA prototyping platform, the EBI is used to exchange the control signals and all the required data between the processor and the ME co-processor.

5. Implementation and experimental results

The performance analysis of the FPGA and the ASIC implementations of the proposed programmable ME core was realized for a specific configuration of this parameterizable structure. The considered setup, which was described using both behavioral and fully structural parameterizable IEEE-VHDL, adopted a simplified AGU that does not allow data re-usage and a power efficient serial processing structure for the SADU module [13]. Such architecture was selected as the result of a compromise between the amount of required hardware resources, the circuit power consumption and its usability for real-time operation. Previous research work [13] has shown that for single reference frame ME and for image formats up to CIF (352×288 pixels), a serial structure for the SADU presents the best trade-off between the parameters formerly described. However, depending on the target application, the proposed architecture can be reconfigured to use other AGU and SADU modules that represent different trade-offs.

5.1. FPGA implementation

The video encoding system described in section 4.1 was implemented in the Xilinx ML310 development platform, using the EDK 8.1i and ISE 8.1i tools from Xilinx. Table 2 presents the obtained implementation results. These values do not consider the BRAM modules that are used to store the compiled assembly code of the ME algorithm and the pixel data for both the reference and search areas. Nevertheless, these results evidence that FPGA based implementations of the proposed ME architecture allow the estimation of MVs in real-time for the QCIF and CIF image formats. They also show that very few hardware resources (less than 6k equivalent logic gates) are required to implement the proposed ME co-processor in an FPGA device.

Table 2: Implementation results of the motion estimator using the Virtex-II Pro XC2VP30 FPGA device.

Occupied Slices	2046 (14%)
Occupied LUTs	1844 (6%)
Estimated Equivalent Logic Gates	6 kGates
Occupied BRAMs	2 (1.5%)
Maximum operating frequency	85.84 MHz

The functionality of the implemented video coding system was successfully verified by encoding a set of benchmark CIF video sequences with quite different characteristics, both in terms of movement and spacial detail, by using several different ME algorithms and by adopting a typical set of video coding parameters: 8-bits to represent the pixel values, macroblocks with 16×16 pixels and search areas with 32×32 pixels. This performance assessment considered the FSBM, the 3SS, the DS and the adaptive MVFAST ME algorithms, which were programmed using the instruction set presented in section 2.1.

5.2. ASIC implementation

The implementation of the video encoding system described in section 4.2 using the AT91SAM9263-EK evaluation kit from ATMEL was realized using the GNU toolchain for the ARM architecture. The video encoding system includes the expansion board with the ASIC implementation of the ME co-processor. This ASIC adopted exactly the same configuration as the one used for the FPGA implementation of the ME core and also included a complete set of testing structures that can be accessed by means of an integrated JTAG controller. The circuit was manufactured under the *mini*@SIC program from EUROPRACTICE, using a StdCell library [14] based on a 0.18μm CMOS process with 1 poly and 6 metal layers from UMC (UMC L180 1P6M MM/RFCMOS). Table 3 presents the obtained implementation results, after placement and routing, by considering typical environmental operating conditions: $T = 25^oC$ and $Vdd = 1.8V$. These re-

Table 3: Implementation results of the motion estimator using the UMC 0.18μm CMOS ASIC.

Silicon Area / Equivalent Logic Gates	**IP Core**	0.25 mm^2 / 25 kGates
	Test Struct.	0.15 mm^2 / 16 kGates
	Memories	0.68 mm^2 / 70 kGates
Max. operating frequency		100 MHz
Power (Core @100MHz)		31 mW

sults show that the ASIC implementation (excluding the program code and pixel data local memories) only requires 41k equivalent logic gates (111k equivalent logic gates are required to implement the whole processor). When compared with the FPGA implementation, this difference arises from the absence of optimized arithmetic cells, such as fast carry-propagate-like adders and multipliers, that are usually available in FPGAs. Consequently, such arithmetic units had to be fully designed and implemented.

The functionality of the implemented video coding system was verified using the same methodology as the one adopted for the FPGA implementation and proved to allow the real-time computation of MVs for the QCIF and CIF image formats.

5.3. Comparison analysis

The performance results presented in Table 3 for the ME ASIC are quite similar to those obtained with the FPGA implementation, presented in Table 2. They evidence that the ASIC implementation only provides a slightly advantage in terms of operating frequency over the FPGA implementation. In addition, the low power consumption of the implemented ASIC circuit, when operated at 100MHz, proves its suitability to efficiently implement ME algorithms in battery-supplied devices. Nevertheless, it should be noted that despite the slightly better performance levels provided by the ASIC implementation, the FPGA implementation of the ME co-processor also presents important advantages for certain specific video encoding applications, due to its reconfigurability properties. By using such capability to reconfigure the ME co-processor and use different SADU or AGU structures, it is possible to dynamically adapt the video encoder to the characteristics of the target application and/or of the communication channel. In such situations, the FPGA implementation can be regarded as a suitable alternative for video encoding applications running on portable and mobile devices.

6. Conclusions

This paper presents a performance analysis of two distinct implementations of a recently proposed high performance programmable and specialized architecture for ME. Such comparison is performed by considering the integration of such structure in a video encoding system as a motion estimation co-processor, using two quite different technologies: a high performance FPGA device, from the Xilinx Virtex-II Pro family, and an ASIC based implementation, using a $0.18\mu m$ CMOS standard cells library.

The 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 considered implementations present very similar performance levels and allow the estimation of MVs in real-time (above 25 fps) for both the QCIF and CIF image formats. Such results also demonstrated that the power consumption requirements of the ASIC implementation makes it more suitable to efficiently implement ME algorithms in battery-supplied devices. Nevertheless, the reconfigurability properties of the FPGA implementation allow the motion estimator to dynamically adapt the video encoder to the characteristics of the target application and/or of the communication channel.

Acknowledgment

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] V. Bhaskaran and K. Konstantinides. *Image and Video Compression Standards: Algorithms and Architectures*. Kluwer Acad. Publish., 2nd edition, June 1997.

[2] 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, December 2002.

[3] S. Ang, G. Constantinides, W. Luk, and P. Cheung. The cost of data dependence in motion vector estimation for reconfigurable platforms. In *Proc. of Int. Conf. on Field Programmable Technology - FPT'2006*, pages 333–336. IEEE, December 2006.

[4] Y. Jehng, L. Chen, and T. Chiueh. An efficient and simple VLSI tree architecture for motion estimation algorithms. *IEEE Transactions on Signal Processing*, 41(2):889–900, February 1993.

[5] W. Chao, C. Hsu, Y. Chang, and L. Chen. A novel hybrid motion estimator supporting diamond search and fast full search. In *IEEE Int. Symp. on Circuits and Systems - ISCAS'2002*, pages 492–495, May 2002.

[6] Joint Video Team of ITU-T and ISO/IEC JTC1. *ITU-T Recommendation H.264, "Advanced Video Coding for Generic Audiovisual Services"*, May 2003.

[7] T. Dias, S. Momcilovic, N. Roma, and L. Sousa. Adaptive motion estimator for autonomous video devices. *EURASIP J. on Embedded Systems*, 2007.

[8] *WILDCARD^{TM}-II and WILDCARD^{TM}-4 Ref. Manual - Rev. 3.6*. Annapolis Micro Systems, Inc., 2006.

[9] Telenor. *TMN (Test Model Near Term) - (H.263) encoder/decoder - version 2.0 - source code*. Telenor Research and Development, Norway, June 1996.

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

[11] *AT91SAM9263-EK Evaluation Board - User Guide*. ATMEL Corporation, March 2007.

[12] *AT91 ARM Thumb Microcontrollers - AT91SAM9263*. ATMEL Corporation, December 2006.

[13] T. Dias, N. Roma, and L. Sousa. Low power distance measurement unit for real-time hardware motion estimators. In *Int. Workshop on Power and Timing Modeling, Optimiz. and Simulation - PATMOS'2006*, September 2006.

[14] *Faraday ASIC Cell Library FSA0A_C 0.18µm Standard Cell (v1.0)*. Faraday Techn. Corp., August 2004.