Application Specific Programmable IP Core for Motion Estimation: Technology Comparison Targeting Efficient Embedded Co-Processing Units *

Nuno Sebastião†, Tiago Dias†‡, Nuno Roma†§, Paulo Flores†§ and Leonel Sousa†§
†INESC-ID, ‡ISEL-PI Lisbon, §IST-TU Lisbon
Rua Alves Redol 9, 1000-029 Lisboa, PORTUGAL

Abstract

The implementation of a recently proposed IP core of an efficient motion estimation co-processor is considered. Some significant functional improvements to the base architecture are proposed, as well as the presentation of a detailed description of the interfacing between the co-processor and the main processing unit of the video encoding system. Then, a performance analysis of two distinct implementations of this IP core is presented, considering two different target 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 StdCell library. Experimental results have shown that the two alternative implementations have quite similar performance levels and allow the estimation of motion vectors in real-time.*

1. Introduction

In the last few years there has been a growing trend to design very complex processing systems by integrating already developed and dedicated IP cores which implement, in a particularly efficient way, certain specific and critical parts of the main system. Such designs can be conducted in order to obtain complete and autonomous processing architectures, based on a System-on-Chip (SoC) approach. On the other hand, these IP cores can also be used to implement specific and dedicated processing structures that are integrated with other larger scale modules in the form of co-processors.

Consequently, a significant amount of quite different IP cores of specialized processing modules have been proposed and made available, providing a substantial reduction of the design effort. Nevertheless, these IP cores have to follow strict design methodologies, in order to provide an easy integration with the target processing systems and an efficient implementation in a broad range of target technologies.

*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.

One of such modules that has deserved a particular attention in the scope of digital video coding is the motion estimator. This block is often regarded as one of the most important operations in video coding to exploit temporal redundancies in sequences of images, it usually involves most of the computation cost of these systems [7]. As a consequence, real-time Motion Estimation (ME) is usually only achievable by adopting specialized VLSI structures.

The majority of the processing cores for hardware (HW) ME that have been proposed in the literature [13, 4] consist of custom ASIC implementations of the Full-Search Block-Matching (FSBM) algorithm, mostly owed to its regularity and data independence. To achieve real-time ME, few architectures for faster search algorithms have been proposed using sub-optimal search strategies, such as the Three-Step-Search (3SS), the Four-Step-Search (4SS) and the Diamond Search (DS) [11, 8]. However, the highly irregular control-flow that characterizes such algorithms tends to compromise the efficiency of such architectures and has therefore limited their implementation to general purpose programmable systems. In fact, although some individual architectures were actually proposed [11, 8], they usually resulted in complex and inefficient HW designs, that do not offer any reconfiguration capability.

Meanwhile, data-adaptive ME algorithms have been proposed, as a result of the advent of the H.264/AVC coding standard [12]. 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) [3]. 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, just like the fast sub-optimal ME algorithms, few efficient HW implementations have been presented for these new ME approaches, mainly due to the inherent computational complexity of their operation. One example of a highly efficient IP core of an Application Specific Instruction Set Processor (ASIP) that is capable of implementing any of these complex ME algorithms has been recently proposed [9].

Nevertheless, despite the considered application sce-

978-0-7695-3277-6/08 $25.00 © 2008 IEEE
DOI 10.1109/DSD.2008.66

nario, the availability of several feasible implementation technologies to implement these IP cores has devised a growing need to compare and assess such alternatives in what concerns the resulting implementation performances. As an example, with the advent of the most recent generations of FPGAs, it has been proved that these devices can be regarded as feasible alternatives to other faster but more costly implementation platforms, such as the ASICs [6].

In this paper, it is presented a performance analysis of two distinct implementations of the programmable ME co-processor IP core based on the architecture recently proposed in [9]. Although other faster but rather more expensive alternatives are currently available, this comparison considers two distinct implementation technologies that represent the current *best value for money* compromise in terms of the implementation cost and of the resulting performance levels: 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. Besides this technology assessment, some significant functional improvements to the base architecture will be also proposed, as well as a detailed description of the interfacing mechanisms between the implemented co-processor and the main processing unit of the video encoding system. A brief presentation of the introduced improvements on the implemented development tools, as well as their integration into the Eclipse framework [2] will be also presented. Such tools represent an important contribution to maximize the programmers' productivity when using this co-processor.

2. Motion estimator architecture

The programmable and specialized architecture for ME proposed in [9] 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 MPEG-4 and H.264/AVC video standards.

The offered flexibility is attained by adopting the simple and efficient micro-architecture, illustrated in Fig. 1(a), whose modular structure is composed by optimized units that support a minimum and specialized instruction set. This data-path is also developed around a specialized arithmetic unit that efficiently computes the Sum of Absolute Differences (SAD) similarity function. Furthermore, a quite simple and hardwired control unit is used to generate all the required control signals [9].

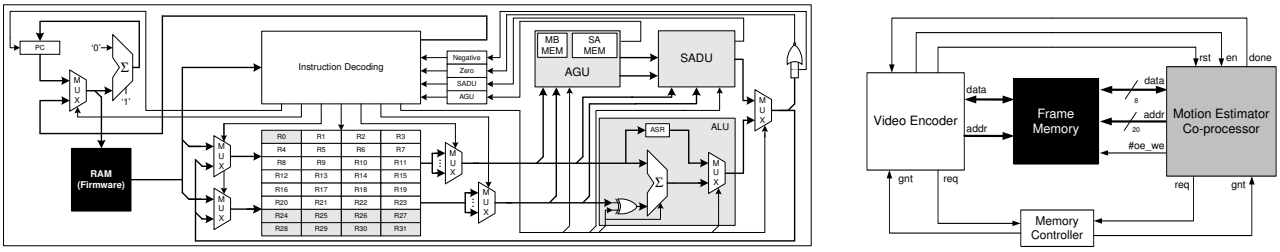
2.1. Base architecture

The Instruction Set Architecture (ISA) of the ASIP proposed in [9] was designed to meet the requirements of most ME algorithms, including some recent approaches that adopt irregular and random 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:

- `J` - *control operation*, to change the execution-flow of a program, by updating the program counter with an immediate value that corresponds to an effective address;
- `MOVR` - *data transfer operation*, to store the content of a given register in another target register;
- `MOVC` - *data transfer operation*, to store an 8-bit immediate value in the upper or lower byte of a target register;
- `SAD16` - *graphic operation*, to compute the SAD value considering two sets of 16 pixels and to accumulate the result in a target register;
- `ADD` - *arithmetic operation*, to add the contents of two registers;
- `SUB` - *arithmetic operation*, to subtract the contents of two registers;
- `DIV2` - *arithmetic operation*, to evaluate the integer division by 2 of the contents of a given register;
- `LD` - *memory data transfer operation*, to load the pixels data into two fast and small scratch-pad local memories.

These instructions directly operate the values stored in a register file composed by 24 General Purpose Registers (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. The second group, consisting of registers R16-R23, can only be accessed using two specific instructions: `MOVR`, which allows data exchange between the two register 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.

The processor data-path, depicted in Fig. 1(a), includes two specialized units to increase the efficiency of the most complex and specific operations: an Address Generation Unit (AGU) and a SAD Unit (SADU). Each of these units can directly access the SPRs. The `LD` operation is efficiently executed by the dedicated AGU, which is capable of fetching all the pixels of either the reference MB or of the corresponding search area. On the other hand, the `SAD16` graphic instruction is implemented by the SADU. In the architecture that is now considered, the implemented SADU adopts a serial structure, that restricts the required HW at the cost of requiring more clock cycles to compute this operation. Nevertheless, a complete parallel implementation of the SADU could equally be considered, leading to a faster execution time with an added cost of requiring more HW resources. Further and more detailed information about this processor architecture can be found in [9].



(a) Base architecture (SPRs are shaded in the register file).

(b) Interface with the video encoding system.

Figure 1. Proposed application specific programmable IP core (ASIP).

2.2. Functional improvements to base architecture

To maximize the efficiency of the implemented co-processor, some significant functional improvements were introduced in the base micro-architecture. In particular, the status register that stores the flags reflecting the current state of the processor and that allow to control its execution-flow was modified. In fact, according to the base architecture proposed in [9], the *Negative* and *Zero* flags of this status register are updated by the `ADD`, `SUB`, `DIV2` and `SAD16` instructions, in order to provide some extra information about their outcome. Such information can then be used by the jump (`J`) instruction to conditionally change the control-flow of a program. Besides these flags, two extra flags were introduced in the presented co-processor implementation, in order to reduce the Clocks per Instruction (CPI) performance metric achieved by this architecture: the *AGU* and *SADU* flags. These two new flags are represented in the processor block-diagram depicted in Fig. 1(a) and provide the following information:

- The *AGU flag* reports the conclusion of a transfer operation of pixel data from the external frame memory into an internal scratch-pad local memory, triggered by the last `LD` instruction. This particular feature allows the programs to be highly optimized: by using this status information in conditional jump instructions, it allows 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, thus allowing the AGU to work in parallel with the remaining functional units.
- The *SADU flag* indicates that the result of the last computed `SAD16` operation is a minimum SAD value. With this feature, efficient early-stopping mechanisms can be easily implemented without any additional arithmetic operations, which significantly increases the performance of the ME algorithms.

3. Integration with the video encoding system

To embed the 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 implemented programmable and configurable architecture for ME presents a simple and reduced pin count interface, as it can be seen in Fig. 1(b). 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 a reference macroblock and of a 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. 1(b). 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 the pixel values used 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 the 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). This *data* port is also used to transfer the result of the ME operation to the video encoder, 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 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. 2. Consequently, the number of required I/O connections is minimized.

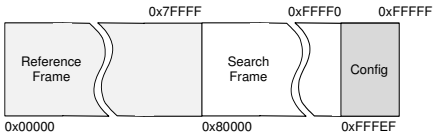


Figure 2. Memory map of the proposed ASIP.

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.

Lastly, the *en* input port is used to control the co-processor operation, while the *rst* input port is used to set the co-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 this programmable architecture and the video encoder is carried out 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. 3. 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 co-processor is not very often executed.

The second task consists in all data transfers concerning not only the pixels of a given reference MB and of its 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. A protocol entirely similar to the one described above is used to support this task, as depicted in Fig. 4. This task occurs on demand by the co-processor and is controlled by the AGU. This unit must firstly generate all 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 MB, or of its corresponding 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 and it is controlled by its main control unit. The co-processor starts the output operation by requesting the bus ownership, as it can be seen in Fig. 5. 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. Improved integrated development tool

Each programmed ME firmware of the implemented ASIP can be developed using a custom toolchain. Such programming environment was integrated into the Eclipse framework [2] with a plug-in, thus maximizing the programmers' productivity when developing the firmware. This environment was specially designed for the proposed ISA and consists of three different programming tools: a symbolic assembler (`amep-elf-as`), a linker (`amep-elf-ld`) and a cycle-based accurate simulator (`amep-elf-sim`).

The assembler compiles the source files, written using the co-processor's ISA (see section 2.1) and a syntax very similar to the one adopted by the GNU `as`, and generates the corresponding object code files in the ELF format. The `amep-elf-as` tool also has the capability to validate the syntax and semantics of the assembly instructions, to manage comments and symbols, that can either refer to labels or linked values, and even to process macros. This feature allows the development of firmware modules faster and better structured, since the original ISA does not support subroutine nor function calls.

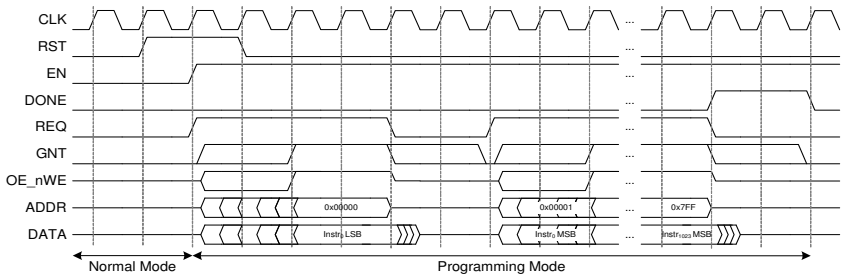


Figure 3. Temporal diagram concerning the loading of the firmware into the proposed ASIP.

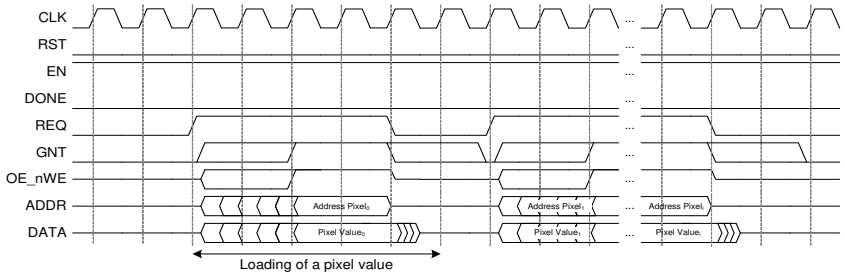


Figure 4. Temporal diagram concerning the loading of MB/search area pixels into the proposed ASIP.

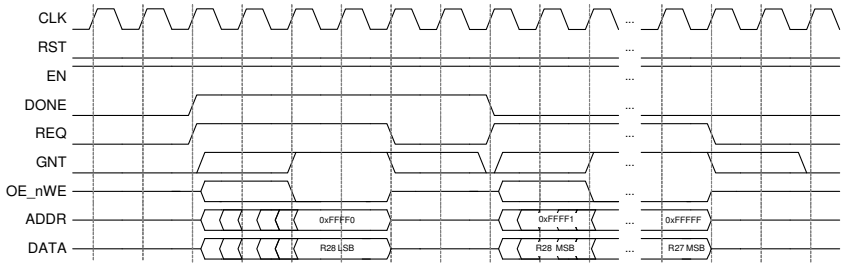


Figure 5. Temporal diagram concerning the output of the result of a ME operation.

The `amep-elf-ld` tool generates a file containing the firmware for the proposed micro-architecture, i.e., the executable file to be stored in the program memory of the ASIP. Such file is obtained by relocating the machine code instructions of the object code file provided by the `amep-elf-as` assembler to memory base address `0x0`. For more complex ME algorithms, where the algorithm implementation consists of several different source assembly files, the developed linker is also capable of linking all the corresponding object files into a single executable file and of resolving all the crossed references to the corresponding symbols within the object files. This procedure may involve the retargeting of *jump* instructions, since in the proposed architecture these instructions always refer to absolute memory address values.

The operation of the proposed ASIP for a given firmware can be validated using the `amep-elf-sim` simulation tool. The simulations are executed in a cycle by cycle basis and allow the profiling of the implemented ME algorithms, by providing the required mechanisms to count the number of clock cycles required to execute the whole algorithm, or some parts of it. Moreover, the developed simulation tool provides the most commonly adopted debug techniques, namely the use of breakpoints and of the `step`, `run until` and `continue` execution commands. The simulator also allows to examine the three memory banks of the ASIP, i.e., the program memory and the current MB and search area memories, as well as all the registers of the processor: the GPRs and the SPRs banks, the flags and the program counter.

5. Prototyping platform

To validate the functionality of the ME programmable architecture in a practical realization, a complete video encoding system was developed and implemented. The base configuration of the encoding system 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 IP core processor, 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 with the other operations, the several MVs that are required by the encoder to implement the temporal prediction mechanism. Two different

platforms were considered for prototyping the ME core, in order to evaluate its performance when implemented using both FPGA and ASIC target technologies.

The implemented video encoder consists of a software implementation of the H.264/AVC standard [12]. Such implementation includes some optimizations of the JM H.264/AVC Reference Software [1], in order to make its use more efficient in embedded systems. The modifications include the redesign of all functions used in ME, the declaration of all variables in these functions using the prefix *register*, so as to optimize the program execution time and the adaptation of the memory allocation mechanism in order to convert all dynamic allocations to static memory allocations. To maximize its performance, 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 MB and of its corresponding search area.

5.1. FPGA based prototype

The implementation of the video encoding system using an FPGA device was realized using a Xilinx ML310 development platform [14], which includes a 100MHz 256MB DDR memory bank and a Virtex-II Pro XC2VP30 FPGA device from Xilinx. This FPGA offers a significant set of implementation resources, two Power-PC processors, several Block RAM (BRAM) modules and high speed on-chip bus-communication links.

The programmable architecture for ME is implemented using the configurable logic blocks provided by this FPGA, 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.264 video encoder [1], 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, the *me* section of the application was located in a 128kB FPGA BRAM module, while the *text*, *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.

5.2. ASIC based prototype

The implementation of the ME co-processor in an ASIC was carried out in conjuction with an AT91SAM9263-EK evaluation kit [5] from ATMEL. This development board includes an AT91SAM9263 micro-controller, based on the ARM926EJ-S processor, widely adopted by the latest generation of mobile phones and PDAs. This board also has an extensive set of peripherals for control, communication and data storage purposes. Such 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 SDRAM memory bank. In addition, this development board offers the possibility to easily embed user-developed peripherals, by making available some connectors to the External Bus Interface (EBI) of the processor.

Hence, the main processing unit of the video encoder was implemented in the AT91SAM9263 processor, operating at 99.33MHz. Just as the FPGA prototyping system, such processing unit runs the optimized software implementation of the H.264 video encoder [1]. In this prototyping platform, all program code and data sections of the application are located in the 64MB SDRAM memory bank. An expansion slot is used to connect the AT91SAM9263-EK prototyping platform to a daughter board, with the ASIC implementation of the ME co-processor, by making use of the processor's EBI. This EBI is used to exchange the control signals and all the required data between the processor and the ME co-processor.

6. Implementation and experimental results

The performance analysis of the FPGA and ASIC prototypes of the programmable ME IP core was realized for a specific configuration of this parameterizable structure. The considered setup adopted a simplified AGU that does not allow data re-usage and a power efficient serial processing structure for the SADU module. Such architecture was selected as the result of a compromise between the amount of required HW resources, the circuit power consumption and its usability for real-time operation. Previous research work 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. However, depending on the target application, the considered architecture can be reconfigured to use other AGU and SADU modules that represent different compromises.

6.1. FPGA implementation

The video encoding system described in section 5.1 was implemented using the EDK 9.1i and ISE 9.1i tools from Xilinx. The implementation layout of the whole video encoding system in this FPGA is presented in Fig. 6. Table 1 presents the implementation results obtained for the ME co-processor IP core. These results evidence that FPGA based implementations of the considered ME architecture allow a maximum operating frequency of about 100 MHz. They also show that very few HW resources (about 6k equivalent logic gates) are required to implement the ME co-processor

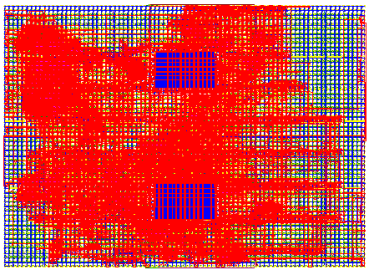


Figure 6. Implemented prototype of the ME ASIP in a Virtex-II Pro XC2VP30 FPGA device.

Table 1. Implementation results of the motion estimator using the Virtex-II Pro XC2VP30 FPGA device.

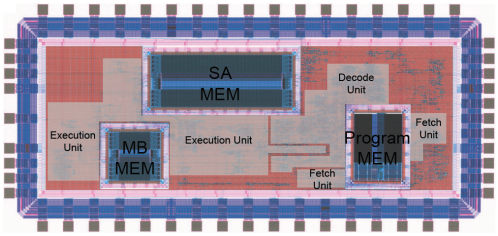
Occupied Slices	811 (5%)
Occupied LUTs	1235 (4%)
Estimated Equivalent Logic Gates	6 kGates
Occupied BRAMs	4 (2%)
Maximum operating frequency	100.30 MHz

in an FPGA device. In fact, the ME co-processor uses only 20% of the total slices required to implement the whole video coding system, that can operate at a maximum frequency of 62 MHz. Such operating frequency limits the performance of the ME co-processor and is due to the extra hardware resources required to implement all the system peripherals (i.e., an USART, that allows communication with the encoder; a timer module, to evaluate the performance of the encoder and of the ME task; and BRAM local memories) and its interconnections with the ME co-processor and the Power-PC processor.

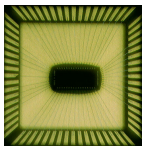
The functionality of the implemented video coding system was successfully verified by encoding a set of benchmark QCIF video sequences, with quite different characteristics in terms of movement and spacial detail, and by using several different ME algorithms. The adopted encoding used the typical set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

6.2. ASIC implementation

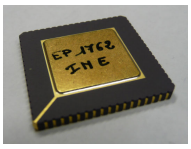
The video encoding system described in section 5.2 was implemented with the GNU toolchain for the ARM architecture, targeting the AT91SAM9263-EK evaluation kit connected to the expansion board with the ASIC implementation of the ME co-processor. This ASIC adopted the same configuration as the one used for the FPGA implementation. In addition, it also included a complete set of testing structures, that can be accessed by means of an integrated JTAG controller. The circuit, whose layout is presented in Fig. 7, was manufactured under the *mini*@SIC



(a) ASIC layout.



(b) Wire bonding.



(c) Packaged chip.

Figure 7. Implemented prototype of the ME ASIP in an ASIC based on UMC 0.18μm CMOS process.

Table 2. 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

program from EUROPRACTICE, using a StdCell library based on a 0.18μm CMOS process with 1 poly and 6 metal layers from UMC (UMC L180 1P6M MM/RFCMOS) [10]. Table 2 presents the obtained implementation results, with $V_{dd} = 1.8V$. These results 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 encoder was verified using the same methodology as the one adopted for the FPGA implementation and proved to allow the real-time computation of MVs up to the CIF image format.

6.3. Comparison analysis

The performance results presented in Table 2 for the ME ASIC are quite similar to those obtained with the FPGA implementation, presented in Table 1. However, the results obtained for the implementation of the whole video encoding system show that the FPGA implementation provides a lower operating frequency than the ASIC implementation:

Table 3. Comparison of the performances of the two considered implementations of the ME co-processor.

ME Algorithm	Average #Clk/MB	Max. frame-rate [fps]			
		QCIF		CIF	
		FPGA	ASIC	FPGA	ASIC
FSBM	75922	8.29	13.30	2.07	3.33
3SS	7453	84.49	135.52	21.12	33.88
DS	10588	56.68	94.12	14.67	23.53

66 MHz *vs* 100 MHz. This decrease in the operating frequency of the FPGA implementation, which is owed to the embedding of the PowerPC processor and of the remaining peripherals that compose the system, causes a slight degradation in the performance of the video encoding system, as depicted in Table 3. Nevertheless, the considered ME systems still allow the estimation of MVs in real-time for the QCIF and CIF image formats (e.g.: when the 3SS algorithm is adopted). In fact, better performance levels can be achieved by using different SADU architectures. For example, by using a fully parallel architecture for the SADU, it is possible to estimate MVs in real-time for images up to 4CIF resolution.

Furthermore, the FPGA implementation of the ME co-processor also presents increased 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 and/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, this implementation can be regarded as a suitable alternative for video encoding applications running on portable and mobile devices [6].

On the other hand, when battery-supplied devices are considered, different requirements must be taken into account. For such cases, where power consumption is a mandatory requirement, the implemented ASIC ME circuit clearly evidences its suitability to efficiently implement ME algorithms, by requiring only 31 mW when operating at 100 MHz.

7. Conclusions

This paper presents a performance analysis of two distinct implementations of a recently proposed high performance programmable and specialized architecture for ME. The 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 and DS) 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) up to the CIF image format. 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.

References

[1] *JM H.264/AVC Reference Software - version 13.2*. http://iphome.hhi.de/suehring/tml/, 2007.

[2] *Eclipse framework*. http://www.eclipse.org, 2008.

[3] I. Ahmad, W. Zheng, J. Luo, and M. Liou. A fast adaptive motion estimation algorithm. *IEEE Transactions on Circuits and Systems for Video Technology*, 16(3):439–446, Mar. 2006.

[4] 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, Dec. 2006.

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

[6] A. Berić, R.Sethuraman, H. Peters, J. L. van Meerbergen, G. de Haan, and C. A. A. Pinto. A 27 mW 1.1 mm^2 motion estimator for picture-rate up-converter. In *Proc. of the 17^{th} Int. Conf. on VLSI Design - VLSI'04*, pages 1083–1088, 2004.

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

[8] 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.

[9] T. Dias, S. Momcilovic, N. Roma, and L. Sousa. Adaptive motion estimation processor for autonomous video devices. *EURASIP Journal on Embedded Systems - Special Issue on Embedded Systems for Portable and Mobile Video Platforms*, (57234):1–10, May 2007.

[10] Faraday Techn. Corp. *Faraday ASIC Cell Library FSA0A_C $0.18\mu m$ Standard Cell (v1.0)*, August 2004.

[11] 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, Feb. 1993.

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

[13] 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.

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