DESIGN OF APPLICATION SPECIFIC INSTRUCTION-SET PROCESSOR FOR IMAGE AND VIDEO FILTERING

*S. Saponara, *L. Fanucci, +S. Marsi, +G. Ramponi, #M. Witte, #D. Kammler

*DIIEIT, University of Pisa, Italy{s.saponara, l.fanucci}@iet.unipi.it
+DEEI, University of Trieste, Italy {marsi, ramponi}@units.it
#ISS, RWTH, University of Aachen, Germany {witte, kammler}@iss.rwth-aachen.de

ABSTRACT

Two architectures for cost-effective and real-time implementation of non-linear image and video filters are presented in the paper. The first architecture is a traditional VHDL-based ASIC (Application Specific Integrated Circuit) design while the second one is an ADL (Architecture Description Language) based ASIP (Application Specific Instruction Set Processor). A system to improve the visual quality of images, based on Retinex-like algorithm, is referred as case study. First, starting from a high-level functional description the design space is explored to achieve a linearized structural C model of the algorithm with finite arithmetic precision. For the algorithm design space exploration visual and complexity criteria are adopted while a statistical analysis of typical input images drives the algorithm optimization process. The algorithm is implemented both as ASIC and ASIP solution in order to explore the trade-off between the flexibility of a software solution and the power and complexity optimization of a dedicated hardware design. The aim is to achieve the desired algorithmic functionality and timing specification at reasonable complexity and power costs. Taking advantage of the processor programmability, the flexibility of the system is increased, involving e.g. dynamic parameter adjustment and color treatment. Gate level implementation results in a 0.18µm standard-cell CMOS technology are presented for both the ASIC and ASIP approach[1].

1. INTRODUCTION

In the field of digital processing systems ASIPs are gaining ground to fill the gap between ASICs, highly optimized hardware platforms but lacking flexibility, and the solution offered by software development on DSPs (Digital Signal Processors), reusable and programmable but providing too little performance and energy-inefficiency. ASIPs are flexible in general and optimized for an application domain. This makes them more useful than ASICs for applications requiring a certain degree of programmability. Moreover, since the customization of the design is focused on the addressed application domain, they are more specialized and therefore more optimized than DSPs, being able to provide the right features in terms of timing performance, energy consumption and required area. Architecture Description Languages (ADLs) [1-3] help the processor designer by automatically generating the software tool-suite (compiler, assembler, linker, simulator) as well as the Register Transfer Level description of the processor. While designing an ASIP, the designer has the full freedom to do the trade-off between performance, flexibility and physical criteria like silicon area and power consumption. Extending the instruction set by specialized instructions is particularly beneficial for applications involving hot spot elaboration kernels, like image and video signal processing. In such a field some innovative algorithms are spreading up involving highly nonlinear operators [4-10]. By changing the data and control flow but keeping the kernel arithmetic, the same class of filters can be used for different applications. For this goal, dedicated ASICs are not suitable since they provide only very limited flexibility. However, DSP solutions are not acceptable either, because high computational performance, low energy cost and low silicon area are very important specifications in handheld and mobile scenarios. In this paper the ASIP implementation of the Retinex class of algorithms [4-9] is presented. After high-level algorithmic optimization the identification of the operation kernels and their mapping onto an instruction set are described. Some special processor concepts used to achieve a good performance vs. flexibility trade-off are detailed. Finally, CMOS synthesis results are presented with a comparison to a benchmark ASIC design.

2. RETINEX-LIKE IMAGE AND VIDEO FILTERS

In the Retinex theory, first proposed in [4], an image is expressed as the pixel-by-pixel product of the ambient illumination y and the reflectance r of the scene object. The values of the latter are determined as the pixel-by-pixel ratio between the input image and an estimate of the illumination. This way we can control independently illumination and reflectance, as example modifying the dynamic of the illumination without any modification in the details which are transmitted through the reflectance channel. It is also possible to process the reflectance signal to improve the details in the final image. Target applications include image contrast enhancement, correction of images acquired in bad lighting conditions, control of dynamic in logarithm sensors [5-10]. All these filters exploit a similar structure sketched in Fig. 1 (in case of logarithmic sensors multiplication and division are replaced by add and subtract, respectively): F is a luminance estimator while the Γ and β blocks respectively process the luminance dynamic and improve the details. We found that effective expressions are:

$$\Gamma(y) = 255 \cdot \left(\frac{y}{255}\right)^{\gamma\left(1+\frac{y}{255}\right)}, \quad \beta(r) = \frac{1}{1+e^{-b\cdot\log r}} + \frac{1}{2} \qquad (1)$$

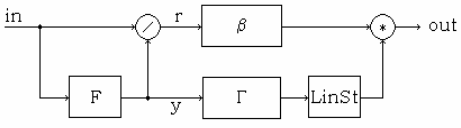

Figure 1. Block diagram of Retinex-based operators

The core of retinex-based methods is the estimation of the illumination component. As matter of fact in natural images the illumination typically changes very smoothly between contiguous pixels, with the exception of some particular cases, like the presence of luminous sources in the image, or the illumination of the scene by various light sources with abrupt transitions between them (e.g. images which represent an

[1] Work partially supported by PRIMO and NEWCOM projects.

indoor and an outdoor scene in the same frame). A good estimator of the illumination must take into account these considerations; it could be realized with an edge-preserving low-pass filter with a quite narrow band. The narrow band, which implies a wide impulse filter response, is the primary reason for the high complexity of the algorithms proposed in the literature [7, 8] and makes them unsuitable for real time applications. To avoid this drawback a new algorithm, based on Recursive Rational Filters (RRF), has been proposed by the authors in [6]. This method uses an IIR to obtain a long impulse response, while the edge preserving effect is achieved with a suitable system which controls the filter bandwidth according to the characteristics of the input signal. The recursive filter is very effective but presents the disadvantage of introducing a phase distortion in the output image, and an asymmetric response. To avoid such problems, a 2D extension of the well known time-reversal method must be used. For such reason the filter must be applied four times to each input image. During these iterations the pixels are processed along all the possible directions from top to bottom and from left to right and viceversa. It should be noticed that just two filter passes can be performed only if a fairly small impulse response is requested, i.e. if the input image is quite small. The output of the low-pass edge-preserving filter F is yielded by the function:

$$y(n,m) = \frac{S_h \cdot f_h + S_v \cdot f_v + in(n,m)}{S_h + S_v + 1}, \text{ being}$$

$$f_h = \alpha \cdot y(n-1,m) + (1-\alpha) \cdot in(n,m), S_h = \frac{10^{-2}}{10^{-5} + \left(\log_{10}\left(\frac{in(n-1,m)+1}{in(n+1,m)+1} \right) \right)^2}$$

Similar expressions are available for fv and S_v, evaluating the gradient through the m-direction (vertical). The α parameter controls the local cut-off frequency. As an example, Fig. 2 shows a portion of an image acquired in bad lighting conditions (2a). The application of the classical histogram equalization brings to the result visualized in Fig. 2b. While trying to get the image clearer, a detail blurring comes up. The Retinex algorithm, instead, permits to obtain the effect in Fig. 2c solving the problems of image contrast and brightness together.

a b c

Figure 2. a) Original, b) Histogram equalization, c) Retinex

The above algorithm refers to monochrome images while for color images several color spaces can be considered. It is not advisable to process the three color components R, G, and B separately, because of a significantly increase of the computational effort and also because hue variations are most likely to be obtained. The simplest approach is to process the luminance component, computed e.g. according to the *YUV* or the *YCbCr* standard, and eventually to modify the color components accordingly, trying to preserve the hue. An alternative could be to use the *HSV* domain; in this case the above described algorithm could be applied to *V*, which is somehow related to the luminance. However, the nonlinearities inherent in the *RGB* to *HSV* conversion, and viceversa, are most likely to yield unwanted hue or saturation variations. In order to deal with video sequences, a first idea could be to simply use, on a frame by frame basis, the algorithm described above. There is however a main issue which has to be taken into account, i.e. the high sensitivity of the human eye to temporal artifacts: if subsequent frames are independently processed, they may appear pleasant if observed one by one but yield annoying effects when visualized in a sequence; this may happen for instance if different luminance corrections are performed on subsequent frames. This issue suggests the use of temporal filtering. In particular a novel algorithm has been proposed, where the input signal is split into three different contributions according to the scheme depicted in Fig. 3. One contribution represents the background illumination (LL). This signal is temporally filtered to reduce flashing effects and abrupt temporal variations. A second contribution (LH) is devoted to the information about small light sources, like car lights, which must not be temporally filtered, otherwise very annoying artifacts will appear. This signal is combined with the previous one in the luminance signal and is processed by the Γ block to compress the dynamic. The last contribution represents the details, i.e. the reflectance (R). This signal could be amplified by a suitable block β.

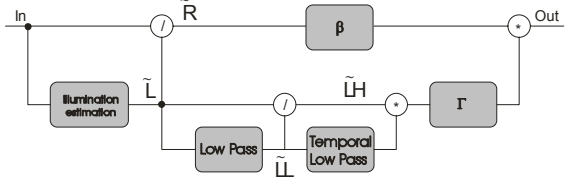

Figure 3. Retinex-based operators applied to videos

3. DESIGN OPTIMIZATION FLOW

The adopted design flow is sketched in Fig. 4. First, starting from a high-level functional description the design space is explored in a C/Matlab environment to achieve a linearized structural model, with finite arithmetic precision, of the class of algorithms described in Section 2. For this purpose some effective methodologies for bit-true arithmetic definition and linearization of non linear operators have been developed requiring some pre-fixed optimization schemes based on piecewise linear and piecewise constant (see Figs. 5 and 6 for linearization examples and implementation block diagrams).

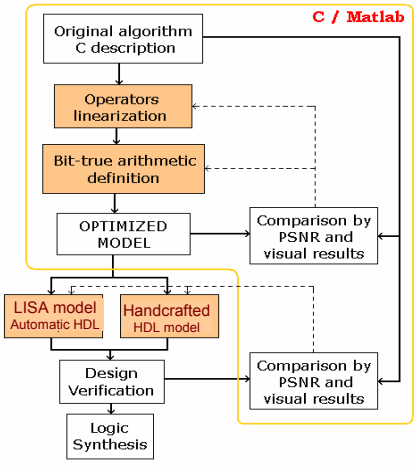

Figure 4. Optimization flow

Two criteria are used to keep a good degree of approximation: a PSNR-based objective criterion and a subjective one based on visual perception. The linear piece-wise approach allows for better quality results and is adopted for the Γ and β blocks. The constant piece-wise approach is preferred when the non linear transformation involves quantities not directly observable at the system output, like the filter F coefficients depending on S_h and S_v. In such case the resulting visual quality using constant piece-wise is the same of linear piece-wise but the former has a simpler implementation being based on Look-up-Table (LUT)

and avoiding the use of multipliers (see Fig. 5a). Starting from the linearized and bit-true algorithmic model the hardware design is addressed using two methodologies. In Fig. 4 ASIP and ASIC approaches are depicted, referring to their respective description languages: ADL and HDL. The HDL netlist of the ASIP is automatically generated from the LISA description.

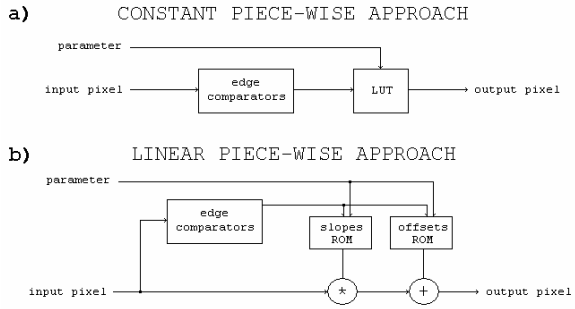

Figure 5. Block diagrams for linear and constant piece-wise

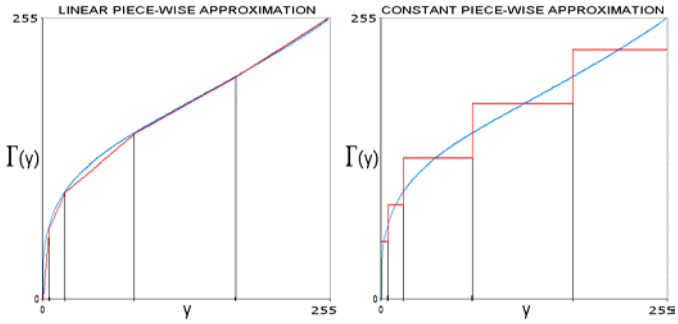

Figure 6. Linear and constant piece-wise approximation

4. ASIP DESIGN FOR RETINEX FILTER CLASS

4.1 Memory Organization

For the Retinex class of applications described in Section 2, ASIPs offer an excellent trade-off, since the most repeated application kernels can be grouped in optimized hardware units, while keeping the activation of those hardware accelerators at a software level by the definition of a suitable instruction set. Like most multimedia applications, the design of video filtering architectures is dominated by the memory size and data transfer rate. In case of video processing, it is often necessary to store more than one image since previous frames are needed for the elaboration (e.g. temporal filtering). It can be also the case that the video processing is split over several pipeline stages (by the means of frame pipelining) in order to increase the information throughput. To determine the required memory size to store the intermediate images, the memory size of one frame has to be multiplied with the number of used pipeline stages. Referring to VGA format, a worst case evaluation leads to a memory requirement of 10 Mbytes, unacceptable for systems designed for a single die. To reduce the memory amount, one way was pursued in the C /Matlab optimization step, by reducing the number of precision bits while keeping acceptable algorithmic performance. In the case study, we used 8 integer bits and 6 fractional bits for data representation. Lowering the number of fractional bits below the found optimum value can lead to a great worsening of the algorithmic performance. Another effective way to reduce memory is to remove the pipelining at a frame level. This solution is based on a re-utilization of the same memory to store the intermediate data concerning the partially elaborated frames. The main drawback is, of course, the throughput reduction, which is a critical specification item. Because of the trade-off between memory resources and data throughput, we decided to use two frame memories. This solution allows to keep a slight parallelism in the elaboration, since it is possible, for instance, performing the Γ and β transformations at the same time, without increasing memory requirements too much compared to the simplest solution involving a single frame memory. The total required memory for VGA format processing is 1.03 Mbytes. Moreover, there is a highly effective methodology to improve timing performance keeping the benefits of this memory organization. This is achieved by re-introducing a pipelining of the elaboration moving it from the frame level to the pixel level, which is more efficient in terms of memory usage. That allows for parallel elaboration of several pixels making the architecture timing efficient as well. Entering in more details about memory architecture implementation, Synchronous SRAM memories have been used for data storage. The two RAMs have been named X RAM and Y RAM. They are read scanning the whole image in order to produce the illumination component (y) according to the F filter functionality. This process requires four passes of the whole image and the intermediate results are stored in the Y RAM, while the X RAM contains the input image. After that both RAMs are further scanned and the reflectance component (r) evaluation is performed by division. Also the Γ and β transformations are performed. Then the Γ output is stored in the Y RAM, while the β output is stored in X RAM. At the end, a further scan of the two RAMs is required for the component recombination leading to the output image, which is finally stored in the X RAM. In all frame processing stages, a pipelining of subsequent pixels is used to speed up the architecture. Both, the particular memory organization and the data pipelining are important hardware customizations applying to the case study application. These sorts of customizations of memory and pipeline architecture are major advantages of ASIPs. Other resources utilized in the processor arithmetic can be customized according to the application needs, too. In the case study, 16 general purpose 32-bit registers have been instantiated. Some additional dedicated registers have been used for the storage of processing parameters which can be easily used during the elaboration. 14-bit fixed point arithmetic has been used for data representation whereas instructions have been coded using 32-bit words.

4.2 Pipelined Architecture and By-pass Mechanism

The pixel elaboration has been split over a pipelined architecture. This choice has the benefit of increasing the architecture parallelism and to shorten the critical path. This property of pipelined systems leads to an increased data throughput, which is highly desirable in our case. However, this strategy can have some drawbacks due to increased latencies, silicon area overhead, e.g. by pipeline control and registers, and dependencies in the pipeline. Data dependencies can exist between neighbored instructions, that is, a result produced by an instruction may be used as an operand by the following instructions. Such situations might require pipeline interlocking mechanisms. Using ADLs, the design space is fully explorable with no restriction given by pre-designed parts or templates. Nevertheless, templates can be used as a first starting point, but the designer is not limited by that. Customizations of the pipeline structure and the memory architecture are presented in the following. In the case study, seven pipeline stages have been introduced. This pipeline organization resulted from the design space exploration as the best trade-off between throughput and complexity. Particularly, to understand why such a pipeline structure has been used we have to refer to a repeated optimization technique used all over the design: the piecewise approximation of non linear operators. Since this is a widely utilized functional kernel in the optimized application, some particular attention was paid to its implementation. As example, let us consider the piecewise

linear technique used to approximate Γ and β transformations. Since the throughput is a pressing specification, it is desirable having an instruction able to load an operand from the data memory, to perform the Γ or β transformation in the piecewise linear form and to store the result back to the data memory. The designed pipeline allows for the processing of such an instruction, using the following stages (see Fig. 7):

- FE: the fetch stage in which the instruction is fetched from the program memory.
- DC: the decode stage in which the instruction is decoded, producing the control signals for the operating part.
- LD: the load stage in which the operand is loaded from the data memory.
- CMP: the comparison stage where the loaded operand is compared to the edges on the abscissa axis in order to identify the correct approximation interval.
- ROM: stage in which the result of the previous comparison is used to address a ROM storing the parameters (offset *Q*, slope *K*) of the correct piecewise segment.
- ARITH: the arithmetical stage in which the fetched parameters are used to calculate the output according to piecewise segment expression $K \times IN + Q$.
- WB: the write-back stage in which the output is stored back to the data memory.

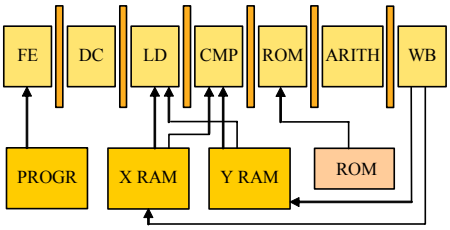

Figure 7. Pipeline and memory organizations for the ASIP

The names assigned to each stage are mnemonical names applying to the presented particular case. Depending on the instruction, different operations can be executed in the stages, meaning e.g. the ROM stage is not used for ROM accesses exclusively. The piecewise linear approach allows also for the implementation of the division operation with a throughput of one division per cycle, which is a great advantage for the system performance. The performed division is a customized operation leading to acceptable results only operating on inputs in the working range. Otherwise the approximation introduced by our procedure would compromise the result. Nevertheless the LUT technique used for our customized division shows satisfactory results. In the division instruction, the LD stage is used to load the denominator from the data memory, the CMP stage is used to load the numerator and the WB stage to write the computed ratio back to the data memory.

Data dependencies are a problem related to the pipeline architecture. A 7-stage pipeline obviously leads to the following disadvantage of data hazards: in the LD stage an instruction ("consumer") may read from a shared storage (a general purpose register or a memory location), which is expected to be written by a previous instruction ("producer"). If the producer instruction has not yet reached the WB stage in which the final result is stored in the shared storage, the consumer instruction will load an outdated value. That will cause a completely wrong result. There are two standard solutions for this issue: pipeline interlocking and bypassing. Using interlocking, the instructions trying to access data, that has not yet been written back, causes the pipeline to be stalled partially. This causes unacceptable throughput degradation, especially in performance critical loops. This drawback can be solved by instruction rescheduling – either by the processor or by the compiler. This approach is usually strongly limited by the data and control flow. A more efficient way of resolving the data dependencies is to implement bypasses. Bypasses forward data immediately from a pipeline stage back to a previous stage. In the case study the majority of the instructions can provide the final result not before the ARITH stage. Therefore, two kinds of bypasses were implemented depending on the starting point of the bypass path: bypasses from the ARITH stage or bypasses from the WB stage. In both cases, more than one path was implemented depending on the end point of the bypass. They include: bypasses to the LD stage, the CMP stage, the ROM stage and the ARITH stage (Fig. 8). Most of the implemented bypasses are extensively used e.g. in the non linear filter F (Fig. 1), which implements one of the key elaboration steps of our case application.

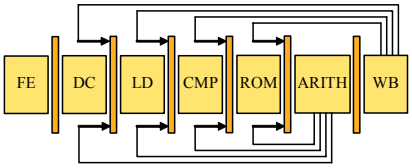

Figure 8. Implemented by-pass mechanisms

4.3 Customized Instruction Set

One of the most important advantages of ASIPs is the fact that the instruction set can be customized according to the requirements of the application. This enables a trade-off between computational performance, silicon area and energy consumption. In order to increase the architecture efficiency it can be beneficial to implement complex pipelined instructions. This shortens the length of the final assembly program that is in our case study strictly related to the number of clock cycles needed for the complete elaboration. Since the specific scenario is image/video processing, it is important to notice that there will be a portion of the assembly program (referred to as main loop) that has to be repeated a large number of times according to the image size (one iteration per pixel), typically in the order of hundreds of thousands of times. That means that a particular attention has to be paid to the number of program lines setting up the main loop, in order to avoid any waste of cycles and to maximize the throughput. In particular we show this for our case study in the following after giving a list of the most important instruction set customizations: Single instruction non linear transformations, Automatic address calculation and Zero overhead loops.

For example, considering the address generation for the data memory, from the algorithmic specifications it can be noticed that some pre-fixed patterns are established iterating over the image. Thus an Address Generation Unit (AGU) calculating the next address for the data memory by incrementing the pixel pointer can be implemented in hardware. This is reflected in the syntax of several instructions by a short extension. Thus the address update is performed in parallel without the need of wasting cycles just to do the data address update. Another observation is, that in conventional loop implementations comparisons and conditional branches create a significant instruction overhead and, even worse, cause pipeline control hazards. They lead to pipeline stalls and flushes. These problems can be avoided by implementing a loop mechanism in hardware. This is possible for loops being executed a pre-calculated number of times (equal to the image size). In this case it is enough to have a loop-parameter initialization before entering the loop and to manage the loop jumps by the hardware. This technique is known as zero-overhead loop implementation. With these implementation strategies, the programming is made easier and pipeline stalls and flushes resulting from control hazards can be eliminated. The designed Instruction Set includes 42 instructions categorized in the

following groups: non linear transformations (9), arithmetical computations (11), space colour conversions (6), memory accesses (9), processor initialization (6) and loop control (1).

5. CMOS SYNTHESIS AND PERFORMANCE

5.1 ASIC benchmark

The ASIC benchmark has been obtained by handcrafted translation of the linearized algorithm bit-true model (14-bit fixed point arithmetic) into a register transfer level VHDL description. The architecture closely follows the structure in Fig. 1 plus dedicated units for *RGB* from/to *YCrCb* conversion. The memory organization is the same described for the ASIP. The requirements in terms of SRAM data memory and number of clock cycles for the ASIC and ASIP units are listed in Table 1 for different formats. The data refer to the implementation of the Retinex algorithm applied to coloured images in the *YCrCb* space and with four passes of the filter F in Fig. 1.

Format	cycles·10^6 ASIP	cycles·10^6 ASIC	Data SRAM ASIP/ASIC
QCIF (176x144)	1.3	2.4	88704 bytes
SIF (352x240)	4.5	8	295680 bytes
VGA (640x480)	16.3	30	1075200 bytes

Table 1. Cycles and RAM required to process different formats

5.2 Synthesis Results and Comparisons

The HDL description of the ASIP, generated by LisaTEK tool starting from the ADL design described in Section 4, and that of the ASIC have been synthesized with Synopsys tool in a 0.18 μm CMOS standard-cells library at 1.55 V supply voltage. The designed macrocells have been mapped on prototyping boards, based on Xilinx Virtex FPGA technology. Since the speed of the FPGA emulation is much higher than the speed of any HDL simulation this enabled to carry out complete life demonstration of the effects introduced by the algorithm on images. The algorithmic performance for ASIC and ASIP are comparable with a PSNR vs. the original non-linear and infinite-precision model of the algorithm higher than 30 dB. As example, the PSNR for the Swan image in Fig. 2 is 30.7 dB for the ASIP and 30.8 dB for the ASIC. The difference in terms of visual subjective quality is negligible.

The logic synthesis of the ASIP processing core results in a complexity of 96 kgates plus 18544 bytes of ROM to implement LUT-based operators. An instruction SRAM of 1 kByte is used: its size is enough to support the different possible algorithms described in Section 2 belonging to the Retinex-filtering class; for the color application referred in Table 1 roughly 400 bytes are needed. The max. clock frequency is 100 MHz corresponding to a max. throughput of about $1.9 \cdot 10^6$ pixels/s. Hence the ASIP can be used to process in 1 s very large still images (e.g. SXGA, WXGA) and allows for real-time SIF videos up to 22 Hz. The ASIC processing core resulted in a complexity of 52.9 kgates plus 7528 bytes of ROM to implement LUT-based operators. The max. clock frequency is 130 MHz for a throughput of $1.4 \cdot 10^6$ pixels/s. As for the ASIP this allows the processing in 1 s of very large still images whereas real-time SIF videos are supported up to 16 Hz. For both ASIC and ASIP a speedup factor of 1.8 can be achieved by performing only two passes of the filter F in Fig. 1 instead of four. In such case up to 39 Hz SIF and 29 Hz SIF videos can be processed in real-time by the ASIP and ASIC, respectively. The visual quality reduction is negligible for small formats (e.g. fractions of dB of PSNR reduction for QCIF) whereas it becomes visible for SIF formats or larger. The above synthesis results demonstrate that the ASIP paradigm is a promising solution to achieve comparable ASIC performance but for a higher flexibility. ASIP and ASIC require the same data memory complexity; the ASIP processing core is bigger, in terms of logic gates, but is faster, in terms of throughput. This is a satisfactory result considering that we moved to a programmable architecture, opposed to a dedicated one, and that we were able to make the processor flexible but also efficient enough to allow for the outer control of the elaboration parameters, of the output dynamic and for the processing of coloured images represented in *RGB*, *HLS*, *HSV*, *YCrCb* or *YUV* spaces whereas the ASIC version is limited to *RGB* and *YCrCb*. Moreover, the ASIP can implement video processing based on (i) the frame by frame repetition of the filtering structure in Fig. 1 and (ii) the algorithm in Fig. 3 using temporal filtering. The ASIC instead is designed to support only the first option and, as discussed in Section 2, this can cause the appearance of annoying artifacts if different luminance corrections are performed on subsequent frames. Finally the ASIP is designed with ADL at a higher abstraction level thus making development and design space exploration more efficient, including the automatically generation of synthesizable and competitive VHDL code.

6. CONCLUSION

Two designs suitable for the cost-effective and real-time implementation of Retinex-like non-linear image and video filters are presented in the paper. The design process is splitted over two hierarchical optimization steps at algorithmic and architectural levels. The main considerations leading to the designed ASIP architecture are listed, from the memory organization to the architecture pipelining and to the further customization of the architecture by the addition of some hardware features like bypasses, AGU and special structures for hardware looping. During the whole design the basic idea of the Instruction Set is kept in mind as a guide for hardware design. The ASIP design by ADL is compared to dedicated VHDL implementation. Synthesis results on CMOS 0.18 μm technology demonstrate that the ASIP paradigm is a promising solution to achieve comparable ASIC performance but for a higher flexibility, reusability and design efficiency.

REFERENCES

[1] H. Peters et al., Application specific instruction-set processor template for motion estimation in video applications, *IEEE Trans. Circuits and System for Video Tech.*, vol. 15, April 2005, pp. 508-527

[2] A. Hoffmann et al., *Architecture Exploration for Embedded Processors with LISA*, Kluwer Academic, 2002

[3] O. Schliebusch et al., A framework for automated and optimized ASIP implementation supporting multiple hardware description languages, *IEEE ASP-DAC05*

[4] E. Land, J. McCann, Lightness and retinex theory, *Journ. of the Opt. Soc. of America*, vol. 61, pp. 1-11, 1971

[5] G. Orsini et al, A modified retinex for image contrast enhancement and dynamic control, *IEEE ICIP03*, pp.393-396

[6] S. Marsi et al., Image contrast enhancement using a recursive rational filter, *IEEE IST04*, Stresa, pp. 29-34

[7] M. Ogata, T. Tsuchiya, T. Kubozono, K. Ueda, Dynamic range compression based on illumination compensation, *IEEE Trans. Cons. Electr.,* vol. 47, n.3, pp. 548-558, 2001

[8] D.J. Jobson, Z. Rahman, G.A. Woodell, Properties and performance of a center/surround Retinex, *IEEE Trans. on Image Process.*, vol.6, no.3, pp. 451-462, March 1997

[9] S. Saponara et al., Cost-effective VLSI design of non linear image processing filters, *IEEE DSD05*, pp. 322-329

[10] R. Fattal et al., Gradient domain high dynamic range compression, *ACM Trans. on Graphics*, 2002, pp.249–256