A NOVEL SCALABLE DEBLOCKING FILTER ARCHITECTURE FOR H.264/AVC AND SVC VIDEO CODECS

T. Cervero[1], A. Otero[2], S. López[1], E. De La Torre[2], G. Callicó[1], R. Sarmiento[1], T. Riesgo[2]
[1]IUMA, Universidad de Las Palmas de Gran Canaria, Spain
[2]CEI, Universidad Politécnica de Madrid, Spain
{tcervero, seblopez, gustavo, roberto}@iuma.ulpgc.es; {andres.otero, eduardo.delatorre, teresa.riesgo}@upm.es

ABSTRACT

A highly parallel and scalable Deblocking Filter (DF) hardware architecture for H.264/AVC and SVC video codecs is presented in this paper. The proposed architecture mainly consists on a coarse grain systolic array obtained by replicating a unique and homogeneous Functional Unit (FU), in which a whole Deblocking-Filter unit is implemented. The proposal is also based on a novel macroblock-level parallelization strategy of the filtering algorithm which improves the final performance by exploiting specific data dependences. This way communication overhead is reduced and a more intensive parallelism in comparison with the existing state-of-the-art solutions is obtained. Furthermore, the architecture is completely flexible, since the level of parallelism can be changed, according to the application requirements. The design has been implemented in a Virtex-5 FPGA, and it allows filtering 4CIF (704x576 pixels @30fps) video sequences in real-time at frequencies lower than 10.16 Mhz.

Index Terms— H.264/AVC, SVC, deblocking-filter, FPGA, parallelism and scalability.

1. INTRODUCTION

Video coding applications have changed a lot during the last decade. They tend towards systems with higher levels of quality, flexibility and, at the same time, with higher compression efficiency. However, these improvements come at the price of increasing the computational requirements of emerging video applications.

The cooperation between the Joint Video Team of the ITU and ISO/IEC standardization organizations in 2003 gave as a result the development of the H.264/AVC video coding standard [1]. It provides better characteristics than its predecessors due to the new coding tools. More recently, both organizations have joined their efforts again in order to develop a scalable extension of the aforementioned H.264/AVC standard, named Scalable Video Coding (SVC) [2].

This work is supported by the Spanish Ministry of Science and Innovation, and European Union (FEDER funds) as part of the I+D+I Plan 2008-2011.

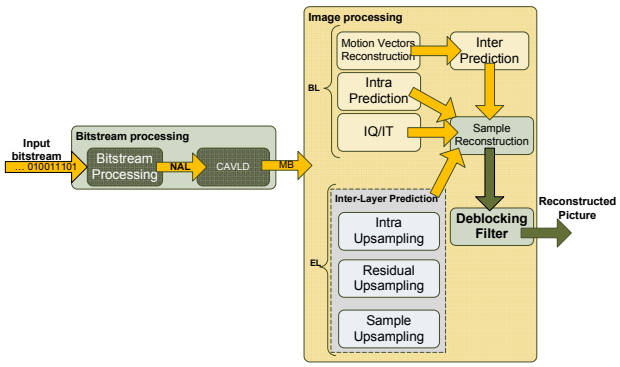

Fig. 1 SVC Decoder Diagram Blocks

SVC enables the possibility of decoding partial video bitstreams to provide video services with lower temporal or spatial resolutions or reduced fidelity while keeping reconstruction quality. Hence, this standard provides features such as graceful degradation in lossy transmission environments as well as bit rate, format and power adaptation [3]. The SVC bitstream has a hierarchical structure divided into layers, where the base layer must be H.264 compliant, while the enhancement layers include extra information related with temporal, spatial or quality scalability. Fig.1 shows a schematic view of all the functional modules involved in the decoding loop, separating the base layer (BL) modules from the enhancement layer (EL) ones.

Unfortunately, the better performance of H.264/AVC and SVC standards, the higher computational complexity they demand in comparison with previous standards. This makes the implementation of real-time video codecs exclusively based on sequential software running on embedded CPUs almost unfeasible. Hence, the current trend to deal with this emerging complexity is to parallelize the processing algorithms, and afterwards, take advantage from multiprocessor systems, GPUs or hardware architectures to implement them.

As it has been shown in different implementation profiles, like the one in [4], DF represents a critical task in H.264/AVC video encoders and decoders The importance of this coding tool increases in the SVC standard, since it can be used not only at the end of the reconstruction loop, but also as part of the inter-prediction step for recovering

978-1-61284-350-6/11/$26.00 ©2011 IEEE

information of the enhancement layers from the data contained at the base layer. Furthermore, the computational requirements are highly dependent on the type and levels of scalability which may be dynamically selected by the users.

In this work, a novel macroblock level parallelization of the DF algorithm is proposed. This technique has been applied on a novel coarse grain architecture based on a systolic array structure, adapting the original proposal provided in [5]. This means that different FUs work in parallel, like a multiprocessor system, but with the performance advantage of using specific elements instead of general purpose microprocessor cores. Furthermore, data transactions between elements can take advantage of the regularity, modularity and local communications of systolic arrays. This fact allows simplifying the communication schemes and also the distributed memory access and control.

The rest of the paper is organized as follows. Section 2 details the DF behaviour within H.264/AVC and SVC standards. Then, Section 3 makes a brief explanation about the state-of-the-art on multiprocessing DF approaches. Regarding the algorithm parallelization, our proposal will be described in Section 4, while the architectural approach is detailed in Section 5. Finally, Section 6 shows the most significant results, while in Section 7, conclusions from this work are outlined.

2. DEBLOCKING-FILTER BEHAVIOUR

The DF algorithm reduces blocking artifacts created on a macroblock (MB) by other functional modules, during its encoding or decoding process.

According to H.264/AVC and SVC standards, vertical edges of each 4x4 block within a MB are first filtered from left to right, and then horizontal edges are filtered from top to bottom. As it is shown in Fig. 2, in the filtering process of the 16x16 luminance component, the vertical edge V0 is conventionally first filtered horizontally from top to bottom, followed by edge V1, edge V2, and edge V3. The vertical filtering is performed in a similar way. Normally, edge H0 is vertically filtered from left to right, followed by edge H1, edge H2, and edge H3.

For each edge, the filter takes as its inputs four pixels of both sides of the edge, as shown in Fig. 3.

DF presents a highly adaptive nature. There are several conditions that determine whether a 4x4 block edge will be filtered or not, and the strength of the filtering for the block edges that will be filtered. The Boundary Strength (BS) parameter, α and β thresholds, and the values of the pixels in the edge determine the outcomes of these conditions. The BS parameter varies adaptively per block according to the coding information and the MB pixel values. Two adjacent 4x4 blocks share a BS value. The BS value ranges from four – strongest filtering– to zero – no filtering–. For a BS value of four, up to three pixels on either side of an edge can be modified by the filter, while for BS values from three to one, at most two pixels on either side of an edge may be affected.

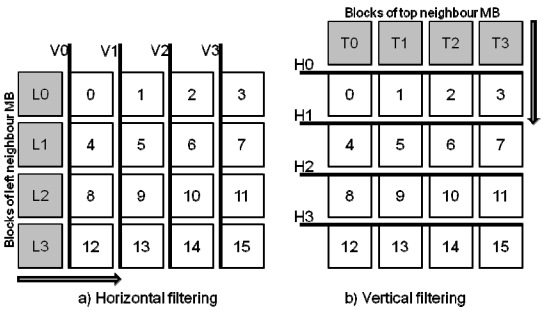

Fig. 2 MB edges for horizontal and vertical filtering

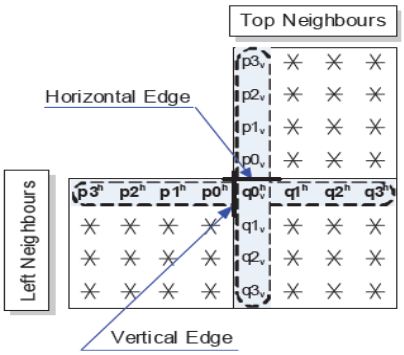

Fig. 3 Pixels distribution for horizontal and vertical filtering processes

The horizontal filtering must precede the vertical filtering as a restriction imposed by the standard. For filtering one MB there exist direct data dependencies with its upper and left neighbors even if they belong to different MBs, so they have to be filtered first. These dependencies limit the degree parallelism of this module.

3. STATE OF THE ART

In this section, some representative existing approaches published during the last years are reviewed, focusing both on specific hardware architectures and multiprocessing approaches.

3.1. Raster-Scan Deblocking Filter architectures

Conventional architectures are focused on reducing the number of cycles needed for filtering one MB. All of them have in common that they try to optimize the deblocking operation itself.

Regarding to this, [5], [6], and [7] use different memory accessing techniques in order to satisfy real-time constraints. With the same purpose, [8] and [9] explore internal data parallelism. Other proposals, such as [10] and [11], enhance the speedup of processing one MB by implementing double-filter architectures, where horizontal and vertical filtering operations are executed in parallel. However, in order to respect data dependencies among MBs, all of them are obeyed to filter MB by MB in a raster scan order, since they only use one FU.

It is important to remark that all these proposals do not take any benefit of data-parallelization at MB-level, so they finally have the limitations in speed terms derived from the fact of processing MB by MB.

3.2. Multiprocessing Approaches

Therefore, to overtake the performance bounds of sequential raster-scan solutions, it is mandatory to exploit some techniques for accelerating the execution process.

Some works have proposed MB-level parallelism techniques from a high level point of view. Their proposals are focused on parallelizing the whole H.264/AVC decoder, or at least some of its modules, making use of multi-core platforms such as [12], [13], [14] and [15]. Main characteristic of this parallelization is that they do not use the raster scan order for processing MBs, but they use a wavefront pattern like the one shown at Fig. 4, where T<x> denotes the execution period for each MB.

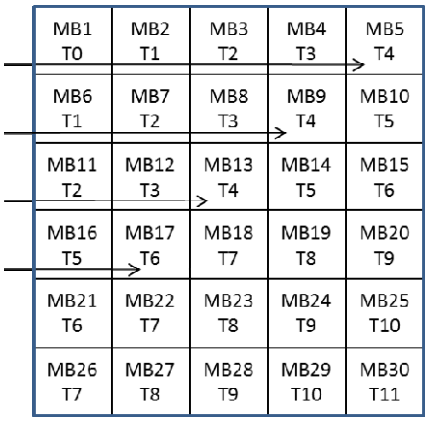

Fig. 4 Wavefront MB-level parallelization

However, none of these architectures is easily scalable. Therefore, they have to be designed for the worst processing case. As a result, many resources may stay in an idle state when facing more favourable conditions, such as smaller image size and/or frame rate. This reduces the efficiency of the system and it supposes an unnecessary waste of power and area. In addition, multiprocessing architectures exploit software acceleration instead of using hardware resources, which are faster and cheaper, suffering from a huge communication overhead.

4. PROPOSED PARALLELISM PATTERN

In order to achieve parallel execution, existing data dependences have to be analysed, rather than using raster scan order. A possible solution might be to exploit MB-level parallelism using a wavefront order in the same way than the state-of-the-art multiprocessing solutions. However, in these architectures it is necessary to wait all the clock cycles necessary for filtering a full MB, before the subsequent core starts its processing, as it has been depicted in Fig. 4.

An exhaustive analysis of the DF algorithm's behaviour permits to clarify data dependencies among MBs. All MBs are directly related with their left and top neighbour MBs, which have been previously filtered. This means that, on Fig. 4, MB6, for instance, has to be filtered after MB1, because the former needs some filtered pixels of the latter. Analysing more in detail the relationship among data dependencies and filtering processes (horizontal and vertical), it is observed how current MB horizontal filtering depends on previous neighbour MB vertical filtering, and how the current MB vertical filtering depends on top neighbour horizontal filtering. Considering these facts, the architecture proposed in this paper overcomes the drawbacks of previous multiprocessing proposals following an optimized wavefront order scan, in which data dependencies are considered by separating horizontal and vertical filtering in sequential stages. As a result, one *MB cycle* is saved among FUs, such as depicted in Fig. 5.

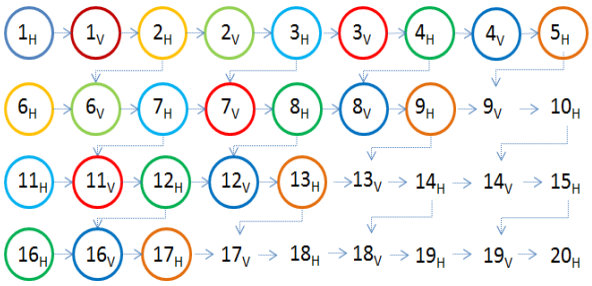

Fig. 5 MB scan order proposal and data dependencies

In spite of separating both the horizontal and vertical processes for each MB, both will be carried out in the same FU. This allocation strategy has been defined in order to minimize communication cost. Consequently, since semi-filtered data have to be shared between an MB and their top and left neighbours, an MB will be always filtered in the same unit than its left neighbour, and in the unit below the top neighbour. As it will be explained in the following section, specific connections between FUs have been created to allow the exchange of this semi-filtered information.

The pattern used for filtering all MBs contained in a frame is dependent on the total number of FUs of the architectural array and also the number of MBs of the height image frame. For instance, Fig. 6.a) represents the case in which the number of FUs matches the number of rows of MBs in a frame. Thus, each FU filters all MBs contained in a particular row of the frame but always respecting data dependencies. However, if the number of FUs is lower than the height of MBs in a frame, the filtering process is modified. The frame will be processed by stripes with a height same as the number of total FUs in the array. This case is depicted in Fig. 6.b), in which is shown how several FUs filter different rows of MBs belonging to two different stripes of the same frame.

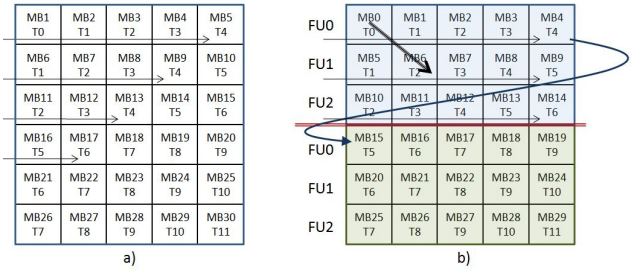

Fig. 6 Frame parallelization processing order with; a) infinite resources, b) limited resources

In addition, a reduction of the amount of transferred information between the external memory and FUs in

order to process one MB is obtained since, unlike state-of-the-art proposals, only current MB data must be requested, as the information related with the neighbouring MBs are received during horizontal and vertical filtering stages, directly from other FUs.

Following section explains in detail the proposed architecture in terms of its elements and their functionality.

5. SYSTEM ARCHITECTURE

The core of the proposed architecture is a homogeneous array of FUs**.** Each unit is able to carry out a complete filtering operation on an MB, so that the full array can process in parallel a region of the image, according to the strategy described above. To feed each FU with the required MBs, as well as to synchronize the array, modules in charge of controlling input and output memories have also been included in the architecture.

In the next subsections, a global view of the architecture will be provided, followed by a description of the basic FU. Finally, global synchronization issues will be explained.

5.1. Global Architecture

The proposed architecture is based on a coarse grain systolic array, as it can be seen in Fig 7. Systolic arrays are regular networks of interconnected elements that process data streams in a rhythmic fashion. The main strengths of these structures are their inherent parallelism, regular connections and data processing capabilities. The architecture described in this paper offers these features, not only regarding the parallelism, but also exploiting the regularity of the connections to reduce data transference overheads. To obtain these advantages, specific memory and control elements have been added to the processing array, as will be described in the following paragraphs.

Regarding parallelism, it is necessary to provide the architecture with a scheme to read the MBs from an external memory, in the order defined by the proposed pattern. This mechanism has been implemented in the Input Controller (IC), a module that generates the address sequences and control signals to read the necessary MBs in the correct time. The IC receives the MBs sequentially from the memory. However, all the parallel units have to be fed simultaneously. To parallelize data provision to the FUs, other modules named Input Memory (IM) blocks have been included at the top of the processing array. The main components of these blocks are FIFO memories. Input MBs coming from the IC are transmitted across these FIFOs, keeping each one only the MBs to be processed by the column of FUs below it. Once these memories have been filled, MBs have to be distributed in the vertical direction, that is, to the FUs of the same column. This process is performed with the help of modules named MB_routers**,** which have been attached to the FUs. These routers capture the first MB received from the IM in each processing stage, and transmit without change the subsequent MBs to the FUs below. The router also stores the luminance and chrominance information included on the MB into the corresponding internal memories of its FU. This strategy, together with the implemented address generation, assures that each FU receives the proper MB during the data sending stage.

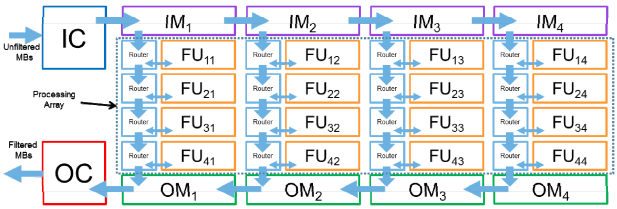

Fig. 7 Systolic array structure

Once all the FUs have been fed with the necessary MBs, they are processed in parallel. After the processing stage, the router will transmit the processed MBs to the elements in the bottom, called Output Memory (OM) blocks. These blocks are also based on FIFO memories that store the elements received from the vertical connection, and transmit them again in sequential order to the Output Controller (OC). This controller will send back the processed MBs to the intermediate buffer. Data sending, processing and results transmission stages have been pipelined and overlapped, as it will be later explained.

The blocks described above include distributed control logic to manage data transmission. This logic is also connected with the FUs to synchronize data management with the processing stage. Distributed control makes the architecture fully scalable, by means of the addition or removal of modules to it. These modules automatically communicate only with their neighbours using shared signals, without having to implement a centralized control designed *ad hoc* for each possible architecture size. Regarding the generation of the MB addresses, the IC generates the correct sequence, just configuring some generic values included in its RTL description. The unique limitation of the maximum size of the architecture is the size of the memories of the IM and the OM blocks, as will be analysed in the implementation results section.

Regarding the data shared among the different units, each MB is completely filtered in the same FU. Consequently, MB data don't have to be transmitted among FUs. However, semi-filtered MBs information has to be transmitted to the units in charge of the execution of the top and left neighbouring MBs, as it has been described in Section 4. Going back to the allocation of MBs to FUs, thanks to the locality of the communications, neighbours will be processed in contiguous FUs. In the worst case, instead of being in the position directly below, the next FU will be the one at the top of the next column. In consequence, extra signals to communicate this information with the next MB have been included, considering both possibilities. Furthermore, a mechanism has been also included in the IC to synchronize this semi-filtered data, when a FU reaches the end of one line of the image.

5.2. Computational Element: Functional Unit

A FU has been designed to be able to work alone or to be assembled, forming arrays or 2D-matrix structures. Each one behaves like a DF unit on its own, where full MBs are filtered (luminance and chrominance). Although these

variations might suppose a notable increase in area and resources, final performance is also enhanced.

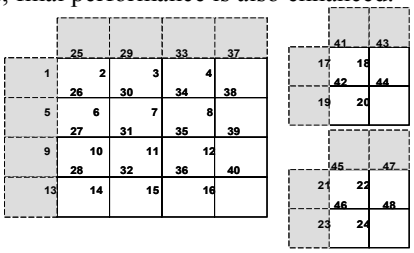

Fig. 8 Filtering order

All operations for filtering an MB are executed sequentially instead of operating in parallel. First, horizontal filtering is executed and, after it, the vertical one. Although this method seems to be slower than combining both filtering operations at the same time, it saves resources and also provides a slot with time enough for interchanging data among MBs, receiving a new unfiltered MB or sending previous filtered one.

Finally, in order to adapt as much as possible the filtering process to our execution requirements, Fig. 8 shows block by block the filtering order.

There are four important internal elements within the scalable DF architecture: router, BS, filter and two matrix transposers. The first one was explained in detail in the previous subsection. BS_module calculates the filtering strength as the standards demand. Then, filter_module is used for horizontal and vertical filtering; modifying two LOPs (line of pixels) per cycle, where the number of pixels involved depends on the BS value. Finally, the transposer_module modifies the orientation of a LOP, applying a transposition by using three pipelined registers of 32 bits.

5.3. Proposed synchronization stages

After having described both the global architecture and the structure of the FU, the different processing stages of the proposed DF architecture are shown in this section, including how they have been overlapped to reduce the communication overhead.

Time needed for filtering one MB in one direction will be referred as an MB cycle. Each FU spends two MB cycles, one for horizontal filtering (HF) and another one for vertical filtering (VF). This strategy does not increase the global cost in terms of time significantly, due to the processes that are executed together with the filtering operations, as part of the HF and VF.

More in detail, the overlapping has been designed as follows:

- At the same time that HF is being executed, the filtered MB finished at the previous cycle is sent to the reconstructed memory through the router. Moreover, the semi-filtered left neighbour of the current MB is sent to the bottom FU for acting as a top neighbour into the VF. Simultaneously, the current FU is receiving data from its top FU neighbour.
- Once HF is finished, VF starts. Concurrent to the VF, the MB that is being filtered is stored into the internal memories for being used as a left neighbour with the next MB, in the same FU. The current top neighbour is prepared to be sent to the router after being filtered. Furthermore, during this MB cycle, the FU also receives a new unfiltered MB which will be filtered in the following MB cycle.

6. IMPLEMENTATION AND RESULTS

In this section, the proposed architecture is analysed in terms of resource consumption and performance results. The proposal has been synthesized on a medium size Virtex-5 LX110T FPGA, using ISE 12.1 tools.

Regarding resource occupation, Table 1 shows synthesis results of each basic block of the architecture. Information about the maximum achieved frequency has been also included.

Table 1. Synthesis results of the architecture with one FU

	Synthesis results using V5LX110T					
	IC	OC	IM	OM	Router	FU
Slices reg.	357	116	172	124	90	1814
Slices LUTs	444	108	134	226	112	2274
Block RAM (36kb)	1	0	2	2	0	10
Freq. (Mhz)	246	236	400	286	232	124

From data shown in Table 1, it can be drawn that the resource overhead introduced by the communication (IM, OM) and control (IC, OC) blocks is small, compared with the cost of each FU. Also, the maximum working frequency of these functional blocks is not a limitation, regarding the FU value. Moreover, despite that the communication and computation structures vary according to the desired configuration, control blocks remain unaltered since they are only implemented once.

In Table 2, total area of the architecture is shown, for different processing array sizes. Concerning the flexibility of this HW proposal, different architectures with four FUs have been included in the analysis. The growth of these configurations is different, since 1x4 grows vertically, 4x1 horizontally and 2x2 covers both directions. As a consequence, the performance in resources, latency and control complexity varies, since vertical structures need more cycles to load all the FUs, and horizontal structures occupy more logic elements because of the communications overhead.

Regarding performance, Table 3 shows a comparison of the FU designed to be included in the architecture presented in this paper, with other state-of-the-art architectures. Despite of the existence of architectures that spend fewer cycles for filtering one MB, the difference it is not significant enough considering the speed-up that can be achieved filtering a full video frame, increasing the number of FUs working in parallel, according to the proposal of this work. An evaluation of the impact of the number of FUs of the architecture, in terms of MB filtering cycles (equivalent to 208 clock cycles), is shown in Table 4.

Table 2. Area occupation of the architecture with different array size

	Array size of FUs					
	2×2	4x1	1x4	3×3	4×4	5×5
Nº of slices reg.	6743	7329	6450	14493	25129	38718
Nº of slices LUTs	9088	9703	8817	19712	34400	51979
Nº of BRAMs	50	62	44	99	169	259
Clock period (Mhz)	125	125	125	124	124	123

Table 3. Architectural comparison for 1 FU

Architecture	Clock cycles/MB	Clock Freq. (Mhz)
[16]	232	85.2
Proposal (1 FU)	208	124.3
[17]	110	126
[18]	100	100

Table 4. MB cycles on different image format and array sizes

Resolution (MB/frame)	MB cycles for filtering one frame according to the number of FUs						
	1FU	2FU	3FU	4FU	5FU	6FU	7FU
SQCIF(8x6)	57	33	25	19	18	17	15
QCIF(11x9)	111	57	45	35	27	24	25
CIF(22x18)	419	221	155	113	92	89	71
4CIF(44x36)	1629	837	573	441	354	309	266
16CIF(88x72)	6425	3257	2201	1673	1323	1145	971

Performance measurements shown in the previous table consider the acceleration provided by the parallelism, but also the time required to fill all the processing units with valid info, both at the beginning and the end of the frame. These facts make that, for each size of the image, the optimal number of FUs may be different. Attending to these results, this architecture is able to process a 4CIF (704×576 pixels @30fps) video sequence in a wide range of frequencies, varying its values between 10.16Mhz, 5.22Mhz, 2.75Mhz in case of using one, two or four FUs respectively.

7. CONCLUSIONS AND FUTURE WORK

This paper presents a novel scalable deblocking filter architecture which exploits an optimal MB-level parallel strategy. This characteristic provides the necessary flexibility to allow an easy reuse of the design among different profiles and scalability levels. The scalability refers to the possibility of varying the number of FUs according to the final system requirements. Regarding to the MB-level parallelism, this pattern demands to use only one filter unit per FU for executing horizontal and vertical filtering operations. This customizable parallelism offers the possibility of achieving, by changing some generic parameters, different trade-off points of the area/performance design space.

About future developments, we are working on implementing this architecture exploiting dynamic reconfiguration features. This approach will avoid stopping system execution for loading a new configuration bitstream. Definitively, this will allow us to work with a scalable and dynamically reconfigurable DF architecture, capable of adapting its resources and performance to the variable external conditions.

8. ACKNOWLEDGMENT

This work is supported by the Spanish Ministry of Science and Innovation and European Union (FEDER funds) as part of the I+D+I Plan 2008-2011) support program in the context of Dynamic Reconfigurability for Scalability In Multimedia Oriented Networks (DR. SIMON) project, under contract TEC 2008-065846-C02.

9. REFERENCES

[1] ITU-T Rec. H.264 and ISO/IEC 14496-10 (MPEG-4 AVC), ITU-T and ISO/IEC JTC. *H.264/AVC Standard Advanced Video Coding for Generic Audiovisual Services*, Version 1: 2003 – Version 7: 2007.

[2] ITU-T Rec. H.264 and ISO/IEC 14496-10. *H.264/AVC extension* (Scalable Video Coding - SVC). *Advanced Video Coding for Generic Audiovisual Services*, Version 8: 2007 – Version 10: 2009.

[3] H. Schwarz, D. Marpe, T. Wiegand. *Overview of the Scalable Video Coding Extension of the H.264/AVC Standard*. IEEE Trans. on Circuits and Systems for Video Technology, vol. 17, n 9, pp. 1103-1120, 2007.

[4] I.Werda, T. Dammak, T. Grandpierre, M. Ayed, N. Masmoudi. *Real-time H.264/AVC baseline decoder implementation on TMS320C6416*. Journal of Real-Time Image Processing. pp. 1-18, 2010. Springer Berlin.

[5] Otero, A.; de la Torre, E.; Riesgo, T.; Krasteva, Y.E.; , "Run-Time Scalable Systolic Coprocessors for Flexible Multimedia SoPCs," *International Conference on Field Programmable Logic and Applications (FPL), 2010*, vol., no., pp.70-76, Aug. 31 2010-Sept. 2 2010

[6] Y.W. Huang, T.W. Chen, B.Y. Hsieh, T.C.Wang, T.H. Chang, L.G. Chen. *Architecture design for deblocking filter in H.264/JVT/AVC*. IEEE International Conference on Multimedia and Expo, 2003.

[7] C.-C. Cheng, T.-S. Chang, K.-H. Lee. *An in-place architecture for deblocking filter in H.264/AVC*. IEEE Trans. on Circuits and Systems II, vol. 53, n 7, pp. 530-534, 2006.

[8] T.-M. Liu, W.-P. Lee, T.-A. Lin, C.-Y. Lee. *A memory-efficient deblocking filter for H.264/AVC video coding*. IEEE International Symposium on Circuits and Systems, vol.3, pp.2140-2143, 2005.

[9] L.Li, S. Goto, T. Ikenaga. *A highly parallel architecture for deblocking filter in H.264/AVC*. IEICE Trans. on Information and Systems, vol. E88-D, n 7, pp. 1623-1629, 2005.

[10] S.-Y. Shih, C.-R. Chang, Y.-L. Lin. *A near optimal deblocking filter for H.264 Advanced Video Coding*. Asia and south Pacific Conference on Design Automation, pp. 170-175, 2006.

[11] T.-H. Tsai, Y.-N. Pan. *High efficient H.264/AVC deblocking filter architecture for real-time QFHD*. IEEE Trans. on Consumer Electronics, vol. 55, n 4, pp. 2248-2256, 2009.

[12] M. N. Bojnordi, M. R. Hashemi, O. Fatemi. *A fast two dimensional deblocking filter for H.253/AVC video coding*. CCECE, pp. 2017-2020, 2006.

[13] A. Azevedo, B. Juurlink, C. Meenderinck, A. Terechko, J. Hoogerrugge, M. Alvarez, A. Ramirez, M. Valero. *A Highly Scalable Parallel Implementation of H.264*. Trans. on HIPEAC, vol. 4, Issue 2, 2009.

[14] M. Alvarez, A. Ramírez, A. Azevedo, C. Meenderinck, B. Juurlink, M. Valero. *Scalability of Macroblock-level Parallelism for H.264 Decoding*. ICPADS, 2009.

[15] W. Kim, K. Cho, K. Chung. *Stage-based Frame-Partitioned Parallelization of H.264/AVC Decoding*. IEEE Transactions on Consumer Electronics, vol.56, Issue 2, pp. 1088-1096, 2010.

[16] K. Sihm, H. Baik, J. Kim, S. Bae, H. Song. *Novel approaches to parallel H.264 decoder on symmetric multicore systems*. ICASSP pp. 2017-2020, 2009.

[17] K. min, J.-W. Chong. *A memory and performance optimized architecture of deblocking filter in H.264/AVC*. IEEE Conference on Multimedia and Ubiquitous Engineering, 2007.

[18] F. Tobajas, G.M. Callicó, P.A. Pérez, V. de Armas, R. Sarmiento. *An efficient Double-FIlter Hardware Architecture for H.264/AVC Deblocking Filter*. IEEE Transactions on Consumer Electronic, vol. 54, n 1, pp. 131-139, 2008.

[19] M. Torabi, A. Vafaee, N. Movahhedinia. *A fast architecture for Deblocking filter in H.264/AVC using clock cycles saving process*. IMPACT, pp. 324-327, 2009.