Motion estimation and CABAC VLSI co-processors for real-time high-quality H.264/AVC video coding

Motion Estimation and CABAC VLSI Co-Processors for Real-Time High-Quality H.264/AVC Video Coding

*Sergio Saponara, †Maurizio Martina, *Michele Casula, *Luca Fanucci, †Guido Masera
*Department of Information Engineering, University of Pisa, via Caruso 16, I-56122 Pisa, Italy
† Department of Electronics, Politecnico di Torino, C.so Duca degli Abruzzi 24, I-1029 Torino, Italy

Summary
Real-Time and high-quality video coding is gaining a wide interest in the research and industrial community for different applications. H.264/AVC, a recent standard for high performance video coding, can be successfully exploited in several scenarios including digital video broadcasting, high-definition TV and DVD-based systems, which require to sustain up to tens of Mbits/s. To that purpose this paper proposes optimized architectures for H.264/AVC most critical tasks, Motion Estimation and Context Adaptive Binary Arithmetic Coding. Post synthesis results on sub-micron CMOS standard-cells technologies show that the proposed architectures can actually process in real time 720×480 video sequences at 30 frames/s and grant more than 50 Mbits/s. The achieved circuit complexity and power consumption budgets are suitable for their integration in complex VLSI multimedia systems based either on AHB bus centric on-chip communication system or on novel Network-on-Chip (NoC) infrastructures for MPSoC (Multi Processor System on Chip).

Key words:
Video Coding, Hardware Architectures, Motion Estimation, Entropy Coder, Network on Chip, VLSI Multimedia Systems

1. Introduction

H.264/AVC is the new video coding standard released by ITU-T and ISO/IEC [1]. Compared to previous H.26x and MPEGx standards, H.264/AVC superior compression efficiency and high scalability make it suitable for different scenarios. Target applications range from low bit-rate video communications, supported by the Baseline Profile of H.264/AVC, with maximum data rates of few hundreds of kbits/s and QCIF (174×144) formats up to high-quality video delivery and storage at tens of Mbits/s and with large formats. The high quality scenario, covered by the Main Profile of the standard, includes applications such as digital video broadcasting-terrestrial and handheld (DVB-T/H), high definition TV and DVD-based systems. The implementation of hardware co-processors, able to sustain real-time and high-quality H.264/AVC video coding, is needed to grant high performance.

Figure 1 shows a block diagram of the H.264/AVC encoder. Like its H.26x/MPEGx ancestors it is based on a hybrid scheme compressing the information both in the temporal domain by INTER-frame Motion Estimation and Compensation (ME and MC in Figure 1) and in the spatial domain by INTRA-frame Transform coding and Quantization of the residual estimation error (T and Q units in Figure 1). The block scheme shown in Figure 1, that is referred to INTRA-frame coding, can be straightforwardly adapted to INTER-frame coding feeding the adder and the subtracter with the *MC multiblock size multiframe* block output (*inter*). An Entropy Coder further reduces data redundancy in the bit-stream. The compressed output stream is then passed to a Network Abstraction Layer (NAL) unit where data are packaged depending on the characteristics of the specific communication network.

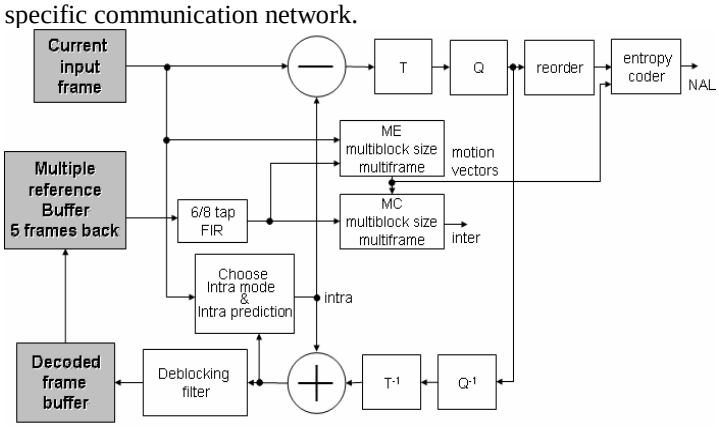

Fig. 1. Block diagram of the H.264/AVC Video Coding Layer

With respect to previous coding standards, H.264/AVC includes additional features, particularly in the ME task for inter frame prediction, adopting multi-reference frames and variable block sizes, and in the Entropy Coding task, adopting a Context Adaptive Binary Arithmetic Coder (CABAC). The performance and complexity profiling analysis on the reference C model of the encoder proves that these features improve the coding efficiency by a factor two at the expense of increased computation and memory costs up to one order of magnitude [2-4].

Variable block size and multi reference frame ME is supported in both Baseline and Main Profiles; CABAC is supported in the Main Profile while for the Baseline Profile a Context Adaptive Variable Length Coder (CAVLC) is used. With respect to CAVLC the CABAC scheme allows up to 15% bit-rate saving for a fixed visual quality at the expense of computation and memory complexity overheads of 30% [2, 5]. As proved in literature [2-4], ME and CABAC are the bottlenecks of the standard in terms of required computation complexity and the design of hardware co-processors to support such

features is mandatory for the real-time and cost-effective realization of H.264/AVC-based systems. To this aim two VLSI intellectual property (IP) macrocells, dedicated to ME and CABAC processing, are presented in the paper. They allow for real time implementation of H.264/AVC coding in high quality scenarios where up to tens of Mbits/s are reached, as in the Main Profile. Optimizations are addressed at algorithmic and architectural level and their complexity and power consumption budgets are suitable for integration in complex VLSI multimedia systems targeting real-time up to 30 frames/s 720x480 formats. To ease the IP assembling, two interfaces towards an AHB bus centric infrastructure and a Network-on-Chip (NoC) communication backbone are provided.

Hereafter, Section 2 briefly reviews state of the art in hardware design for H.264/AVC video coding. Section 3 describes a novel context-aware low-complexity ME technique. The ME hardware architecture and the relevant synthesis results in submicron CMOS technology are presented in Section 4. Section 5 deals with CABAC algorithmic description. CABAC hardware architecture and CMOS implementation results are detailed in Section 6. Section 7 is about the design of a Network Interface (NI) for the proposed IPs towards a Spidergon NoC platform; the latter is configured to connect AHB bus based H.264/AVC blocks to a NoC thus taking advantage of parallel computing. Conclusions are drawn in Section 8.

2. Hardware Design for H.264/AVC

In the literature several works have been proposed concerning the hardware implementation of building blocks of the H.264/AVC codec [6-15], [18-20], [23]. Single-chip coders have been also proposed, as in [4] where a RISC programmable core supports the control tasks while the signal processing functions are all realized by dedicated hardware units. All the cores in the above cited works are connected by means of a standard shared-bus communication model. The single-chip coder in [4], realized in a 0.18 μm CMOS technology, implements the Baseline Profile of the standard targeting 30 frames/s CIF (352×288) and CCIR (720×480) formats. However the support of CABAC and hence of the Main Profile is still missing since the basic CAVLC entropy coder is adopted in [4].

Due to the advances in on-chip communication paradigms, new solutions are available for future embedded sub-micron MPSoC. NoCs [25, 26] allow large bandwidth data delivery in parallel multi-core systems and offer a better performance for video encoding standards [27-32, 34] compared to single/ hierarchical bus systems. That means future MPSoC can implement video encoding applications designing specific blocks for the different tasks and connecting them through high throughput NoC-based parallel communication. The burden of CABAC and Motion Estimation operations is still the bottleneck in H.264/AVC even if efficient on-chip networking improves overall performance.

Some works in the literature concern CABAC implementation: in [7] a mixed hardware/software system is proposed, whereas [8, 9, 36] concentrate on CABAC dedicated coprocessors. Optimized hardware implementations limited to the CABAC unit are proposed also in [18, 19, 20, 37]. In [18] an FPGA based RISC CPU extension is proposed to accelerate CABAC in a rate distortion framework. The works [19, 37] deal with the architecture of a CABAC decoder while in [20] an encoder implementation is investigated.

As far as ME is concerned, the adopted solution in [4] is a large systolic array of 256 processing elements implementing a classic Full Search (FS) technique, known to be not efficient in terms of performance vs. complexity trade-off [11-13], [23]. Hardware engines based on a systolic array of processing elements for FS ME with variable block sizes have been also proposed in other works, e.g. [15]. To reduce the complexity of the conventional brute force FS while keeping similar coding efficiency many fast ME techniques have been investigated in the literature [11-15], [23]. Among them UMHexagonS (UMHS) [11] has been officially accepted as the fast ME technique in the JM reference software model of the standard [16, 17]. It realizes a predictive search, which adopts a hexagonal window in the refining phase plus proper stop criteria. A hardware architecture for UMHS ME has been discussed in [23]. The implementation of a different approach based on the Phase Plane Correlation was described in [38].

It is worth noting that in FS, in UMHS and in most of known ME techniques, the basic search is repeated multiple times in case of multiple reference frame or variable block sizes (7 block sizes in H.264/AVC: 16×16 macroblock and its sub partitions 16×8, 8×16, 8×8, 8×4, 4×8 and 4×4 blocks). Since ME operations increase with the number of blocks and reference frames, unnecessary redundancy is introduced by UMHS and most of known ME techniques in terms of computations and memory accesses. Adaptive ME techniques have been proposed by the authors in [14, 24, 40] but the ME algorithms were mainly optimized for low/mid image formats (QCIF/CIF), with fixed search ranges and moderate dynamic scenarios (telconferencing or video phone applications) while this work extends the research to larger formats (CCIR) and search sizes (from 4 to 64), considering higher dynamic scenes and higher quality video applications. Moreover, the architectures in [14, 40] have a single search area local buffer which, as discussed later, is a bottleneck when working with high dynamic scenes using always more than 1 reference frame. Our previous ME architecture was also conceived as a coprocessor with a simple interface, tightly coupled to a single RISC core; however high quality video systems are based on multi core architectures requiring

more complex communication infrastructures whose design are addressed in this paper.
This paper concentrates on the whole H.264/AVC framework and deals with the most computationally intensive tasks, showing architectures suited for real-time and high-quality video coding in VLSI multimedia systems. As far as CABAC is concerned a high speed FIFO-based architecture is presented. Since the FIFO size impacts both on the performance and the complexity of the CABAC architecture, it should be consciously sized. In this paper the FIFO sizing is thoughtfully discussed to grant very high performance with a reduced complexity increase. For ME an innovative adaptive algorithm with its relevant hardware architecture is proposed. The novel technique avoids unnecessary computations and memory accesses, whereas it achieves the same high coding quality of FS.
To ease their integration in complex embedded systems, the CABAC and ME architectures have been designed as reusable IP VHDL macrocells and have a wrapper to AMBA AHB bus, which is a defacto standard for system-on-chip (SoC) communication. To facilitate the IP integration in MPSoC also a Network Interface has been designed which wraps the AHB interface of the IPs to the Spidergon NoC protocol. The two IP macro-cells have been also characterized in terms of area, power and computing performance on 180 nm and 65 nm CMOS standard-cells technologies.

3. Adaptive Fast ME Technique

To avoid unnecessary computations and memory accesses for the ME task in H.264/AVC-compliant coders we propose to add a low complexity context-adaptive controller to a basic search engine, FS or Fast ME as UMHS. The controller extracts from the search engine partial results, i.e. Motion Vectors (MV) and Sum of Absolute Difference (SAD) cost, information on the input signal statistic, using them to automatically configure run-time the following ME search parameters: number of reference frames, valid block sizes and search area for each 16×16 macroblock and its sub-partitions down to 4×4-pixel blocks. This section presents briefly the algorithmic level design and performance, focusing on performance over search area displacements ranging from ±4 to ±64. The global context-adaptive controller combines three basic algorithms which have been proven efficient in both FS and fast ME engines in the case of a fixed search area displacement of 16 [24]. In this work 7 middle format video sequences with high-dynamic scenarios are codified using the basic UMHS ME and our adaptive-controlled UMHS ME with search displacements ranging from 4 to 64. The adopted test videos are reported in Table 1. They have been encoded at 30 frames/s with a Quantization factor of 28 using the reference JM10 software implementation of H.264/AVC with UMHS ME. Bit-rate and ME time performances are reported in the case of a search displacement of 4.

Sequence	Format	Bit-rate, kbit/s	ME Time (s)
Stefan	SIF	1190.9	49.5
Mobile	SIF	1668.1	46.6
Garden	SIF	2280.5	62
Bus	CIF	1265.1	78.3
Foreman	CIF	840.1	72.8
TennisTable	CCIR	5625.8	248.9
Mobile	CCIR	5937.2	223.6

Table 1 – Absolute performance of UMHS (search displacement of 4)

In Figure 2 bit-rate performances, normalized vs. the values in Table 1, are reported for sequences using UMHS ME with displacements of 4, 8, 16, 32, 64. Figure 3 reports normalized ME time performances obtained in the same conditions of Figure 2. For the horizontal axes in Figures 2 and 3 a logarithmic scale is used. In Figures 2 and 3 it can be seen that displacements greater than 16 do not add meaningful improvements in the compression efficiency, while the ME processing time is increased by a factor 3 when sweeping from 16 to 64. It is then a waste of resource trying to estimate each macroblock (MB) position in all the possible interpolated positions of the search area. These results depend on the fact that ME engines use motion vector predictors, which centers the search area in the most probable location of a local minimum cost. Block matching operations do not start around the position of the current processed MB but the starting point is translated elsewhere depending on statistic data.
The aim of our work is to improve the results in Figures 2 and 3 and Table 1 applying a dynamic control to the ME search engine. Basically the trade-off between compression efficiency (measured in terms of bit-rate for a given PSNR) and ME complexity (evaluated in terms of ME processing time inside the JM coding framework) must be improved when the search area, the number of reference frames and the block types grow.

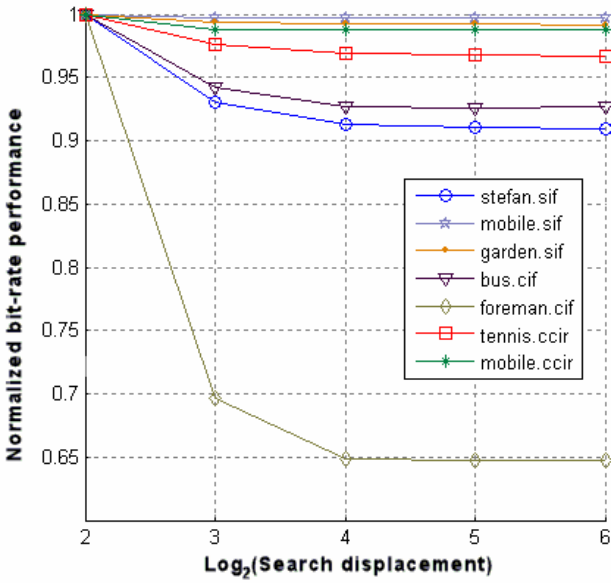

Fig. 2 Normalized bit-rate for search displacements ranging from 4 to 64 (in $\log_2$ scale) using the UMHS ME

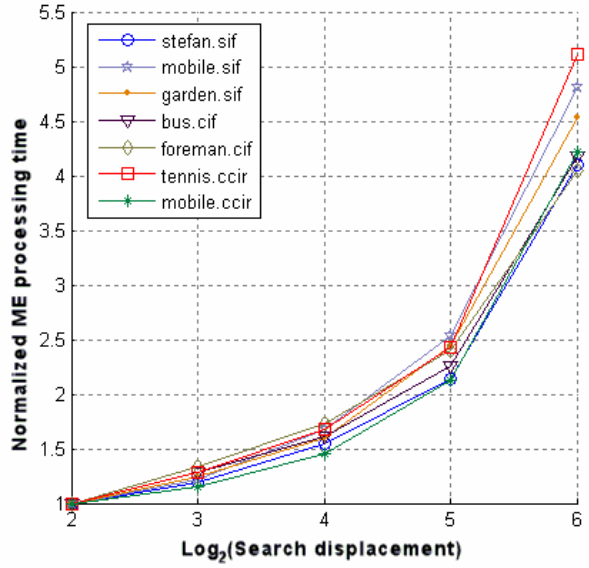

Fig. 3 UMHS ME Time growth factor vs. search displacements ranging from 4 to 64 in $\log_2$ scale

The three controls implemented to adjust the ME parameters at coding time are summarized hereafter:

A) The **Search Area Control**. It was originally proposed for the automatic search range configuration of a FS engine in [14] and then extended for fast ME engines [24], but limited to a fixed search displacement of 16. In this work the optimal horizontal and vertical search displacements for the block under estimation, using a Fast ME engine, are selected in the range (1, 64) by comparing with proper thresholds the SAD and MV values of already encoded neighboring blocks: 3 spatial blocks in up, up-left and up-right positions in the current frame and 1 temporal block, occupying in the previous frames the same position of the block under estimation in the current frame.

B) The **Modes Control**. As far as variable block sizes is concerned, a profiling analysis of the JM standard software model conducted by the authors in [2] proves that using the smaller block sizes is useful for images with complex texture while it can be avoided for homogenous ones to reduce complexity. Thus, we have devised a control over smaller block sizes (4×8, 8×4 and 4×4 partitions) to decide which of them must be enabled for ME each time a 16×16 macroblock is encoded. It accomplishes its task by comparing the SAD cost of the current 16×16 partition with 3 thresholds. The objective of such comparisons is to enable smaller partitions when the SAD cost is high. So when the first threshold is exceeded 4x8 partition is enabled, if the second threshold is exceeded even 4x8 is enabled. The third threshold controls if the 4x4 partition has to be enabled.

C) The **Reference Frame Control**. It decides the maximum number of reference frames RF to be used in the range (1, 5) for the ME of a 16×16 macroblock and its selected sub-partitions. According to the flow chart in Fig. 4, the SAD cost (SAD_{min}) and optimal reference frame RF are initially detected for the 16×16 partition and then are used to decide how many reference frames are useful for the enabled smaller sub partitions in the current frame and for the same 16×16 partition in the next frame. To this aim SAD_{min} is compared to 5 thresholds from THR1 to THR5. Details on how thresholds are worked out and how they impact ME parameters are similar to what we discussed in [24].

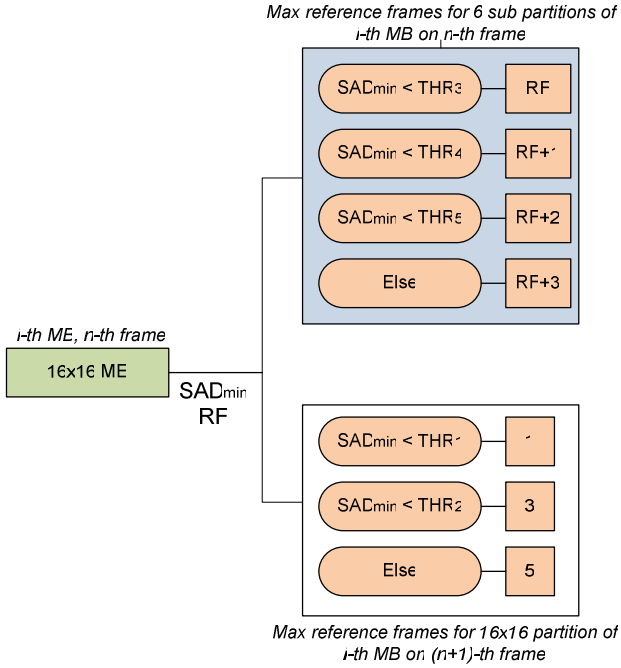

Fig. 4 Reference frame decision scheme

The encoding process using all the three controls is accomplished following these 5 steps.

• **Step 1.** The first frame of a video sequence is coded in INTRA mode and the ME technique is not used. The second frame is the first frame coded INTER and for the ME of all its macro blocks the basic search engine (FS or a FAST engine as UMHS) is used without the new context-aware control. Starting from the third frame, the steps as reported in Figure 5 are repeated iteratively.

• **Step 2.** The optimal search area and reference frame number for the 16×16 macroblock are preliminarily sized using the algorithms in A) and C).

• **Step 3.** The basic search engine, UMHS or FS, performs the ME for the 16×16 partition.

• **Step 4.** Using data (SAD_{min} value and optimal reference frame RF) from the previous operation the controls in B) and C) decide which sub partitions must be enabled for the ME and how many reference frames (see flowchart in Fig. 4) must be used for the search. The search size for the enabled sub-partitions is the same derived for the 16×16 partition in Step 2.

• **Step 5.** The search engine completes the ME task for the current macroblock and then starts with a new macroblock processing by iterating the same flow from Step 2 to Step 5.

With reference to the test videos with different image formats and target bit-rates, Figures 6 and 7 compare the performance of the new control technique applied to UMHS vs. the original UMHS.

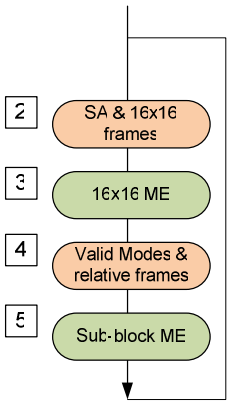

Fig. 5 Operations' flow for ME (Steps 2-5) using search parameters dynamic control

Results in Figures 6 and 7 are expressed in terms of:
- ΔBR% which is the bit-rate deviation measured as (BR2 – BR1)/BR1. BR2 is the bit-rate obtained by our ME adaptive-controlled encodings, while BR1 is the one obtained with the basic ME.
- ΔMET% which is the saved % of the ME processing time when integrating our controller into the JM 10 model and running it on a general purpose AMD Athlon processor. It is measured as (MET1 – MET2)/MET1. MET1 refers to ME encoding time with basic ME, while ME2 refers to ME encoding time when the proposed controls are applied.

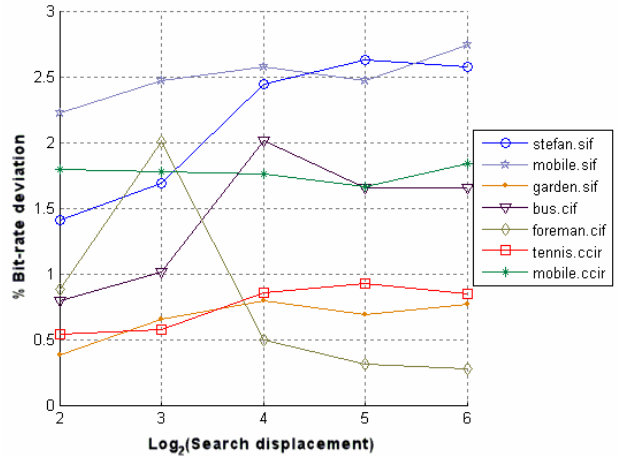

Fig. 6 ΔBR% for test sequences vs. search displacements from 4 to 64

The bit-rate deviation of our adaptive controlled ME vs. original UMHS is always below 3% with an average bit-rate slightly over 1%, while the saved ME time averages in the range of 60 – 70% (see Figures 6 and 7). This is true for all displacements. The longer the search displacement, the higher the saved ME time. This is because at displacements greater than 16 the adaptive control of the search area detects optimal motion estimation in a sub region of the available one. As seen in Figure 2, costs found for MB in searches with a 32 or 64 displacement are not better than those for 16, thus the algorithm is able to find such costs adapting the displacement to a much smaller one. The results in Figures 2 and 3, Figures 6 and 7 refer to a fixed quantization parameter QP=28 for the encoder. The same results can be obtained for other QP values. As example, Fig. 8 shows the Stefan sequence with a search displacement of 16. The comparisons are reported in terms of PSNR and bit-rate values changing the QP in a typical range 20 to 40. To be noted that Stefan is one of the most demanding sequences in terms of ME resources and the two curves have a gap limited to 0.1 db at fixed bit-rate. Time savings obtained with our adaptive controlled ME vs. basic UMHS are always in the range of 60% - 70% for every point in the curve.

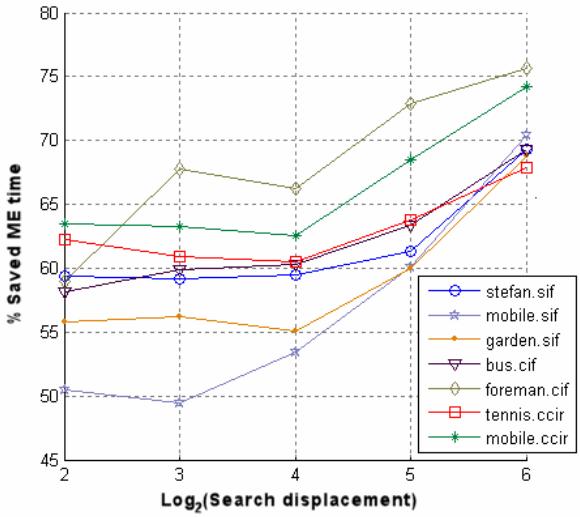

Fig. 7 ΔMET% for test sequences vs. search displacements from 4 to 64

Table 2 shows how FS ME behaves when coupled with our adaptive control for a search displacement of 16. Bit rate deviations are on average very close to the ones obtained on UMHS. The saved ME time is slightly better on FS achieving an average value of about 70%. This is because the area control reduction has more influence in FS, where all points inside a given area are used to perform a block matching operation. These results, achieved with FS and UMHS, demonstrate that out adaptive controls can be applied and behave well with different ME engines.

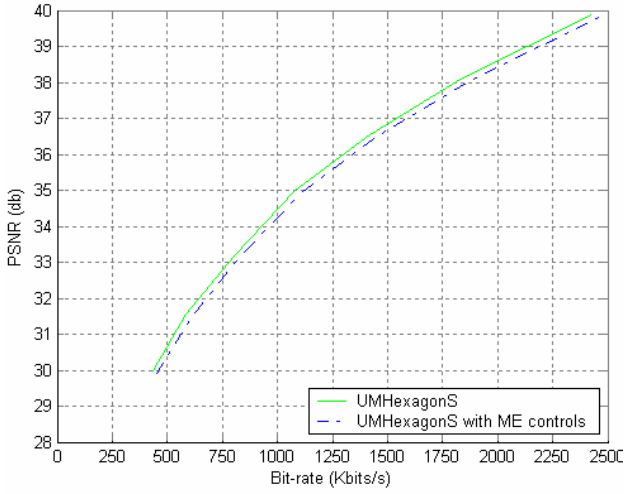

Fig. 8 Rate-distortion curves for Stefan SIF

Sequence	ΔBR%	ΔMET%
Stefan	2.5	69.8
Mobile SIF	2.3	64.8
Garden	0.9	66.2
Bus	2.3	72.7
Foreman	0.7	75.4
Tennis Table	1.4	77.2
Mobile CCIR	1.1	74.1

Table 2 – Bit-rate deviation and saved ME time percentages for the test sequences applying our adaptive ME controls to FS

4. ME Architectural Design

4.1 Architecture Description

The results reported in Section 3 for the ME refer to a software implementation on a general purpose Athlon processor at 2 GHz with 1GB RAM and Windows XP operating system. The original UMHS software implementation is far from real-time coding. However, thanks to the complexity reduction of our technique, real-time is achieved for the 30 frames/s QCIF videos; for CIF ones the real-time is allowed at a frame rate between 15 and 30 frames/s depending on the sequence dynamism. To achieve real-time for larger formats and/or to reduce the power consumption of the software approach for low-power terminals, a dedicated hardware architecture is needed. In this case the proposed technique can be implemented according to the block diagram in Figure 9. For the realization of the relevant building blocks (hardware search engine, ME parameters and I/O controller, AHB wrapper) we have reused pre-designed IP VHDL cores already described in [21, 22]. The overall architecture has been synthesized, characterized and tested in 65 nm and 180 nm CMOS standard-cells technologies.

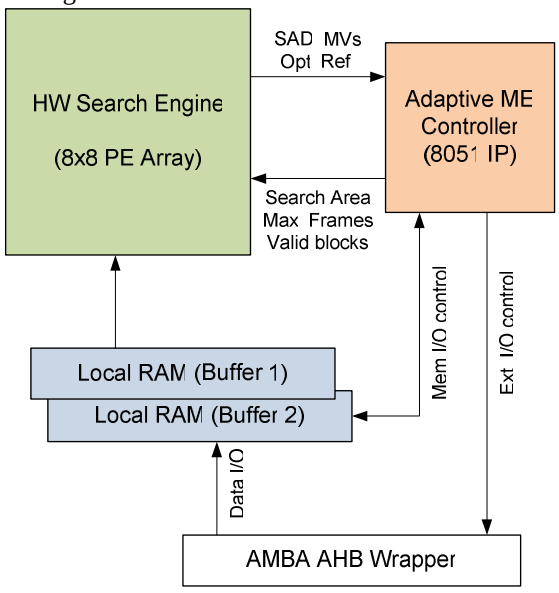

Fig. 9 Block diagram of the ME hardware architecture

In Figure 9 a microcontroller core is used to implement all the ME context-aware control tasks plus the scheduling of UMHS operations and the management of I/O data flow. Such tasks can be easily realized in real–time for the target video formats by the power optimized 8051-compliant microcontroller core that we described in [21, 35]. It features a fully synchronous 8-bit architecture executing most of instructions in one clock cycle. Its circuit complexity amounts to roughly 10 kgates. The HW search engine in Figure 9 is realized reusing the parametric array architecture we proposed in [22] to calculate the SAD and MV cost functions. The implementation adopted in this work features an array of 64 basic processing elements (PE), organized as a matrix of 8×8 units, each implementing the SAD calculation at pixel level on 8-bit data plus a parallel adder tree and a module for minimum SAD and MV detection. As detailed in [22] each PE consists of a combinatorial core implementing absolute difference (AD) and accumulation operations, see Figure 10, plus registers for data propagation and synchronization within the 8×8 array. When sizing the NxN PE array a trade-off has to be found between area occupation and processing throughput while the PE array size do not limit the size of the search area or the size of the macroblock. In this paper a size 8x8 has been selected with a resulting PE array complexity of 24 kgates. The throughput is one 8×8 block matching operation, i.e. 64 SAD cost function evaluation, per clock cycle.

An overall local SRAM of 48 kbits is used to implement search area buffers (*buffer 1* and *buffer 2* in Figure 9) to reduce access frequency to large background frame memories. The SRAM is organized with two buffers, each of 18 kbits, to store search areas from different reference frames plus 6 kbits of memory for each buffer to perform efficient updates when passing from a macroblock to the spatial adjacent one (their search area overlap). Further details on spatial data reuse strategy by exploiting search area overlap for a single buffer can be found in [40]. Statistical data (mainly MVs and SADs) are stored in a smaller local memory. Finally an AHB wrapper is inserted to exchange data with an AMBA AHB-compliant bus (a defacto standard for high data-rates communication in embedded systems).

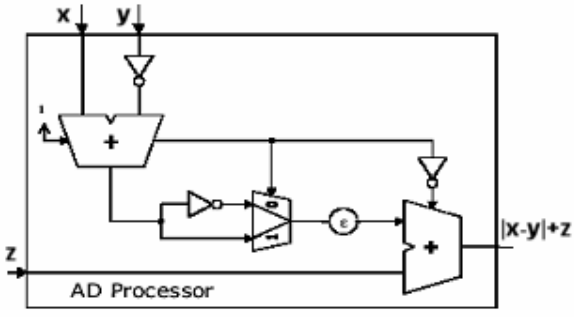

Fig. 10 Combinatorial part of each PE unit

Figure 11 sketches the operation flow for both the search engine and the controller core, counting also the number of clock cycles needed to perform bottleneck tasks, which in this case study are the load of data from frame

memory into the local buffer and MB SAD evaluations. The search engine starts performing the setup of ME of the current MB, accessing MV predictors, MB data and initial search point. If it is the first MB of a sequence also the search area relative to the first reference frame is loaded. Then, for each loaded reference frame two operations are done:

A) MB motion estimation. 16x16 partition ME is performed and results are processed by the controller (8051 Analysis I in Figure 11) to decide how to perform sub-partitions ME. Depending on the SAD it evaluates if going on with the next reference frame and which sub-blocks must be enabled for ME. The clock cycles needed for MB ME are roughly 1050 (150 for 16x6 ME and roughly 900 for sub-blocks ME in Figure 11), while other 8051 operations are done mostly in background and introduce a negligible stall (about 2%, i.e. 20 clock cycles). The number of SADs for a given MB has been evaluated counting their average number when performing ME with a UMHS adapted ME in the most difficult sequences. So Figure 11 reports the average number of cycles needed to do a MB ME in one single reference frame (determining the timing diagram of the control flow) in the case of worst reduction complexity.

B) Next Frame preload. During the ME of the current reference frame, the next reference frame is preloaded in the second buffer (buffer2) of the on-chip RAM. Depending on the decisions taken by the adaptive controller (reference frame control algorithm in Section 3) the new uploaded search area in buffer 2 can be related to: (i) the next reference frame of the current MB or (ii) the first reference frame of the next MB (no more reference frame will be processed for the current MB). Loading a search area (48x48 pixels) via a 32 bit bus takes about 576 clock cycles; as reported in the timing diagram of Figure 11 such clock cycles are in parallel to the 900 clock cycles spent for sub-block ME.

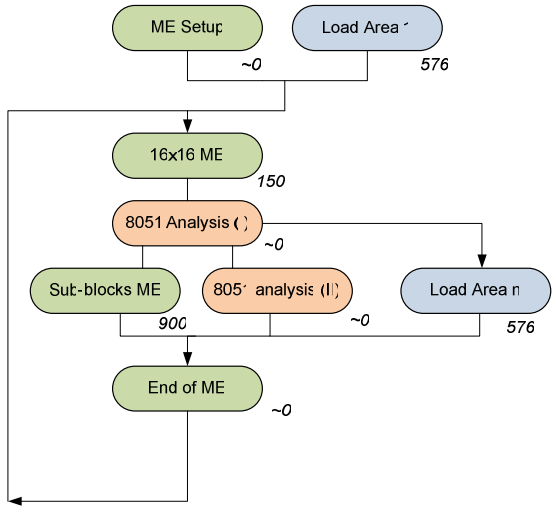

Fig. 11 Timing diagram and parallelization of operations between the 8051 IP adaptive controller and the hardware search engine

Then the maximum clock cycles needed to process one MB is about n x 1070 clock cycles being n the umber of used reference frames. Considering that the adaptive controls impose an average frame number per MB around 3 for most difficult sequences (see Table 3), the engine will perform on average the ME of a MB in 3210 clock cycles. In a CCIR format 40500 MB (30 frames each of 1350 MB) must be estimated in one second for real time performance. Considering the above calculated clock cycles this can be achieved clocking the system at roughly 132 MHz. For a 30 frames/s CIF sequence such performance can be obtained clocking at 38 MHz. In the case of a 30 frames/s CCIR format to perform real-time ME a search area must be loaded from main memory via AHB bus in 4.7 μs. That means a bandwidth of 3.8 Gbits/s, which can be obtained with a clocking speed of 132 MHz and a 32 bit AHB interface.

Sequence	Avg. Ref Frames	Avg. 16x16 MB processed each second
Stefan	2.6	9900
Mobile SIF	3.3	9900
Garden	2.8	9900
Bus	2.6	11880
Foreman	2	11880
Tennis Table	2.6	40500
Mobile CCIR	2.5	40500

Table 3 – Average number of reference frames used to encode sequences with our adaptive ME

4.2 CMOS Implementation Results

The whole ME architecture in Section 4.1 has been synthesized according to a semi-custom design flow in:

A) 0.18 μm CMOS technology using a device low-leakage standard-cells library with 1.8 V supply voltage and with 6 metal layers.

B) 65 nm CMOS technology using a device low-leakage standard cells library with 1.1 V supply voltage and 9 metal layers.

The global circuit complexity of the architecture detailed in Figure 9 amounts to roughly 35 kgates plus 48 kbits of SRAM. Both in 180 nm 1.8 V and 65 nm 1.1 V CMOS technologies the required system clock frequency (132 MHz for 30 frames/s CCIR) for real–time processing is met. The power consumption for real time processing of 30 Frame/s CCIR amounts to roughly 135 mW for the implementation in 180 nm CMOS technology and 18 mW for that in 65 nm CMOS technology. A comparison of our architecture with a recent hardware architecture proposed in literature [15] for H.264/AVC-compliant FS ME, is addressed in Table 3. The hardware engine in [15] is based on a systolic array of 256 PEs supporting variable block size FS. Reported values in Table 4 are the circuit complexity, logic gates and SRAM memory, and the clock frequency required to process in real-time video formats up to 30 frames/s CCIR using 5 reference frames and a search displacement of 16. The gain in terms of reduced circuit complexity and clock frequency, and hence power consumption, of our ME architecture vs. that in [15] to achieve real-time processing of the same format mainly depends on the computational

complexity reduction addressed at algorithmic level (see Section 3). Indeed the architecture in [15] implements a FS algorithm while our architecture implements UMHS plus the novel context-aware controller. As reported in Section 3 the efficiency loss, i.e. bit-rate increase for a fixed PSNR value, of our technique vs. FS is in average about 1.5%. A hardware implementation of the basic UMHS ME has been proposed recently in [23]. Such architecture requires a 30 MHz clock frequency to implement in real-time the UMHS technique for a 30 frames/s CIF video with a search displacement of 16. No circuit complexity results are provided in [23] for a complete comparison with the architecture proposed in this paper. However it is worth noting that our ME coprocessor, thanks to the computational saving of the adaptive parameters control, supports an UMHS-like search in real-time up to 30 frames/s CCIR format.

Sequence	Format	ΔBR%	ΔECT%
Stefan	SIF	-9.1	34
Foreman	CIF	-6.9	27.8
Akiyo	CIF	-5.8	31.6
TennisTable	CCIR	-12.2	30.3

Table 4 – ME architectures comparison

5. CABAC Algorithmic Description

CABAC [5] is the Context Adaptive Binary Arithmetic Coder used in H.264/AVC Main Profile as the entropy en-coding engine in alternative to basic CAVLC. Table 5 compares CABAC and CAVLC for different test videos when using both techniques within the JM10 encoder with the fast ME proposed in Section 3: new context-aware controller plus UMHS. Results reported in Table 5 show the % bit-rate saving due to CABAC vs. CAVLC for a given PSNR quality (ΔBR %) and the % increase in processing time (ΔECT %). The results are in line with the CABAC analysis in state-of-art [3] proving that CABAC and the new fast ME can be successfully integrated in the same encoder and that CABAC needs a dedicated implementation. The CABAC encoder structure is reported in Figure 12(a).

Work	Logic Gates	RAM	Clock Frequency	ME
Our	35 k	48 kbits (2 buffers)	132 MHz	Adaptive ME
[15]	106 k	24 kbits (1 buffer)	213.7MHz	FS

Table 5 – CABAC vs. CAVLC

Since CABAC arithmetic encoding engine works only on a binary alphabet, it requires to binarize input symbols. In fact many symbols employed in H.264 are not binary symbols (e.g. motion vectors), thus they ought to be converted in a sequence of binary symbols (*bins*). Furthermore, as CABAC is a context adaptive coder, for each bin a proper context ought to be selected among the probability models defined by the standard.

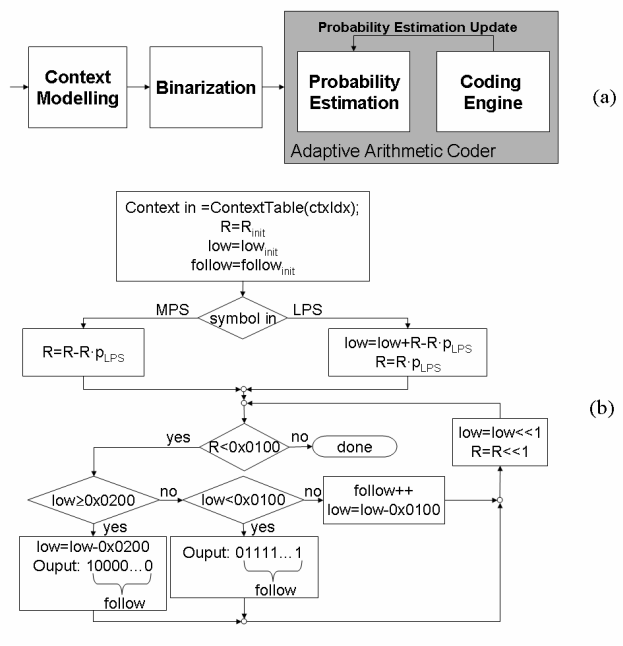

Fig. 12 CABAC structure

Then the encoding engine performs data compression while updating the probability estimation. The binarization is achieved through different techniques depending on the symbol to be binarized.

• **Unary Binarization**: it is used for unsigned syntax elements. They are represented as a sequence of '1' terminated by a '0'.

• **Truncated Unary Binarization**: it is used for a limited number of unsigned syntax elements. Given a threshold cMax, for a syntax element less than cMax, unary binarization is employed. A syntax element equal to cMax is coded as sequence of '1' with length cMax.

• **Concatenated Unary/k-th order Exp-Golomb Binarization**: it is used for signed elements. It is made of a pre-fix generated with truncated unary binarization and a suffix generated with k-th order Exp-Golomb codes.

• **Fixed length binarization**: it is used for a limited number of syntax elements whose values are integers $\in [0,cMax]$.

During the binarization a Context Identifier is assigned to each syntax element. This identifier and the current bin position, through some thresholds, generate an index (ctxIdx) that allows finding the correct context. In fact contexts are stored in a table (ContexTable) that contains the different initial probability values for the arithmetic encoder. Each context can be univocally identified, through ctxIdx.

The coding engine is based on the arithmetic encoding of a bin with its context. The arithmetic coder is binary, namely only two symbols, least probable symbol (LPS) and the most probable symbol (MPS) are allowed. The arithmetic coding is based on the recursive partition of the probability interval [0,1] in sub-intervals whose width is proportional to the probability of the symbol to be coded. Given the probabilities of the LPS (p_{LPS}) and of the MPS (p_{MPS}=1-p_{LPS}), the sub-interval widths (R_{LPS},

R_{MPS}) are updated as in (1) where R is the current interval width.

$$\begin{aligned} R_{LPS} &= R \cdot p_{LPS} \\ R_{MPS} &= R - R_{LPS} \end{aligned} \quad (1)$$

Let us introduce *low* as the lower point of the current interval, the updating of interval bounds follows the rules in (2):

$$\left.\begin{aligned} low_{new} &= low \\ R_{new} &= R - R_{LPS} \end{aligned}\right\rangle MPS$$
$$\left.\begin{aligned} low_{new} &= low + R - R_{LPS} \\ R_{new} &= R_{LPS} \end{aligned}\right\rangle LPS \quad (2)$$

6. CABAC Architecture

6.1 Architecture Description

The top level view of the proposed CABAC coprocessor is shown in Figure 13. External connection is provided through the AHB interface and two buffers are used for input and output data exchange with the network. The processing core includes four main blocks:

• **CMD decoder** dispatches the received commands from the AHB interface to the input buffer and to the CABAC control unit;

• **CABAC data path** and **control unit** manage the entropy coding, generating "packets" of bits;

• The **de-packetizer** block produces the coded bits.

A critical issue in the proposed CABAC architecture design is the FIFO sizing as it impacts both on the performance and on the complexity. In the following paragraphs the proposed architecture is detailed in order to understand the design characteristics and the system requirements. These aspects are crucial to size the FIFO reducing the complexity overhead while granting high performance as detailed in Section 6.2.

6.1.1 CMD decoder

When receiving a new input symbol, three encoding modes can be selected, according to the symbol context: *context based*, *eq_prob* or *final* mode. In the *context based* mode the context must be explicitly provided to the encoding unit, see Figure 12(b), whereas the *eq_prob* mode assumes symbol probabilities equal to 0.5 and the *final* mode allows encoding the last symbol before termination.

The CABAC control unit receives the proper command, the input symbol and the related context that are used to properly drive the data path and to produce the packed data.

Three main commands can be executed by the CABAC core:

• **Init** to initialize the probability interval and internal FSMs.

• **Terminate** to terminate the encoding on the current context.

• **Encode** to process the input symbol and to produce the output bits.

6.1.2 CABAC data path and control unit

In order to avoid the use of multiplications to perform the arithmetic coding, in H.264/AVC significant values of the interval width (R) and of the LPS probability (p_{LPS}) are pre-calculated and stored in two vectors, usually indicated as ***Q*** and ***P*** [5]; vector ***Q*** needs two bits and six bits are required for ***P***. Furthermore $R \cdot p_{LPS}$ values, obtained with ***Q*** and ***P***, are stored into a 4×64 matrix (***M***). The CABAC data path basically implements the flow chart shown in Figure 12 (b): it receives the *context in* value from the CABAC control unit and produces the packed data that are pushed into the FIFO. Furthermore the control unit ought to correctly drive the CABAC data path multiplexers in order to select the proper values.

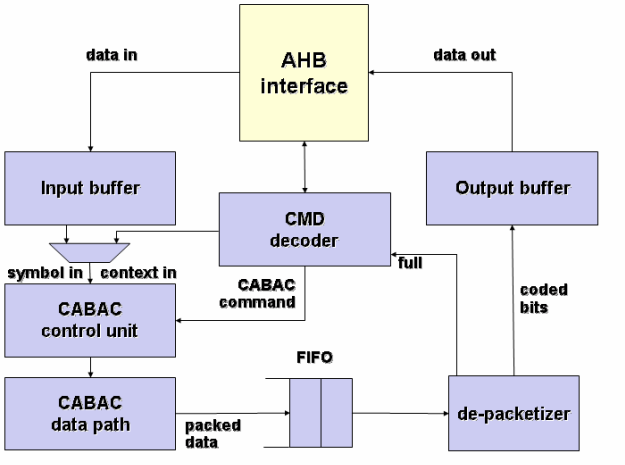

Fig. 13 Proposed CABAC coprocessor

The CABAC data path block scheme is depicted in Figure 14 highlighting four functional units, respectively devoted to probability updating, computation of $R \cdot p_{LPS}$ product, range update and data output. Since the probability p_{LPS} is derived from the context State, in the actual implementation ROM memories are directly addressed by context State, so avoiding the generation of the intermediate p_{LPS} value.

In Figure 14, instead of allocating a single ROM memory with pre-calculated $R \cdot p_{LPS}$ product values, the context State is used to address the four small ROMS M0 to M3 (64×9 bits), while the current R value selects the correct ROM output through a multiplexer; this solution better exploits the delayed arrival of R, which is available one cycle later than the context.

The CABAC coder is adaptive and therefore the context is continuously updated by means of an FSM that generates the *Next context State* on the basis of the received symbol (LPS or MPS) and the current *Context State*. In Figure 14, two small ROMs (64×6 bits) are used

to adaptively generate *Next MPS State* and *Next LPS State*, depending on the input symbol.
The range update unit in Figure 14 simply computes either R_{LPS} or R_{MPS} according to Equation (2); moreover this unit also updates *low* value.
In the data output unit, the *low* value is used to update the output bit according to the flow chart depicted in Figure 12 (b). When *low* is greater than half the interval width (0x0200) a '1' is output. Similarly when *low* is lower than a quarter of the interval width (0x0100) a '0' is generated. Otherwise no coded bits can be produced: in this case, the generated output is not valid and the *follow* counter is incremented; the next valid generated output bit will be accompanied by the number of occurred increments, stored as the *follow* content. Performed simulations on different sequences show that the number of increments in counter *follow* is lower than 64 and 6 bits are enough [9]. Therefore the data output by the CABAC data path is a packet composed by three parts: *output bit*, *output valid* and *follow* value. Extensive simulations show that, depending on the *low* value, no more than seven packets per input bit can be produced by the CABAC data path.

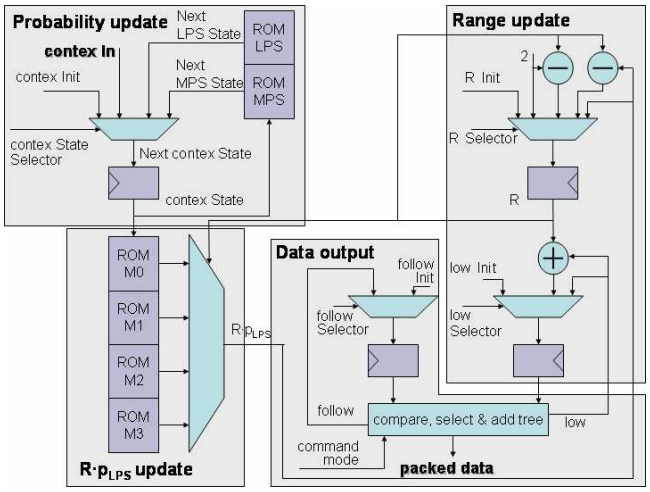

Fig.14 CABAC data path block scheme

6.1.3 De-packetizer

Since the data path requires three clock cycles to produce a new set of packed data, a de-packetizer is required in the CABAC coprocessor (see Figure 13) to obtain the coded bit-stream. Figure 15 details the architectural block diagram of the de-packetizer. This processing block receives packets from the CABAC data path through a FIFO and performs two simple operations. First it checks the *output valid* bit to verify whether a packet is significant or not. Then, if the packet is significant, a 32 bit register is loaded with the *output bit* followed by a sequence of '0' or '1' according to the value of *output bit*; the sequence length is set by the value of the *follow* field of the packet. In order to correctly load the bits into the 32 bit register, a start pointer (ptr_i) and an end pointer (ptr_e) are required. Registers and adders shown in Figure 15 allow updating the two pointers while packet data are received. When the 32 bit register is full, it is written into the output buffer. A counter allows signaling to the command decoder that the output buffer is full. The architecture works at full throughput if a new input symbol is available every clock cycle. In this case every command is executed in 3 clock cycles. If full throughput is not granted, the core requires one additional clock cycle to verify when a new input symbol is ready.

6.2 FIFO Sizing

The number of clock cycles needed to process symbols is affected by the sizing of *follow* register. In fact if a sufficiently high value cannot be represented, stall cycles are introduced and more than 3 clock cycles are required to complete the current compression step. Sizing of the *follow* register also impacts on the size of the FIFO.

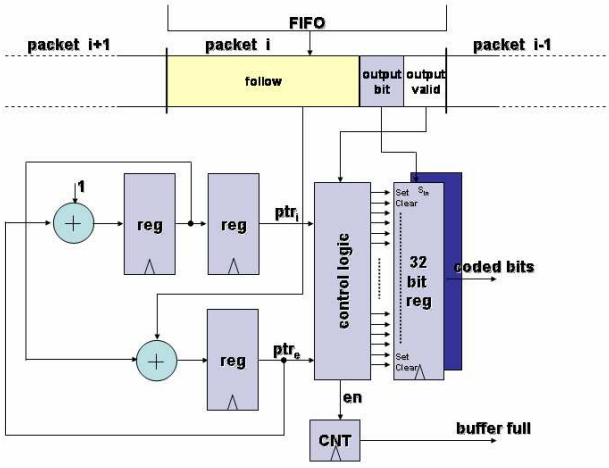

Fig. 15 De-packetizer block scheme

The relation between the FIFO data width and the *follow* register size is expressed in (3):

$$n_{packets} \cdot \left(nbit_{follow} + 2\right) \leq n_{cycles} \cdot nbit_{fifo} \qquad (3)$$

Within the values described in the previous paragraphs, we can set-up $n_{packets}$=7, $nbit_{follow}$=6 and n_{cycles}=3, that leads to $nbit_{fifo} \geq 19$. Even if the FIFO data width is minimized when $nbit_{fifo}$=19, this choice has a negative impact on the de-packetizer architecture. In fact with $nbit_{fifo}$=19, a non integer number of packets can be accommodated in the FIFO data. In order to simplify the architecture and to accommodate an integer number of packets, we accept an increase in the FIFO data width. With a 24 bit wide FIFO, 7 packets can be transferred in 3 clock cycles. On the other hand, sizing the FIFO depth depends on both the distribution of *follow* values and on the distribution of valid packets. If the number of free positions in the 32 bit register is larger than follow, $n_{free} \geq follow+1$, then 2 clock cycles are required to process a packet; otherwise 3 or 4 clock cycles are required depending on the *follow* value. Adding two 32 bit shadow registers, the de-packetizer can always process a packet in 2 clock cycles.

However, even if 6 bits for representing the *follow* value is a conservative sizing, our simulations show that *follow* has a low probability of achieving values greater than 32. As a consequence, instead of using three 32 bit registers (2 shadow registers), only two 32 bit registers can be employed (1 shadow register); the effect of *follow* values greater than 32 can be absorbed by the FIFO. In fact, since the *follow* most probable values are in the [0,31] range, most packets are processed in 2 clock cycles, and a few of them in 3 clock cycles. As a significant example, in Figure 16 the distribution of the *follow* register content for 3 frames of the Foreman sequence is shown. It is worth pointing out that the distribution of valid packets has a stronger impact on the FIFO depth. In particular, the distance between consecutive valid packets is critical. When several consecutive valid packets are generated, a deeper FIFO is required. On the other hand even if seven consecutive valid packets are generated, but they are followed by a reduced number of valid packets, few registers can act as a buffer. Unfortunately the distance between consecutive valid packets can be determined only from the statistical analysis of real sequences. Simulations on several sequences show that less than 1024 cells are required for buffering the packets in the worst case, As a consequence a 25 kbits FIFO (1024×24) can be employed to achieve high throughput. In Figure 17, the growing of the required FIFO for 3 frames of the Foreman sequence is shown as a significant example. As it can be observed in this case the maximum number of required cells is 823 and they can be accommodated in the 1024 cells FIFO.

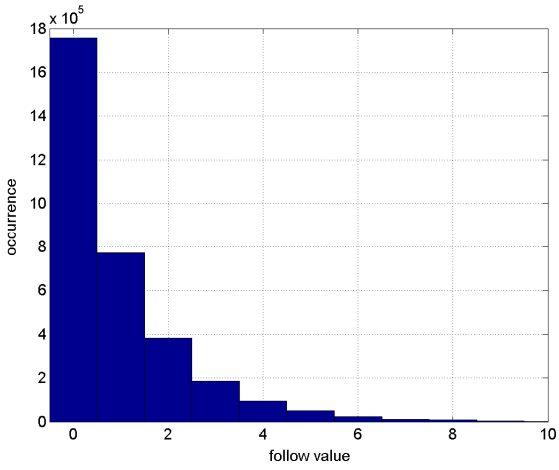

Fig. 16 Distribution of the follow value for the Foreman sequence: the greatest value is 10 but with a very small probability

It is worth pointing out that the FIFO sizing allows a complexity/performance trade-off. Reducing the FIFO size the architecture complexity is decreased at the expense of reduced performance. In fact reducing the FIFO size can cause a stall in the CABAC data path or in the de-packetizer.
As a consequence the number of clock cycles required to complete the arithmetic coding operation increases, reducing the architecture throughput.

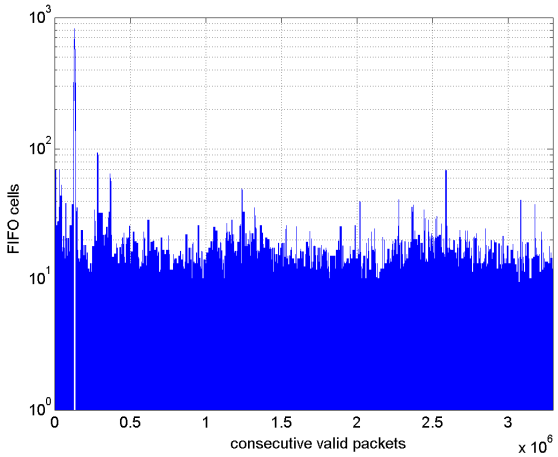

Fig. 17 Required number of cells for packet buffering during the elaboration of 3 frames from the Foreman sequence

6.3 CMOS Implementation Results

The VHDL model developed for the proposed CABAC processor has been synthesized on the aforesaid 0.18 µm and 65 nm CMOS standard-cells technologies. According to post synthesis results, up to 333 MHz clock frequency is achieved for both technologies, with an occupation of about 4.5 kgates for the logic and with about 25 kbits for the FIFO. Thus the proposed architecture is able to sustain an incoming rate of up to 111 Mbits/s. Down-scaling the clock frequency to 150 MHz, a data-rate of 50 Mbits/s can be sustained with a power consumption of about 22 mW and 3 mW in 0.18 µm and 65 nm technologies respectively. It is worth pointing out that an uncompressed 720×480 video sequence at 30 frames/s produces from 83 to 125 Mbits/s (8 bits gray scale or YUV 4:2:0). Thus, on the target application (720×480 at 30 frames/s), the proposed CABAC architecture can sustain also very low compression ratios (~2:1), granting high quality video compression.
Compared to the solutions described in [8, 9, 18, 19, 20, 36] the proposed architecture shows some common points and some differences. In particular, since in [18] FPGA implementation is considered, a fair comparison is not possible, even if an ASIC complexity of roughly 10 kgates is estimated. This value appears compatible with our architecture complexity. On the other hand we can compare the proposed architecture with [8, 9, 19, 20, 36]. The architecture described in [8] is similar to the one proposed in this work, however in [8] the sizing of the FIFO is not investigated. Moreover, in [8] a 0.35 µm standard cell technology is employed leading to a limited clock frequency (186 MHz) and throughput. Since in [8] also binarization is taken into account a fair complexity comparison is not possible.
To the best of our knowledge the architecture described in [9] is one of the highest performance implementations available in the literature. Comparing the architectures described in [9, 20, 36] that are all implemented on a

0.18 μm technology, we can observe that [9] has a higher complexity and can run at 263 MHz sustaining 87 Mbits/s, whereas [20] achieves 4 Mbits/s with a clock frequency of 190 MHz. The solution in [36] is based on fully pipelining all the encoding process and achieves a throughput of an encoded symbol per clock cycle at the expense of 14 kgates for the logic and 15 kbits of RAM.
In Table 6 we compare the proposed architecture with [9, 19, 36]. For a fair comparison the proposed architecture results are shown only for the 0.18 μm standard cell technology. As it can be inferred the proposed architecture has a lower critical path with near the same complexity of [9]. The comparison with [19] is not completely fair as our single-symbol architecture is compared to [19] which is a multi-symbol encoder. The proposed architecture is smaller than [19], with a reduced critical path. However, since the proposed architecture requires three clock cycles to elaborate an input symbol, the number of processed symbols per cycle is lower than [19]. Finally, compared with [36] the proposed architecture requires less logic but more memory, in fact [36] requires 14 kgates for the logic and 15 kbits for the memory respectively, whereas the proposed architecture features 4.5 kgates for the logic and 25 kbits for the memory

	0.18 μm area		**Critical path [ns]**	**Symbols per cycle**
	[kgate]	**[mm^2]**		
[9]	-	0.423	3.8	1/3
[19]	32.1	-	5.2	3.32
[36]	15	-	3.2	1
Our	4.5	0.513	3	1/3

Table 6 – CABAC architectures comparison

7. IP Interface for NoC Parallel Computing

As proved in literature [27, 32] since the H.264 standard is computing intensive the requirements of real time performance can be met not only through complexity reduced new algorithms but also through parallel hardware configuration by means of innovative communication infrastructures such as NoC. Motion Estimation operations are suitable to follow a parallel operation flow since many kind of searches (e.g.: searches for different partitions or different MBs) can be done independently and thus parallelized. As proved in [27], which analyzes an MPEG encoder architecture varying the number of ME processors used, a NoC communication approach outperforms classic point-to-point and hierarchical bus interconnections in terms of scalability for area, power and throughput.
Therefore is important an evaluation of the complexity of a Network Interface (NI) able to connect the proposed CABAC and ME IPs to a NoC. Among the several NoC topologies proposed in literature here we refer to the Spidergon STNoC topology [26, 33]. In our case study, application of NoC solutions on video encoders, the NoC capabilities can be reduced to basic networking/routing schemes with appropriate NI towards the AMBA AHB IP bus. So Spidergon STNoC building blocks need to just implement basic functionality to guarantee efficient parallel communication between cores and memory spaces. The NI converts protocols, data size and frequency between the IP domain and the NoC domain. Once the NI injects packets on the network, 4-port Routers (R) apply routing schemes across the network, whose topology is the same as the Ring one except that one more port connects opposing routers (Figure 18). The architecture of the NI we have designed is organized in two main modules, see Figure 19. The first one, the Shell, implements protocol handshaking with the connected IP and abstracts the transport and network layers. The second one, the Kernel, manages packet assembling and, depending on the configuration, also frequency and data size conversion. A NI Shell has been designed to adapt the AHB bus interface of the two H.264/AVC blocks to the Kernel module. The whole NI has been then configured in two ways depending on the wished interconnect capabilities, as reported in Table 7. Then it was synthesized in a standard cells CMOS 65 nm 1.1V and CMOS 180nm 1.8V processes. The parameters that have been configured are:
- The number of retiming stages in the pipeline, determining the clock latency.
- Frequency conversion scheme, patent filed [39], based on bi-synchronous FIFOs with Gray coded pointers and brute force synchronizers to transmit payload and header data through separate clock domains.
- The data width at AHB and NoC interfaces.
- The amount of internal buffering for payload and header data.

Feature	**Config 1**	**Config 2**
Latency	0 clock cycles	1 clock cycle
Frequency conversion	off	on
Header FIFO	no	240 bits
Payload FIFO	no	288 bits
NoC data width	32 bits	64 bits
AMBA AHB	32 bits	32 bits

Table 7 – NI synthesis configurations for video coding computations

A 0 clock cycle NI with Configuration 1 in Table 7 can work up to 500 MHz clock frequencies in 65 nm (using a low-leakage library version), which is well above the working frequencies of the proposed CABAC and ME H.264/AVC building blocks. Therefore frequency conversion is not strictly necessary and a single clock IP-NoC design can be done. In case of Configuration 2 a FIFO-based frequency conversion is enabled in the case the NoC works at maximum speed. The complexity in terms of kgates is roughly 1.5 for Configuration 1 and 3 kgates for Configuration 2 with leakage power consumption of few tens of μW. Configuration 2 performs also a size conversion packing 32 bit AHB data into 64 bit payload data carried by the network (the small FIFOs are realized using Flip-Flops and hence no embedded RAM is needed).

Comparing such data with the complexity of ME and CABAC cores it can be seen that the overhead of a NI, particularly configuration 1 in Table 7, is minimal. The maximum achievable bandwidth, if the NoC is clocked at 500 MHz, is 15 Gbits/s; such value is much higher than the one needed by the ME engine to transfer pixel data from background frame memories to the local buffers (~3.8 Gbits/s, see Section 4.1). The NoC bandwidth also sustains CABAC throughput which is roughly 0.11 Gbits/s (see Section 6). The whole throughput requirement is then 3.9 Gbits/s and can be sustained clocking the NoC at 125 MHz, a target frequency achievable also in 180 nm 1.8 V technology.

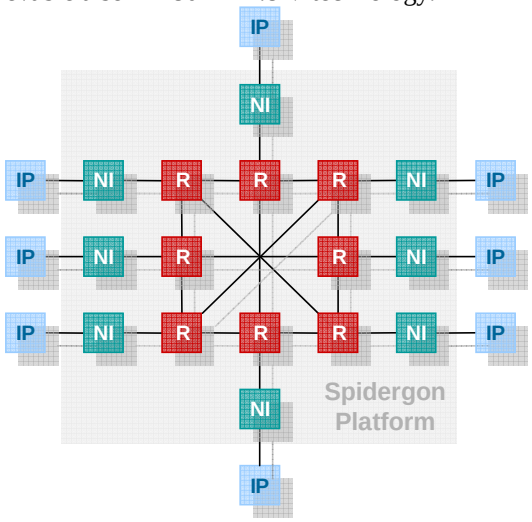

Fig. 18 – Spidergon STNoC topology

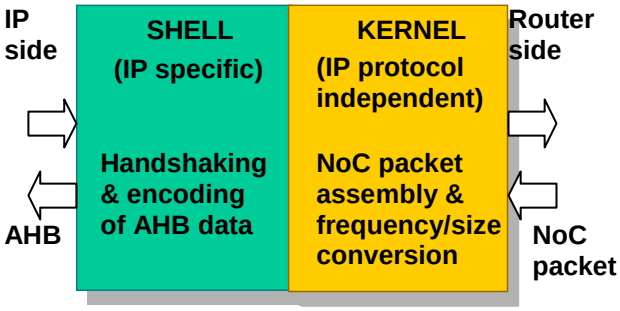

Fig. 19 – NI subdivision in Shell and Kernel

8. Conclusions

In this paper two optimized hardware co-processors for real-time and high-quality H.264/AVC video coding are proposed. Stemming from complexity and profiling analysis the two architectures are derived in a NOC framework as AHB interfaced IP cells for high-performance H.264/AVC-based VLSI multimedia systems. CMOS implementation results highlighted that both the co-processors have interesting figures in terms of performance, complexity and power consumption also compared to other published solutions. The design of a Network Interface is also presented to allow direct integration of the IP cores in advanced on-chip communication infrastructures, NoC, used in MPSoC for video encoding.

This work has been supported by NEWCOM++ NoE.

References

[1] T. Wiegand, G. Sullivan, G. Bjntegaard, A. Luthra, “Overview of the H.264/AVC video coding standard”, IEEE Tran. on Circuits and Systems for Video Tech., vol. 13, n. 7, 2003, pp. 560-576

[2] S. Saponara et al., “Performance and complexity co-evaluation of the Advanced Video Coding standard for cost-effective multimedia communications”, J. Applied Signal Processing, vol. 2, 2004, pp. 220-235

[3] J. Ostermann et al., “Video coding with H.264/AVC: tools, performance and complexity”, IEEE Circ. and Syst. Magazine, vol. 4, 2004, pp. 7 – 28

[4] S.Y. Chien, Y.W. Huang, C.Y. Chen, H.H. Chen, L.G. Chen, “Hardware architecture design of video compression for multimedia communication systems”, IEEE Comm. Mag., 2005, pp. 123-131

[5] D. Marpe, H. Schwarts, T. Wiegand, “Context-based Adaptive Binary Arithmetic Coding in the H.264/AVC video compression standard”, IEEE Trans. on Circuits and Systems for Video Tech., vol. 13, July 2003, pp. 620–636

[6] L. H.-Yao, C.Y.-Chih, C. C.-Hong, L. B.-Da, Y. J-Ferr, “Combined 2-D transform and quantization architectures for H.264 video coders”, Proc. IEEE ISCAS, 2005, pp. 23-26

[7] V. H. S. Ha, W. S. Shim, J. W. Kim, “Real-time MPEG-4 AVC/H.264 CABAC entropy coder”, Proc. IEEE Int. Conf. on Consumer Electronics, 2005, pp. 255–256

[8] R. Osorio, J. Bruguera, “High-Throughput Architecture for H.264/AVC CABAC Compression System”, IEEE Trans. On Circuits and Systems for Video Technology, vol. 16, n. 11, Nov. 2006, pp. 1376–1384

[9] H. Shojania, S. Sudharsanan, “A high performance CABAC encoder”, Proc. IEEE NEWCAS, 2005, pp. 19–22

[10] Y. Huang et al., “Analysis, fast algorithm, and VLSI architecture design for H.264/AVC intra frame coder,” IEEE Trans. Circuits Sys. Video Tech., vol. 15, n. 3, Mar. 2005, pp. 378–401

[11] Z. Chen, J. Xu, Y. He, “Efficient fast ME predictions and early-termination strategy based on H.264 statistical characters”, Proc. ICICS – PCM, Dec. 2003, pp. 213 -218

[12] H. Tourapis, A. Tourapis, “Fast motion estimation within the H.264 codec”, Proc. IEEE ICME, July 2003, pp. 517-520

[13] P. Kuhn, Algorithms, complexity analysis and VLSI architectures for MPEG-4 motion estimation, Kluwer, 1999

[14] L. Fanucci et al., “Self-adaptive algorithmic/architectural design for real-time, low-power video systems”, IEICE Tran. on Inf. and Systems, n. 7, vol. E88-D, pp. 1538-1545, 2005

[15] Y.W. Huang et al., “Hardware architecture design for variable block size motion estimation in MPEG-4 AVC/JVT/ITU-T H.264”, Proc. IEEE ISCAS, 2003, pp. 796-799

[16] http://iphome.hhi.de/suehring/tml

[17] JVT and ITU-T, “Draft ITU-T recommendation and final draft international standard of joint video specification (ITU-T Rec. H.264 — ISO/IEC 14496-10 AVC)

[18] J. L. Nunez-Yanez, V. A. Chouliaras, D. Alfonso, “Hardware assisted rate distortion optimization with embedded CABAC accelerator for the H.264 video codec”, Proc. IEEE Int. Conf. on Consumer Electr., 2006, pp. 95–96

[19] Y.J. Chen, C.H. Tsai, L.G. Chen, “Architecture Design of Area-Efficient SRAM-Based Multi-Symbol Arithmetic Encoder in H.264/AVC”, Proc. IEEE ISCAS, 2006, pp. 2621-2624

[20] O. Flordal et al., “Accelerating CABAC Encoding for Multi-standard Media with Configurability”, Proc IPDPS 2006, pp. 1-8

[21] L. Fanucci, S. Saponara, “Power optimization of an 8051-compliant microcontroller”, IEICE Trans. Electronics, vol. E88-C, n. 4, April 2005, pp. 597-560

[22] L. Fanucci, S. Saponara, L. Bertini, “A Parametric VLSI Architecture for Video Motion Estimation”, Integration-The VLSI Journal, n. 1, vol. 31, pp. 79-100, 2001

[23] C. Rahman, W. Badawy, "UMHexagonS Algorithm Based Motion Estimation Architecture for H.264/AVC", IEEE IWSOC 2005, pp. 207-210
[24] S.Saponara, M.Casula, F.Rovati, D.Alfonso, L.Fanucci, "Dynamic control of motion estimation search parameters for low complex H.264 video coding", IEEE Transactions on Consumer Electronics, vol. 52, n. 1, pp. 232-239, February 2006
[25] L. Benini, G.De Micheli, "Networks on Chip: a New SoC Paradigm", IEEE Computer, vol. 35, n. 1, pp. 70-78, Jan 2002
[26] M.D. Grammatikakis, M. Coppola, G. Maruccia, R. Locatelli, L. Pieralisi, "Design of Cost-Efficient Interconnect Processing Units: Spidergon Stnoc", *CRC Press, 2008*
[27] H. Gyu Lee et al., "On-Chip Communication Architecture Exploration: A Quantitative Evaluation of Point-to-Point, Bus, and Network-on-chip approaches", ACM Trans. on Design Automation of Electronic Systems, vol. 12, n. 3, Aug. 2007
[28] D. Kim et al., "MPEG-4 Performance Analysis for a CDMA Network-on-Chip", Proc. of International Conference on Comm., Circuits and Systems, vol. 1, pp 493-496, May 2005
[29] A. Kumala, E. Salminen, M. Hannikainen, T.D. Hamalainen, "Evaluating SoC Network Performance in MPEG-4 Encoder", SIPS 2006, pp. 250-255, Oct. 2006
[30] S. Dutta, W. Wolf, "A Flexible Parallel Architecture Adapted to Block-Matching Motion-Estimation Algorithms", IEEE Tran. on Circ. and Systems for Video Tech., vol. 6, n. 1, pp. 74 – 86, 1996
[31] V.-D. Ngo, H.-N. Nguyen, H.-W. Choi, "Realizing Network on Chip Design of H.264 Decoder Based on Throughput Aware Mapping", ICCE'06, pp. 337-342, Oct. 2006
[32] J. Chang et al, "Performance Analysis for MPEG-4 Video Coded Based on On-Chip Network", ETRI Journal, vol. 27, n. 5, 2005
[33] F. Vitullo et al., "Low-Complexity Link Microarchitecture for Mesochronous Comunication in Networks on Chip", IEEE Transactions on Computers, vol. 57, n. 9, pp 1196-1201, 2008
[34] K. Srinivasan, E. Salminen "A Methodology for Performance Analysis of Network-on-Chip Architectures for Video SoC", 2009, available at http://www.design-reuse.com/articles/20623/
[35] S. Saponara, L. Fanucci, P. Terreni, "Architectural-level Power Optimization of Microcontroller Cores in Embedded Systems", IEEE Tran. On Ind. Electr., vol. 54, n. 1, pp. 680-683, Feb. 2007
[36] W. Zheng, D. X. Li, B. Shi, H. S. Le, M. Zhang "Efficient Pipelined CABAC Encoding Architecture", IEEE Trans. on Consumer Electronics, vol. 54, n. 2, May 2008, pp. 681-686
[37] P. Zhang, D. Xie, W. Gao, "Varriable-Bin-Rate CABAC Engine for H.264 High Definition Real Time Decoding, IEEE Trans. on VLSI, vol. 17, n. 3 , Mar. 2009, pp. 417-426
[38] A. Molino, F. Vacca, G. Masera, T. Q. Nguyen, "Scalable phase extraction methods for phase plane motion estimation", IEE Proc. – Vision, Image and Signal Proc., vol. 153, n. 6, 2006, pp. 860-868
[39] G. Maruccia, S. Saponara, L. Fanucci M. Casula, R. Locatelli, L. Pieralisi, M. Coppola, "Method for transferring a stream of at least one data packet between first and second electronic devices and corresponding device", Europe, EP 07121139.5, 2007
[40] S. Saponara, L. Fanucci, **"**Data-adaptive motion estimation algorithm and VLSI architecture design for low-power video systems", IEE Proceedings-Computers and Digital Techniques, vol. 151, n. 1, Jan. 2004, pp. 51-59