HARDWARE CO-PROCESSORS FOR REAL-TIME AND HIGH-QUALITY H.264/AVC VIDEO CODING

M. Martina[#], *G.. Masera*[#], *L. Fanucci*[+], *S. Saponara*[+]

[+]Dip. Ingegneria della Informazione, Università di Pisa, 56122, Pisa, Italy, {l.fanucci,s.saponara}@iet.unipi.it
[#]CERCOM – Dip. di Elettronica, Politecnico di Torino, I-10129, Torino {maurzio.martina,guido.masera}@polito.it

ABSTRACT

Real-Time and High-Quality video coding is gaining a wide interest in the research community, mainly for entertainment and leisure applications. Furthermore H.264/AVC, the most recent standard for high performance video coding, can be successfully exploited in such a critical scenario. The need for high-quality imposes to sustain up to tens of Mbits/s. To that purpose in this paper optimized architectures for H.264/AVC most critical tasks, Motion Estimation (ME) and Context Aware Binary Arithmetic Coding (CABAC) are proposed. Post synthesis results on a 0.18 µm standard cells technology show that the proposed architectures can actually process in real time 720x480 video sequences at 30 Hz and grant more than 20Mbits/s in the simplest configuration.

Keywords: *Video coding, H.264/AVC, Hardware architectures, motion estimation, entropy coder*

1. INTRODUCTION

H264/AVC is the new video coding standard released by ITU-T and ISO/IEC. Compared to previous standards, H.264/AVC superior perceptual quality and high scalability, make it suitable for different scenarios. The implementation of hardware co-processors, able to sustain real-time and high quality H.264/AVC video coding, is particularly relevant to grant high performance. Figure 1 shows a block diagram of the H.264/AVC encoding scheme. With respect to previous coding standards, H.264/AVC includes additional features, particularly in the Motion Estimation (ME) task, adopting multi-reference frames and variable block sizes, and in the Entropy Coding (EC) task, adopting a Context Adaptive Binary Arithmetic Coder (CABAC). A performance and complexity profiling analysis on the C-level model of the coder proves that these features improve the coding efficiency by a factor two at the expense of an increased implementation cost (computation and memory) by one order of magnitude [1,2]. Hence the design of hardware co-processors for ME and CABAC is mandatory. Two dedicated architectures are presented in the paper allowing for real-time implementation of H.264/AVC video coding. These architectures are well suited for high quality scenarios where up to tens of Mbits/s are reached, as in the Main Profile of the standard.

In the literature several works have been proposed concerning the implementation of single blocks of the H.264/AVC standard. In [3] H.264/AVC integer transform implementation is addressed. Few recent works concern the CABAC implementation: in [4] and [5] mixed HW/SW systems are proposed, whereas [6] concentrates on a CABAC coprocessor. Many fast ME engines have been proposed in literature [7-11] to reduce the complexity of conventional Full Search (FS). Among them UMHexagonS [7] has been officially accepted as the standard fast ME solution in the JM reference software model [12,13]. It realizes a predictive search which adopts a hexagonal window in the refining phase plus proper stop criteria. In most of known motion estimation algorithms, the basic search is repeated multiple times.

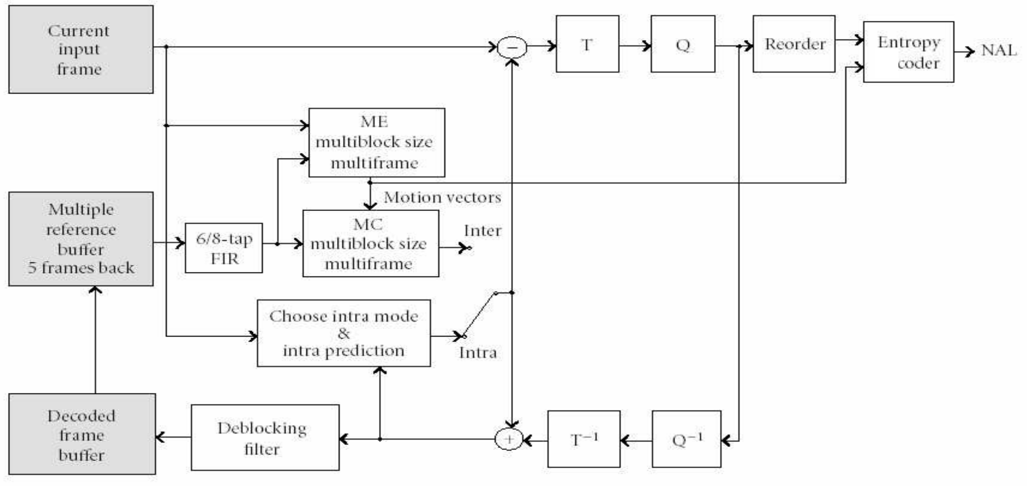

Figure 1. Block diagram of the H.264/AVC encoding scheme

This is critical in case of multiple reference frames or variable block sizes. Since ME operations increase with the number of blocks and reference frames, unnecessary redundancy is introduced in computations and memory accesses.

It is worth pointing out that this paper concentrates on the whole H.264/AVC framework and deals with the most computationally intensive tasks, showing architectures suited for real-time, high-quality video coding. As far as CABAC is concerned a modular implementation has been developed in order to grant an incoming rate scalable with the number of CABAC cores employed. For ME an adaptive algorithm with its relevant hardware architecture is proposed. The novel technique avoids unnecessary computations and memory accesses, whereas it allows the same high coding quality of FS. Hereafter Section 2 deals with CABAC and ME algorithmic description. Relevant hardware architectures are described in Section 3. Conclusions are drawn in Section 4.

2. ALGORITHMS DESCRIPTION

2.1 CABAC

CABAC [14], whose structure is reported in Figure 2, is the Context Adaptive Binary Arithmetic Coder used in H.264 as the entropy encoding engine. It can be employed in the Main Profile to improve the coding efficiency with respect to the Context Adaptive Variable Length Coding (CAVLC). In fact, as proved in [14], for the range of acceptable video quality for broadcast applications (about 30-38dB) bit-rate savings of 9% to 14% can be achieved.

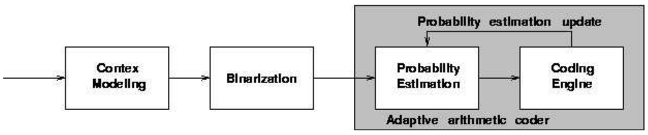

Figure 2. CABAC structure

Since CABAC arithmetic encoding engine works only on a binary alphabet, it requires to binarize the 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. Then the encoding engine performs data compression while updating the probability estimation (see Figure 2). The binarization is achieved through different techniques depending on the symbol to be binarized.

- **Unary Binarization (U):** it is used for unsigned syntax elements. They are represented as a sequence of '1' terminated by a '0'.
- **Truncated Unary Binarization (TU):** it is used for a limited number of unsigned syntax elements. Given a threshold *cMax*, for a syntax element less than *cMax*, U is employed. A syntax element equal to *cMax* is coded as a sequence of '1' with length *cMax*.
- **Concatenated Unary/k-th order Exp-Golomb (UEGk) Binarization:** it is used for signed elements. It is made of a prefix generated with TU and a suffix generated with k-th order Exp-Golomb codes.
- **Fixed length binarization (FL):** 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 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. As the arithmetic coder is binary, only two symbols are allowed, namely the least probable symbol (LPS) and the most probable symbol (MPS). 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-intervals width (R_{LPS}, R_{MPS}) can be updated as

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

where R is the current interval width. Let's introduce *low* as the lower point of the current interval, it holds true that:

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

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

To avoid the use of multiplications to perform the arithmetic coding, in H.264 significant values of the interval width (R) and of the LPS probability (p_{LPS}) are pre-calculated and stored in two vectors, called ***Q*** and ***P***. Furthermore $R \cdot p_{LPS}$ values, obtained with ***Q*** and ***P***, are stored into a 4x64 matrix (***M***) [14]. Given the current interval width and the current LPS probability, a finite state machine (FSM) manages the transitions on the ***M*** matrix values; this FSM will be referred as FSM_M. Furthermore to avoid the interval to become too small some renormalizations are employed.

2.2 Variable block size, multi frames ME

At algorithmic level we propose to add a low complexity context aware controller to basic ME search engines, FS or Fast technique as UMHexagonS. The controller extracts from the search engine some partial results: 1) Motion Vectors (MV), 2) Sum of Absolute Difference (SAD) cost, 3) information on the input signal statistic. Then the controller uses them to automatically configure the ME search parameters: number of reference frames, valid block modes and search area for each 16x16 block and its sub-partitions down to 4x4-pixel blocks. The global control combines three basic algorithms:

A) The Search Area Control, originally proposed for a FS engine in [10]. The optimal search size for the block under estimation is derived by comparing with proper thresholds the SAD and MV values of already encoded neighbouring blocks: 3 spatial and 1 temporal. In this paper the same control has been successfully applied to UMHexagonS.

B) The Modes Control. Profiling analysis of the standard 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. The control over smaller block sizes (4x8, 8x4 and 4x4 partitions) decides which of them must be enabled for ME each time a 16x16 block is encoded. Moreover it accomplishes its task by comparing the SAD cost of the current 16x16 partition with two thresholds. Depending on the results of the comparison the ME will continue using other 6, 5 (avoiding 4x4) or 3 (avoiding 4x4, 4x8 and 8x4) block sizes.

C) The Frame Control, which decides the maximum number of reference frames to be used for the ME of a 16x16 block and its selected subpartitions. The data (SAD cost, MV and optimal reference frame) of the already encoded 16x16 partition are used to decide how many reference frames are useful: for the enabled smaller sub partitions, for the same 16x16 partition in the next frame.

The encoding process, using the three controls is accomplished according to this processing flow: (i) the optimal search area and reference frame number for the 16x16 block are preliminarily sized using the algorithms in **A)** and **C)**. (ii) The basic search engine, UMHexagonS or FS, performs the ME for the 16x16 partition. (iii) using data (MV, SAD value and optimal reference frame) from the previous operation the controls in **B)** and **C)** decide which sub partitions must be enabled for ME and how many reference frames must be used for their search. The search size is the same derived for the 16x16 partition.

Table 1 compares our control applied to UMHexagonS vs. conventional FS: our technique allows for a complexity reduction of two orders of magnitude with an average bit-rate loss below 1%. Results are expressed as % changes of bit-rate for a given PSNR quality (ΔBR%) and of ME processing time (ΔMET%) when integrating our controller into the JM model and running it on a AMD 2.4+ processor.

Figure 3 compares for the Tennis CCIR video the JM9 encoder with FS and the JM9 encoder with UMHexagonS plus our controller in terms of absolute PSNR and bit-rate values. The same high coding quality of FS is kept unaltered for bit-rate applications up to 55 Mbits/s.

	Stefan SIF	**Tempete CIF**	**Coastguard QCIF**	**Foreman CIF**	**Akiyo CIF**
ΔMET%	-93,98	-95,35	-95,88	-96,48	-99,53
ΔBR%	1,01	1,57	0,1	1,54	-0,75

Table 1 –UMHexagonS with all three controls vs. FS

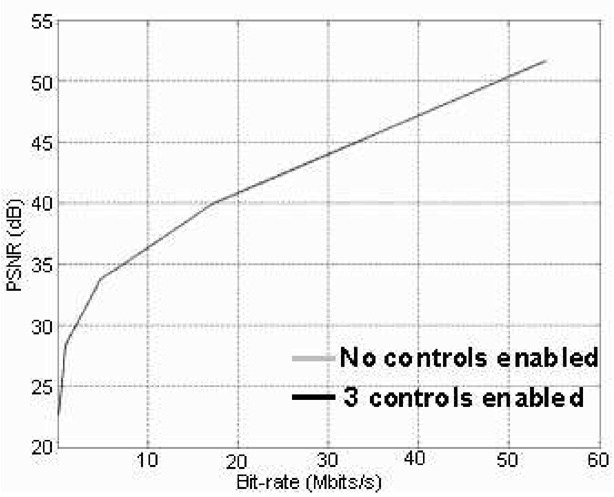

Figure 3. Rate-distortion curve for Tennis CCIR

3. COPROCESSORS ARCHITECTURES

3.1. CABAC coprocessor

This section describes the most critical aspects to implement a CABAC coprocessor.

First, analyzing in detail the JM reference software model [12], it has been observed that most of the encoding time is required by the *Encode Decision* and *Encode Bypass* routines (roughly 20% of the CABAC processing time). Moreover, since the value $R\cdot p_{LPS}$ depends on R, an *As Late As Possible* (ALAP) strategy can be employed, as suggested in [5]. In fact R is quantized on only 4 values (vector $\boldsymbol{Q}$ contains only 4 elements), the 4 corresponding $R\cdot p_{LPS}$ values can be read together from a memory (where the FSM_M transitions are stored) and loaded into 4 registers. Then the right value can be selected based on the correct R value. Furthermore since the arithmetic coder produces a variable number of output bits, the output register needs to be carefully designed. Based on a simulative approach a 48 bits output register has been employed as detailed in the following.

The processing blocks shown in Figure 4 have been developed with a modular design methodology. The architecture is composed of a main control unit, *EC CU* in Figure 4, with a sixteen states FSM devoted to send the proper start signal and commands to the different CABAC encoder blocks. Two simple blocks, namely *Init FSM* and *CTX*, are enabled by the *EC CU*. The former is devoted to send the proper initial probability values to FSM_M. The latter is made of two small RAMs devoted to store, for each context, the MPS and the current state of the FSM that manages symbol probabilities. The computation part of the proposed architecture is made of a ROM where the FSM_M transitions are stored and a unit to compute R and *low* (*R low Unit*). The *R low Unit* is made of a 16 bits counter for already coded symbols and a 16 bits counter for the syntax elements. An adder and a subtracter are used to calculate R and *low* respectively with the aforementioned ALAP strategy.

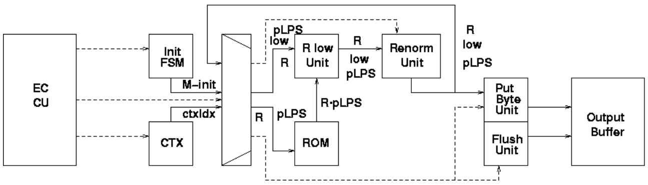

Figure 4. Proposed architecture block scheme

A multiplexer allows to correctly select the input values for the *R low Unit* depending on the current symbols encoding method. The interval renormalization is managed by the *Renorm Unit*. In order to keep the renormalization simple, it has been implemented as a 16 bits subtracter and a shifter. Observing that the smallest value for *R* is `0x0001` and that the renormalization stops when $R \geq$ `0x0100`, the worst case is eight iterations. The output of the encoder is managed by the *Put Byte Unit*. This block has been implemented through some adders, few logic and two 32 bits shift registers (left-shift and right-shift) as depicted in Figure 5.

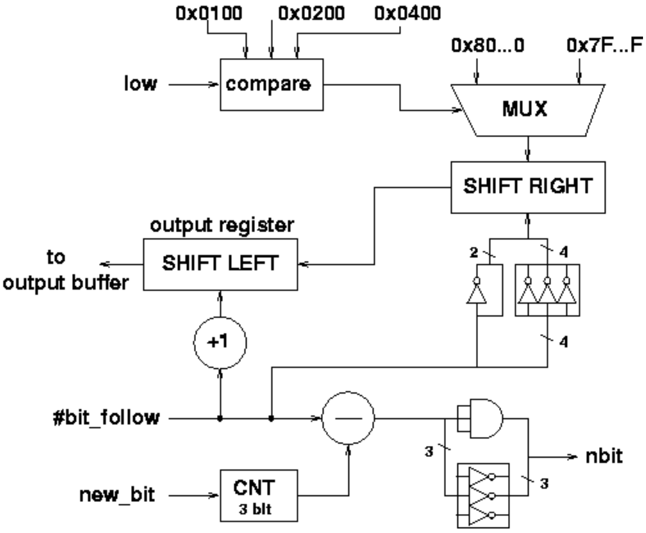

Figure 5. Put byte Unit

Through simulations on the JM software model, it has been found that 32 bits grant to be able to store the coded bits in the worst case. As the worst case we considered the case when one coded bit is generated after the maximum number of "*follow*" bits. The output register, devoted to store the coded bytes needs to be carefully sized in order to accommodate the output bits without dropping or stopping the coding process. Considering that the renormalization can generate up to 8 bits (one for each renormalization step), that the *follow* requires up to 32 bits and that the last generated bit could complete a byte, the output register should be 48 bits wide. Finally the content of this register is stored into the *Output Buffer*. The *flushing* procedure required to terminate the coding of a slice [13] is implemented by the *Flush Unit* (see Figure 4). Its internal structure is the same as for the *Put Byte Unit*. The only difference is that the *follow* is not required and that, if necessary, a certain number of padding bits are added to complete the last byte.

The proposed architecture requires 11 clock cycles to encode a symbol. The VHDL model developed for the proposed architecture has been synthesized on a 0.18 μm CMOS standard-cells technology. Since the amount of ROM and RAM required by the proposed architecture is extremely small, the use of macros generated by ROM and RAM generators would produce an excessive overhead in terms of area. As a consequence, the ROM has been mapped as logic cells and the RAM as an array of flip-flops.

Post synthesis results show that up to 250 MHz clock frequency can be used with an occupation of 176 kgates. Thus the proposed architecture is able to sustain an incoming rate of 22.73 Mbits/s. This rate allows to process in real time 720x480 video at 30 Hz even at low compression ratios (e.g. 5:1). Compared with the solutions described in [4], [5] and [6] the proposed architecture shows some common points and some differences. In particular, since in [4] an FPGA implementation is considered a fair comparison is not possible. On the other hand we can compare the proposed architecture with [5] and [6]. The performance of the architecture described in [5] is given in terms of full adders. So that we evaluated the performance of a full adder on the same 0.18 μm technology employed for our design. The result is that [5] can sustain up to 20 Mbits/s with near the same complexity of the proposed architecture. Considering the architecture proposed in [6] we can state that it achieves a more than 3 times higher throughput with a nearly double complexity with respect to the proposed architecture. Nevertheless, it is worth pointing out that the reduced complexity and the modularity shown by the proposed architecture makes it suitable for a parallel implementation. As an example resorting to two instances of the proposed architecture the total incoming rate can be doubled at the expense of roughly 350 kgates.

3.2. Adaptive ME coprocessor

The results reported in Section 2 for ME refer to a software implementation. The original FS and UMHexagonS software implementations are quite far from real-time coding. However, thanks to the complexity reduction of our technique, real-time is achieved for the 30 Hz QCIF videos; for CIF ones the real-time is allowed at a frame rate between 15 and 30 Hz 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 architecture sketched in Figure 6. The context-aware control system can be easily realized in real–time, also for larger video formats (e.g. CCIR, VGA, 4CIF). A simple microcontroller such as the 8051, public available as reusable VHDL macrocell, with an implementation complexity of roughly 10 kgates in 0.18 μm CMOS standard-cells technology is well suited for this task. The basic search engine can be realized reusing one of the systolic architectures proposed in the literature for FS, e.g. [11]. In fact [11] features an array of 256 SAD processing elements with a circuit complexity of roughly 105 kgates and a throughput of 1 macroblock (MB) matching per clock cycle. A local memory of 13 kBytes can be used as MB search area buffer to reduce access frequency to large background frame memories. The operation flow for both search engine and context-aware controller is described hereafter.

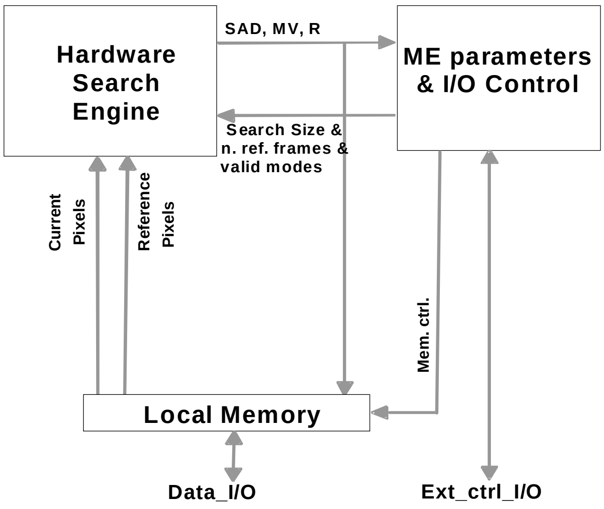

Figure 6. Block diagram of the ME hardware architecture

The search engine starts performing the 16x16 partition ME while the system control waits for prediction cost and optimal reference frame data (step 1). After that, such information can be processed to figure out the allowed partitions and their relative maximum number of reference frames while the ME engine is waiting (step 2). In step 3 the ME engine concludes the estimation while the control system can work on the 16x16 partition for the next MB. According to this flow the systolic search engine is stalled only in step 2 and the estimated percentage stall time is roughly 2%. The required system clock frequency to process in real-time a 720x480 video at 30 Hz is about 70 MHz considering the throughput of 1 MB matching per clock cycle and the 2% processing stall.

4. CONCLUSIONS

In this paper two optimized hardware co-processors, one for CABAC and one for variable block size multi frames ME, have been presented. Both concern the fast implementation of the most demanding H.264/AVC parts; so that they are particularly suited for real-time and high-quality video coding. Post synthesis results on a 0.18 μm standard cells technology show that 720x480 video at 30 Hz and more than 20 Mbits/s can be sustained, proving the proposed coprocessors effectiveness.

ACKNOWLEDGMENT

This work has been supported by EU funds (under NEWCOM NoE) and National funds (PRIMO project).

REFERENCES

[1] 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

[2] 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

[3] 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”, IEEE International Symposium on Circuits and Systems, pp. 23-26, 2005

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

[5] R. Osorio, J. Bruguera, “Arithmetic coding architecture for H.264/AVC CABAC compression system”, IEEE Euromicro - Digital System Design, pp. 62–69, 2004

[6] H. Shojania, S. Sudharsanan, “A high performance CABAC encoder”, in International IEEE-NEWCAS Conference, pp. 19–22, 2005.

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

[8] H. Tourapis, A. Tourapis, “Fast motion estimation within the H.264 codec”, Proc. IEEE ICME'03, July 2003, pp. 517-520

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

[10] S. Saponara et al., “Adaptive algorithm for fast motion estimation in H.264/MPEG-4 AVC”, Proc. Eusipco 2004, Wien, Sept. 2004, pp. 569 – 572

[11] 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, pp. 796-799, Bangkok, 2003

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

[13] 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)

[14] 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, pp. 620–636, July 2003