FPGA-based SoC for transcoding H264/AVC-SVC with low latency and high bitrate entropy coding

Michael Guarisco, Hassan Rabah, Yves Berviller, Serge Weber, S. Belkouch

FPGA-based SoC for transcoding H264/AVC-SVC with low latency and high bitrate entropy coding

Conference Paper · September 2009

DOI: 10.1109/SOCCON.2009.5398004 · Source: DBLP

CITATIONS
2

READS
361

5 authors, including:

Michael Guarisco
Université de Technologie de Belfort-Montbéliard
14 PUBLICATIONS **97** CITATIONS
SEE PROFILE

Yves Berviller
University of Lorraine
53 PUBLICATIONS **224** CITATIONS
SEE PROFILE

Serge Weber
University of Lorraine
135 PUBLICATIONS **725** CITATIONS
SEE PROFILE

FPGA-BASED SoC FOR TRANSCODING H264/AVC-SVC WITH LOW LATENCY AND HIGH BITRATE ENTROPY CODING

M. GUARISCO[†], H. RABAH[†], Y. BERVILLER[†], S. WEBER[†] and S. BELKOUCH[‡]

[†]Université Henri Poincaré Nancy 1, Faculté des Sciences et Techniques - BP 239
54506 Vandoeuvre-lès-Nancy Cedex, FRANCE

[‡]Université Cadi ayyad ENSA
BP575, Avenue Abdelkarim Khatabi, Guéliz, Marrakech, Marocco

ABSTRACT

Scalable Video Coding extension of H.264 standard is very suitable for content adaptation and addressing different terminals. However, in various cases it is necessary to perform transcoding in video coding layer requiring tremendous computation and hardware acceleration. In this paper, we present an efficient hardware architecture of a CAVLC codec based on a new method that provides a constant and reduced latency. The presented method calculates the 16 DCT coefficients in parallel. The results of hardware implementation targeting a Xilinx Virtex 5 FPGA are presented.

I. INTRODUCTION

The H.264 video compression standard is very efficient, since it allows reducing the bitrate of a multimedia stream by half, compared to previous standards such as H.263 [1]. Thanks to predefined profiles and levels, this standard enables a wide range of applications, going from mobile video streaming to high definition TV over wired/wireless networks. The scalable extension of H.264/AVC standard called SVC, is defined to address heterogeneous terminals and different available bandwidth. One of the main features of the H.264 is the separation between the video coding layer (VCL) and the network adaptation layer (NAL). SVC exploits this separation by encoding input video into several layers corresponding to different video size, frame rate or quality. Each layer corresponds to a NAL unit so that it is possible to access to a given spatial/temporal/quality level of the compressed video sequence by selecting appropriate SVC layers. This selection process called SVC layer dropping is applied to perform video adaptation. This adaptation scheme can be efficient in several cases but does not cover all possible scenarios. Therefore, transcoding is still a necessary operation that must be carried out in the video coding layer (VCL). This operation requires a single or multiple entropy decoding and encoding before and after image or video manipulation. Figure 1 depicts a typical use case, where an adaptive home gateway addresses various terminals and capable of performing adaptability in NAL and VCL layers.

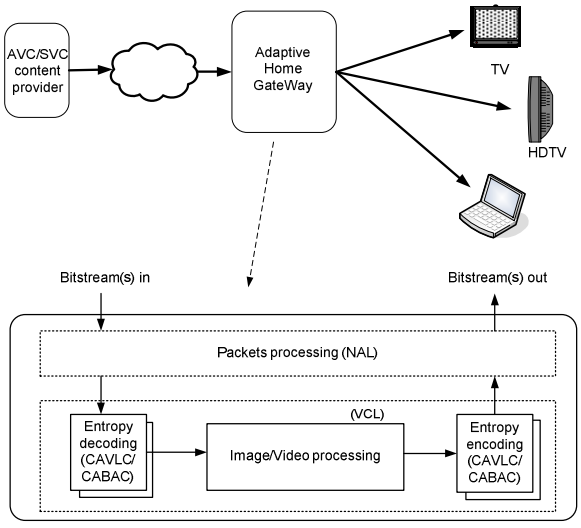

Figure 1: Adaptive Home gateway with High-level architecture of H.264/AVC-SVC Transcoder.

Adaptability in VCL layer requires entropy decoding and encoding that must be performed over one ore more streams depending on the number of required input and output bitstreams. Two statistical coding tools - the Context-based Adaptive Variable Length Coding (CAVLC) and the Context-based Adaptive Binary Arithmetic Coding (CABAC) - have been adopted in different profiles of the H.264/AVC video coding standard, and must be interoperable depending on input profile and output profile. Therefore, the entropy coding and

decoding are important tasks that must be executed efficiently in the transcoding process. In this paper, we propose a hardware real time transcoding architecture of a video stream. The architecture is based on a CAVLC decoder, encoder, and frequency selection in the DCT domain. We particularly focus on the CAVLC architecture.

II. RELATED WORK

The performance of H264 encoding standard is obtained in particular with variable length entropy encoding CAVLC. Such a performance can only be achieved at the price of a high complexity in the compression algorithm. Since this complexity depends on the target architecture, several models have been developed to reduce it [2-5].

Most approaches encode symbols sequentially. In these approaches, the execution time of the VLC encoding for a given block depends on the quantization parameter and the type of video [6] [7]. The corresponding architectures introduce a random execution time that requires an intermediate buffer in upstream of the entropy encoder between the encoder and the quantizer. Indeed, variable execution time implies a variable input rate, whereas the output rate of the quantizer is fixed. It is therefore necessary to provide a buffer and a controller to synchronize the buffer with the CAVLC encoder.

III CAVLC ALGORITHM

The CAVLC algorithm uses a variable length encoding, which plays an important role in image compression. Frequently occurring elements can be assigned very short codes, while infrequent elements can be assigned longer codes. This can significantly reduce information redundancy. CAVLC is called adaptive because, a block i is coded based on the information coming from block i-1; i.e. the neighboring blocks (left-hand and upper previously coded blocks). This adaptation is also used inside the block when encoding the level of each coefficient. The codeword of a coefficient depends on the precedent level or, in the case of the first coefficient, on the number of non-zero coefficients and on the coefficients that are equal to 1 or -1 inside the same block.

The modules that compute and generate the bit stream receive their coefficients from a frequential transformation followed by a quantizer and reorganized through a zigzag scanning. In general, CAVLC encodes each block in five independent steps. In the first step it generates “CoeffToken” that encodes both the total number of non-zero coefficients (TotalCoeffs) and the number of trailing +/-1 values (TrailingOnes) in a block. “CoeffToken” also depends on the number of non-zero coefficients in the left-hand and upper previously coded blocks. The second step allows the encoding of the “TrailingOnes” and represents the sign of each coefficient by a single bit. The TotalZeros parameter encodes the amount of all zeros preceding the highest non-zero coefficient. Afterward comes the actual encoding of the non-zero coefficients contained in the block. It is at this step that we contributed by removing the look-up tables and generating the code from simple values, the coefficient sign and its absolute value. The next step encodes the number of zeros in the block related to the non-zero coefficients. Finally, it is necessary to indicate in the final bitstream where to find these zeros among the non-zeros coefficients. This step is called “Run_Before” because it gives the number of zeros preceding each non-zero coefficient within the zigzag reorganized block. The corresponding syntactic element is calculated in the compression sub-module that we call “Encode RunZeros ». Each of these steps is represented by a syntactic element that, when they are all concatenated, build up the final bitstream.

IV PROPOSED ARCHITECTURE

A. Global view

The majority of the CAVLC hardware implementations encode the coefficient values in an iterative way. The number of iterations, and consequently the time of global calculation, is hence dependent on the number of non-zeros coefficients that belong to the block to be encoded. We noticed during the implementation on a multiprocessor system that this particular task requires numerous operations and that the execution time is significantly modified according to the number of non-zero coefficients. The originality of our architecture is based on the use of massive parallelism that allows on one hand fast calculation and on the other hand a fixed execution time. The architecture is also fully combinatorial and does not need any controller. Thus, except for the loading time of the block, which is loaded sequentially, the encoding is executed in one clock cycle.

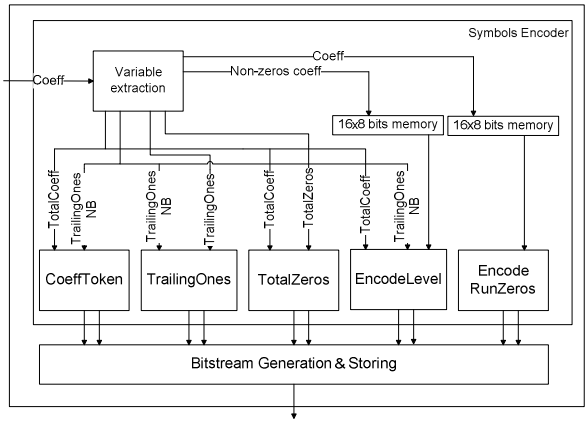

Figure 2: Block diagram of CAVLC architecture

B. Details of operations

In Figure 2 a first phase of pretreatment ("Variable Extraction") can be distinguished, which introduces a latency of a few cycles. At this stage, there are data selections for the repartition to different encoding blocks. Theses data could be coefficients or intermediate results of non-zero coefficients ("Total_Coeff") or the number of zeros ("Total_Zeros") etc. This pretreatment is done on the fly and continuously by updating the output values for each new coefficient input. The expected values are obtained after 16 coefficients are processed and only then, the architecture delivers a valid bitstream for the block. It is then necessary to make a selection at the output of the encoder. Each of the tasks delivering a syntactic element is done in parallel, as they are independent from each other. The last module that generates the bitstream concatenates on the fly the syntactic elements issued from the preceding modules.

The CAVLC bitstream is encoded by a block of coefficients, in addition to encoding parameters that allow a reduction of the bitstream size. Thus, variables that have been defined in the algorithmic part of this paper have to be encoded. Each of the sub-modules uses predetermined tables that accelerate the values encoding. Only the «EncodeLevel» module, that processes coefficient values and «Encode RunZeros» require more complex calculations. The number of non-zeros coefficients can vary from 0 to 16 per block and most of the common architectures use an iterative computation on these numbers. This necessitates a controller and leads to a variable execution time that is unpredictable. For each coefficient, the code-word depends both on its value and on the value of preceding coefficients. We use an index which is going to be incremented or not for the following coefficient encoding depending on whether the coefficient value is higher than a defined threshold. These indexes are calculated beforehand and then distributed to the 16 processing modules. Each module uses this index and the corresponding coefficient value to find out the code-word to be used. The sub-modules «Table generation» contain the algorithm of the code-word generation. The code-words are calculated by means of two parameters: Their size and their value. Using these two parameters, the bitstream generator can reconstitute the global bitstream. We are using the same principle of parallelism in order to calculate syntactic elements of «RunBefore», with the noteworthy difference that we are using predefined tables to find the value and size of each element. These tables are duplicated in each module in order to not overload the access to this table in contrast with the case where a single table is available for all 16 modules. As the number of the value is relatively restricted for these tables, the extra cost of memory is negligible.

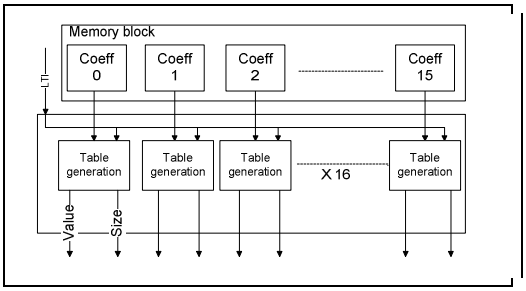

Figure 3: Encode Level detail

We propose a different approach by systematically performing in parallel the calculations of level encoding (Figure 3) and thus obtaining the totality of the codes on one clock transition whatever the number of the coefficients to be processed. We process the iteration in 16 identical modules. These modules are purely combinatorial and allow encoding in a fixed time. The cost of this fixed time is that some of the modules are not always used and continue to function while their results are not going to be taken in account. For example, we could have 7 non-zeros coefficients in the block, which means that 9 modules are not going to be used but they will execute the calculation on the values which most of the time, are coming from the preceding blocks.

V IMPLEMENTATION RESULTS

We have verified our architecture using the simulation tools from Xilinx: first, in terms of functional specification, and then with timing

information. We implemented our CAVLC block on an embedded platform that includes a Virtex 5 FPGA. Table 1 presents the implementation results of the different modules composing the CAVLC encoder. The results come from Xilinx synthesizer. The pretreatment module is a master in the architecture. It works at a maximum frequency of 152 MHz and generates a clock signal for the sub-modules which work at a frequency 16 times lower. Because of the parallelism these sub-modules provides, in one clock cycle, the value of each syntactic element for a block of 16 coefficients.

Table 1: FPG implantation results of CAVLC

	Slice Registers	Slice LUT's	Bufg / BufGCTRL's
CAVLC: preprocessing	32	28	1
CAVLC : memory480	5	75	45
CAVLC: CoeffToken	/	1	/
CAVLC: TotalZeros	/	52	/
CAVLC: MemoryEL	/	16	1
CAVLC : MemoryERZ	/	16	1
CAVLC : EncodeLevel	/	1906	/
CAVLC: EncodeRunZero	/	417	/

Compared to existing architectures [2], we obtain with our proposal a significant reduction in the operating frequency while maintaining a comfortable speed for HD video. One of the advantages of our architecture is that it can save either the internal RAM or registers in the interface between the quantizer and the entropy coder. Moreover, thanks to the parallelization of most computations, only the first part (variable extraction) of the architecture requires a frequency of 63 MHz. This phase corresponds to the first stage of the pipeline. The second stage may work at a frequency 16 times lower (4 Mhz), to provide a real-time encoding of high definition 1920x1080p images at 30fps. Table 2 gives a comparison of our architecture with two similar ones. Our architecture is divided in half to detail the frequencies used with one or the other stages of the pipeline, in order to highlight the fact that most of the elements of the architecture work at a very low frequency. Compared to [6], our proposition has an overhead in terms of logic gate, but can process a larger amount of data with (for the most part of the architecture) a much smaller frequency, so clock consumption is widely reduced.

VI Conclusion

Our architecture allows executing the CAVLC in a fixed time. In addition, this time is considerably reduced at the cost of a slight increase of the area occupied. The principle of this architecture is completely reusable. The modules can be used in any CAVLC decoder as well as any encoder with comparable complexity and execution time [3]. This architecture is designed to integrate into an encoder or transcoder for real time high definition video.

Table 2: Comparison of CAVLC architectures

	Chen *et al.* [7]	Rahman *et al.* [6]	Proposed(stage1/stage2)
Gate count	23600	6855	28152 (351/27801)
Clock frequency	100MHz	50MHz	63MHz/4MHz
Encoding	Full HD 30fps	CIF/QCIF 30fps	Full HD 30fps

REFERENCES

1. N. August and D.S. Ha, "Low Power Design of DCT and IDCT for Low Bit Rate Video Codec" *IEEE Transactions on Multimedia*, Vol. 6, No. 3, pp.414-422, June 2004.
2. N Chih-Da Chien; Keng-Po Lu; Yi-Hung Shih; Jiun-In Guo; "A high performance CAVLC encoder design for MPEG-4 AVC/H.264 video coding applications" Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on, Volume , Issue , 21-24 May 2006
3. Myungseok Oh, Wonjae Lee, Yunho Jung, and Jaeseok Kim, Design of High-Speed CAVLC Decoder Architecture for H.264/AVC, ETRI Journal, Volume 30, Number 1, February 20
4. George, T.G., Malmurugan, N., "The Architecture of Fast H.264/CAVLC Decoder and its FPGA Implementation", IIHMSP 2007, Volume 2, issue, 26-28 Nov. 2007 Pages:389
5. Yi-Chih Chao, Shih-Tse Wei, Jar-Ferr Yang, and Bin-Da Lui. "Combined CAVLC Decoder and Inverse Quantizer for Efficient H.264/AVC Decoding," Circuits and Systems, 2006. APCCAS 2006. IEEE Asia Pacific Conference on 4-7 Dec. 2006 Pages 259
6. Choudhury A. Rahman, Wael Badawy. "CAVLC Encoder Design for Real-Time Mobile Video Applications". IEEE Transactions on circuits and Systems : Express Briefs Vol. 54 N°10, Oct. 2007.
7. Tung-Chien Chen, Yu-Wen Huang, Chuan-Yung Tsai, Bing-Yu Hsieh, Liang-Gee Chen. "Architecture Design of Context-Based Adaptative Variable-Length Coding for H.264/AVC". IEEE transactions on Circuits and Systems : Express Briefs, Vol. 53, N°9, Sept. 2006.