AVC Entropy Coding for MPEG Reconfigurable Video Coding

Hussein Aman-Allah and Ihab Amer
Laboratory of Microelectronic Systems (GR-LSM), EPFL
CH-1015 Lausanne, Switzerland
{hussein.aman-allah, ihab.amer}@epfl.ch

Abstract

The emergence of the Reconfigurable Video Coding (RVC) Standard has led to the development of its accompanying decoding Video Tool Library (VTL). This library is comprised of a set of video coding tool also referred to as Functional Units (FUs) that are combined to form the different video decoders. This paper introduces a set of FUs that represent potential contributors to the RVC Encoding Toolbox corresponding to the AVC baseline profile entropy coding modules. The modules are implemented using RVC-CAL, and has been synthesized into hardware targeting the Virtex 5 FPGA. A novel memory model for the Look Up Tables (LUTs) is also introduced and introduces up to 31.5% of savings in the space required to store them. The results being compared to other implementations from the literature show substantial improvement to traditional implementation approaches using C/C++ or VHDL conclude the paper.

1. Introduction

The RVC VTL is a normative library of video coding tools, also called FUs covering at the moment MPEG-2 simple profile and main profile, MPEG-4 SP, MPEG-4 AVC baseline profile, SVC baseline profile (the so called “MPEG Toolbox”). This library is specified with a textual normative specification and a corresponding reference SW. Such reference SW specification is provided using RVC-CAL as specification language for each library component.

The objective of developing an encoder toolbox is to demonstrate that the existence of RVC encoding tools supports the evolvement of the RVC standard [2]. Many benefits can be achieved by supporting the RVC framework with such encoding tools. Instead of modifying the available C/C++ software reference model of a specific MPEG standard to make it able to generate the Bitstream Syntax Description (BSD) and the Functional Unit Network Description (FND) in order to test the conformance of a corresponding RVC decoder, building an RVC encoder using RVC-CAL would be more convenient. Building the encoder using RVC-CAL enables the exploitation of the commonalities between many components within various MPEG standards. Hence, the existence of an informative VTL would be advisable. In this case, a typical MPEG RVC Encoding/Decoding scenario would be as shown in Figure 1 [3].

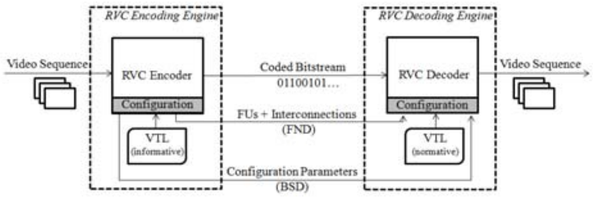

Figure 1 - RVC Encoding/Decoding Scenario

This would allow for the construction of “reconfigurable encoders” using the encoder's informative VTL, which specifies the set of FUs that may be interchangeably combined and connected to form different video encoders, with various compression performances and implementation complexities. Some of the FUs of the normative VTL of the RVC decoder can be used to construct the encoder, either directly (such as the IDCT module), or indirectly, by inferring the IO structure of a module in the encoder from the corresponding module in the decoder.

In this paper, efficient RVC-CAL modules for Exp-Golomb and CAVLC based Entropy Coding are presented. Both modules are part of a complete AVC baseline profile compliant RVC Encoder. The encoder is one of the very first attempts to raise the need for an RVC Abstract Encoder Model (AEM) and MPEG RVC Encoding Toolbox. The modules were implemented targeting the Virtex 5 XC5VLX50T FPGA. The hardware synthesis, software productivity, and memory optimization results are presented in section 3. A discussion that explains the reasons behind such results and potential future work concludes the paper.

2. Entropy Coding

Entropy encoding in AVC/H.264 relies on several architectures. The baseline profile employs Exp-Golomb Coder and Context-Adaptive Variable Length (CAVLC) Coder which both employ Variable Length Coding (VLC). Within the AVC/H.264 standard, the syntax elements above the slice layer are encoded using fixed- or variable-length binary codes. Starting from the slice layer and all the lower layers, elements are encoded using either Exp-Golomb VLCs or CAVLC depending on the type of the parameter [4].

2.1 Exp-Golomb

Exp-Golomb encodes all syntax elements except for the quantized transform coefficients, which are encoded using the CAVLC scheme. Thus, the same VLC tables are used for almost all the syntax elements; which contribute to reducing the memory requirements needed to store such tables.

Exp-Golomb codes are variable length binary codes that are constructed systematically with the following pattern:

$$Code = [Mzeros][1][INFO]$$

The code words constructed in this fashion are guaranteed to have symmetric width; where the INFO field is represented in M bits, making the width of the code word equal to $2M + 1$ [4].

Given a parameter k, the corresponding *code_num* is then calculated according to one of four modes and the mode selection decision is based on the parameter type. Each of such modes is designed to produce shorter codewords for frequently-occurring values and longer codewords for less common parameter values. After the *code_num* has been calculated, codewords can be constructed on the basis of the following equations.

$$M = \lfloor \log_2 (Code\_Num + 1) \rfloor$$

$$INFO = Code\_Num + 1 - 2^M$$

Figure [3] shows the Exp-Golomb module implemented in RVC-CAL as a simple network with one input port, which receives the parameter token to be encoded and one output port which outputs the codeword serially. The Exp-Golomb network provides interconnections among nine different actors, out of which four (plus one, on which more later) actors perform the tasks of the four different mapping modes. The rest of the actors include a *controller* responsible for the mapping mode decision based on the parameter type, a *code generator* to construct the Exp-Golomb codeword as described above, an *assembler* responsible for the reordering, concatenation and outputting of the codeword bits, and finally a *utility* actor that provides decimal to binary conversion to represent the INFO bits.

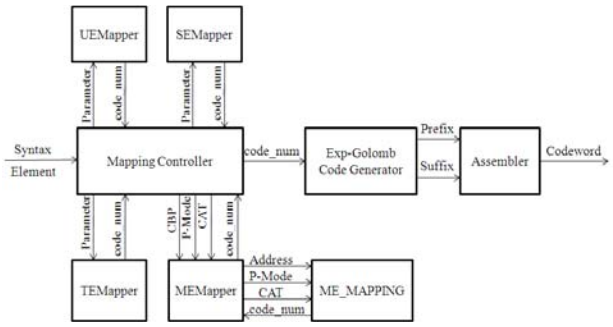

Figure 2 - Exponential Golomb RVC-CAL Network

For each of the four different mapping modes an actor specifies how the mapping from the parameter to the *code_num* is to be performed. The *unsigned mapping* actor is a simple one, which outputs the *code_num* token with a value equal to that of the input token. The *signed mapping* actor defines two different actions depending on the value of the input token; one in which the *code_num = 2*|*k*| for non-negative values and the other in which the *code_num = 2*|*k*|*-1* for negative values. The *truncated mapping* actor also defines two different actions following the unsigned mapping scheme for values greater than one and inverting the binary value of the input token otherwise. The mapped exponent relies on a lookup table (LUT) specified in the ITU-recommendation [5] based on different prediction modes and chroma array types. Thus, the *mapped exponent* actor is a bit more involved than the other three. It acts as a controller communicating with a dedicated actor storing the LUT (targeting a ROM implementation upon synthesis). The communication channel is interfaced with the corresponding address (input) and data (output) ports. The LUT is organized in a reordered manner, so that the *coded_block_pattern* is used as an address for the table to read back the corresponding *code_num.*

Implementing the Exp-Golomb module in RVC-CAL provides a high degree of abstraction that allows for seamless integration within other modules with minimal effort. As opposed to other implementations provided in C for example, no pre-knowledge of the user defined data types or any other implementation-specific details are needed. Integrating the Exp-Golomb module within the larger entropy coding module is as easy as defining the interconnections for its input and output ports; that adds only two lines of code. Thus, the Exp-Golomb module and the CAVLC module can be implemented by two completely different developers and then the integration overhead almost sums up to zero.

2.2 CAVLC

The H.264/AVC recommendation [5] provides two alternative entropy coding methods (both of which are context-adaptive variable-length based), Context Adaptive Variable Length Coding (CAVLC) and Context Adaptive Binary Arithmetic Coding (CABAC). Context based adaptivity improves the performance considerably relative to prior standards. Since CABAC is not a part of the baseline profile only CAVLC is considered hereby.

CAVLC exploits the statistical properties of the quantized 4x4 block with the coefficients to be encoded to provide a compact and efficient lossless representation of the data. It is also context adaptive in the sense that different VLC tables are used for the different syntax elements and are switched according to the values of previously coded elements [4]. Entropy coding performance is improved in comparison to other single-table based schemes because the different VLC tables are designed to accommodate the specific statistics of each syntax element.

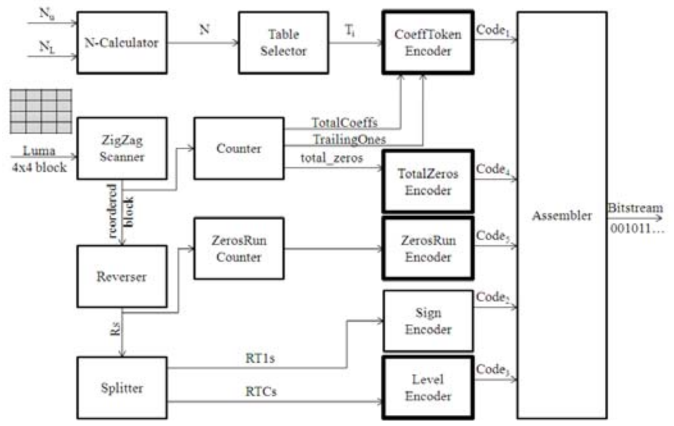

Figure 3 - CAVLC RVC-CAL Network

Figure 3 [3] describes the proposed RVC-CAL implementation of the CAVLC algorithm (thoroughly described in [4]). It starts with the *Zigzag Scanner* actor performing the pre-processing on the block of coefficients and making the reordered block available for both the *Counter* and the *Reverser*. The *Counter* prepares the meta-data needed throughout the algorithm execution; namely the *TotalCoeffs*, *TrailingOnes* and the *total_zeors*. The *N Calculator* calculates the number of non-zero coefficients based on the corresponding values of the neighboring blocks. The *CoeffTokenEncoder* uses the *TotalCoeffs* and *TrailingOnes* to access one of the LUT to retrieve the corresponding codeword. The choice of the LUT to be accessed is made by the conjunction of the *N-Calculator* and the *Table Selector* actors. The algorithm execution proceeds in a distributed fashion among the other actors and follows naturally as illustrated in Figure 3.

The algorithm doesn't execute according to its intuitive order but rather depending on the tokens available at each point in time during execution and that adds yet another advantage to the RVC-CAL implementation. The different actors in Figure execute in independently and it is then the responsibility of the *assembler* to compile the output tokens from the different actors, reorder them and output the encoded stream serially.

H.264/AVC CAVLC encoding relies heavily on the usage of LUTs (Figure 3 shows the sub-modules in which lookup is involved having a thicker border), something which provides significant improvement of efficiency but with the price of complication of the fabrication process and additional consumption of area. The proposed CAVLC module introduces a memory model which preserves the complete LUTs nevertheless still sparing up to 31.5% of the area required to store them.

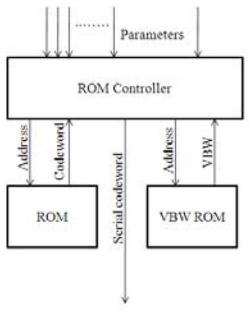

Figure 4 – CAVLC LUT Memory Model

Figure 4 shows the proposed memory model which is represented in RVC-CAL as an actor representing the memory controller, another representing the memory itself and a third with the valid bit widths (VBW) of the corresponding entries of the second actor. This approach exploits an efficient storage technique for the run of zeros to the left of the codeword. For example the codeword 00000000001 is stored as only 001 in the memory with an 11 in the corresponding location in the VBW memory. In this example, only 7 bits are to be stored instead of 11. With almost 475 different variable length code words to be stored [5], such reduction multiplies and offers approximately 21% reduction in the ROM usage. It is then the responsibility of the controller to align the codeword before outputting it, a task which requires minimal computational interference.

3. Results & Analysis

The Exp-Golomb and CAVLC modules have been integrated with other major components comprising the AVC-Compliant RVC Encoder. The entropy coding module has been synthesized into hardware targeting the Virtex 5 XC5VLX50T FPGA.

3.1. Software Productivity

One of the major RVC advantages is that it accompanies its normative description language (RVC-CAL) with many supporting tools that enable automatic code generation into software (CAL2C) and hardware (CAL2HDL) [6]. Table 1 presents a comparison between the proposed RVC-CAL implementation, the AVC/H.264 JM reference software written in C [7], and a reference VHDL implementation.

Table 1 - Comparison Between RVC-CAL, C, and VHDL Implementations

	RVC-CAL	C/C++	VHDL
Lines of Code (LOC)	922	1762	3784
Development Time (MH)	72	N/A	116
Number of Developers	1	3	1

The table shows the lines of code, development time and number of developers required for the RVC-CAL, C and VHDL implementations correspondingly. The numbers show that the RVC-CAL implementation needs less time to be developed and hence requires fewer developers. This gives the RVC-CAL implementation an advantage of reducing the development costs, while at the same time minimizing the Time To Market (TTM).

3.2. Hardware Synthesis

The results are echoed on the hardware implementation level. The HDL model is generated from the presented CAL model using the CAL2HDL tool. The HDL code is synthesized using Xilinx ISE targeting the Xilinx Virtex 5 XC5VLX50T FPGA. The synthesis results of the CAVLC module are provided as an example for the quality of the hardware implementation. Table 2 [8] summarizes the performance of the CAVLC module and compares it against an array of studies and implementations from the literature.

Table 2 - Performance of the CAVLC module compared to other implementations

	Critical Path (ns)	CLK Frequency (MHz)	Number of LUT	Throughput (MSamples/s)
Proposed Implementation	3.729	268.1	112	268
[9]	9.6	103.8	2,467	103.8
[10]	31.326	31.9	84,902	510.4
[11]	3.1	210	N/A*	100
[12]	8	125		74.04
[13]	13.15	76	3,946	6.75

* No precise data available at the time of comparison.

The synthesis results show that the proposed implementation exceeds the throughput of the others with a factor of 2.58 in the worst case, with the exception of [10] because that implementation employs hardware redundancy to exploit parallelism and consumes 758 times more hardware resources. Besides the advancements in FPGA manufacturing technology, the results can be attributed to several factors. The abstract and encapsulated implementation facilitated by RVC-CAL allows for optimization of every submodule (actor) on its own, which collaborates to deliver overall optimization of the whole CAVLC module. In addition, the optimizations performed by the CAL2HDL tool during the HDL code generation (explicitly specified in [14]) also contribute greatly to the quality of the synthesis results.

3.3. Memory Optimization

The implemented ROM Model that was applied to the CAVLC LUTs and discussed in section 2.2 resulted in savings in memory space reaching around 30%. Table 3 examines the savings per each LUT. The table summarizes the memory space consumed by the implementation for each LUT and the corresponding LUT dedicated for storing the VBW versus the space consumed by the LUTs stored using the traditional method.

Table 3 - Savings Resulting from the Proposed Memory Model

	Memory Size	VBW Size	LUT Size	Savings
coeff_token	816	1224	2976	31.5%
total_zeros (Part 1)	336	560	1008	11.2%
total_zeros (Part 2)	144	288	432	0%
run_before	315	525	1155	27.3%

4. Conclusion

In this paper, MPEG-RVC has been introduced with elaboration on the road blocks that relate to the Encoding Toolbox. The RVC-CAL implementation of AVC baseline profile compliant Entropy Coding has been discussed to illustrate the benefit of using RVC-CAL dataflow actor language as a specification language for RVC. The results illustrate the advantages of using RVC-CAL as a specification language for the RVC standard. The results for the different aspects have been presented and compared to other traditional development methodologies and to other implementations from the literature.

6. References

[1] S. Bhattacharyya, G. Brebner, J. Eker, J. W. Janneck, M. Mattavelli, C. Platen, and M. Raulet: "OpenDF - A Dataflow Toolset for Reconfigurable Hardware and Multicore Systems", Computer Architecture News, Special Issue: MCC08 - Multicore Computing 2008, Volume 36, Number 5, December 2008.

[2] I. Amer Development of RVC Encoding Tools, 2009, MPEG Core Experiment M16409.

[3] H. Aman-Allah, E. Hanna, K. Maarouf, and I. Amer. Towards a comprehensive RVC VTL: A CAL description of an efficient AVC baseline encoder. Submitted to, IEEE International Conference on Image Processing (ICIP) 2009.

[4] I. E. G. Richardson. H.264 and MPEG-4 Video Compression. Wiley, The Robert Gordon University, Aberdeen, UK, 2003.

[5] International Telecommunication Union. ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Recommendation H.264, Advanced Video Coding for Generic Audiovisual Services, March 2005.

[6] C. Lucarz, M. Mattavelli, M. Wipliez, G. Roquier, M. Raulet, and J. Janneck et al. Dataflow/actor-oriented language for the design of complex signal processing systems. In Conference on Design and Architectures for Signal and Image Processing (DASIP), 2008.

[7] Joint Video Team (JVT) reference softwarem, version 14.2 http://iphome.hhi.de/suehring/tml/download/old jm/jm14.2.zip, 2008.

[8] H. Aman-Allah, K. Maarouf, E. Hanna, I. Amer, and M. Mattavelli. CAL Dataflow Components for an MPEG RVC AVC Baseline Encoder. Springer Journal of Signal Processing Systems, Reconfigurable Video Coding Special Issue., 2009.

[9] T. Silva, J. Vortmann, L. Agostini, S. Bampi, and A. Susin. FPGA based design of CAVLC and exp-golomb coders for H.264/AVC baseline entropy coding. In 3rd Southern Conference on Programmable Logic (SPL07), 2007.

[10] I. Amer, W. Badawy, and G. Jullien. Towards MPEG-4 part 10 system on chip: A VLSI prototype for context based adaptive variable length coding (CAVLC). In IEEE Workshop on Signal Processing Systems, pages 275-279, 2004.

[11] Y. Yi and B. Cheol Song. A novel CAVLC architecture for H.264 video encoding at high bit-rate. In IEEE International Symposium on Circuits and Systems (ISCAS), 2008.

[12] C. Chien, K. Lu, Y. Shih, and J. Guo. A high performance CAVLC encoder design for MPEG-4 AVC/H.264 video coding applications. In IEEE International Symposium on Circuits and Systems (ISCAS), pages 3838-3841, 2006.

[13] E. Sahin and I. Hamzaoglu. A high performance and low power hardware architecture for H.264 CAVLC algorithm. In 13th European Signal Processing Conference.(EUSIPCO), 2005.

[14] J. Janneck, I. Miller, D. Parlour, G. Roquier, M. Wipliez, and M. Raulet. Synthesizing hardware from dataflow programs: An MPEG-4 simple profile decoder case study. In IEEE Workshop on Signal Processing Systems (SiPS), 2008.