TOWARDS A COMPREHENSIVE RVC VTL: A CAL DESCRIPTION OF AN EFFICIENT AVC BASELINE ENCODER

Hussein Aman-Allah, Ehab Hanna, Karim Maarouf, Ihab Amer

Laboratory of Microelectronic Systems, EPFL
CH-1015, Lausanne, Switzerland

ABSTRACT

The Video Tool Library (VTL) is one of the major normative components of the Reconfigurable Video Coding (RVC) standard. It specifies the set of functional units (FUs) that may be interchangeably combined and connected to form different video codecs, with various compression performances and implementation complexities. In this paper, an efficient AVC baseline encoder that is described in CAL is introduced. The encoder is composed of many modules that also exist in other codecs of the same or different standards. This makes them highly reusable within the RVC framework. The main modules of the designed encoder include: Inter Prediction, Intra Prediction, and Entropy Coding. Brief descriptions of the designed modules, accompanied with CAL design issues are provided.

Index Terms— Reconfigurable Video Coding, MPEG Video Tool Library, CAL Actor Language, AVC.

1. INTRODUCTION

The MPEG RVC aims "to offer a more flexible use and faster path to innovation of MPEG standards in a way that is competitive in the current dynamic environment" [1]. The RVC aims at simplifying the specification of new codecs by reusing components of earlier standards instead of redefining new ones. The existence of reconfigurable tools at the encoder side supports the evolvement of the RVC standard [2]. Figure 1 shows the RVC framework encoding/decoding scenario.

The RVC framework consists of a normative video coding tool library (VTL), a normative language to describe the interconnections among the FUs, called the Functional unit Network Description Language (FND) and a normative language to specify the RVC decoder configuration, called the Bitstream Decoding Language (BSDL) [1]. The VTL bundles a set of Functional Units (FUs) that are common among different profiles/standards. The VTL is specified with a textual specification and a corresponding reference software, implemented in RVC-CAL, a language standardized by MPEG to describe the behavior of dataflow components.

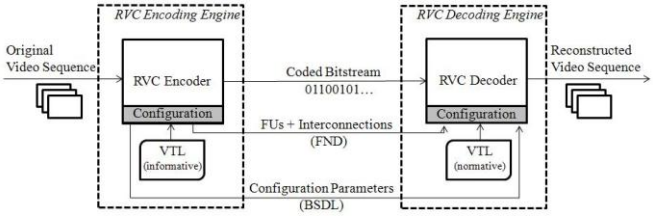

Figure 1 - RVC Encoding/Decoding Scenario

CAL is a textual language that defines the behavior of dataflow components referred to as *actors*. An actor preserves its own state and the interaction between actors is done through *token* exchange. Tokens flow through *I/O ports* that are connected using FIFO channels. Actors are described in a set of actions that execute in sequences of *transitions* during which an actor may *consume* an input token, *modify* its internal state or *produce* an output token. Actions fire depending on the availability of input tokens, their values, the internal state of the actor and the priority corresponding to an action. CAL supports language constructs that describe the action firing process, including *guards, finite state machines* and *priorities.* CAL supports natural constructs that are identified by the RVC framework as essential elements for building MPEG codecs [3]. CAL exploits abstraction and encapsulation features of dataflow programming to define the FUs in a way that they can be interchangeably combined and connected to form different video codecs. CAL also facilitates concurrent development because it improves the maintainability and understandability of the code over other sequential models implemented in C/C++ for example [4].

This paper presents an AVC baseline encoder, described in the CAL actor language (targeting hardware implementation) as an example to demonstrate the concept of component reusability in the RVC standard. The rest of the paper is organized as follows: Sections 2, 3 and 4 present the CAL description of inter/intra frame prediction and entropy coding modules respectively. Analysis of the proposed approach and comparison of results against other

development approaches is provided in Section 5. Finally, Section 6 concludes the paper.

2. INTER FRAME PREDICTION

2.1 Motion Estimation

AVC motion estimation (ME) is based on block matching and transform coding. ME is used to estimate motion between successive frames resulting in high compression of a video sequence [4].

Figure 2 shows the CAL network for the ME module. Full search and Sum of Absolute Differences (SAD) are employed in the proposed implementation.

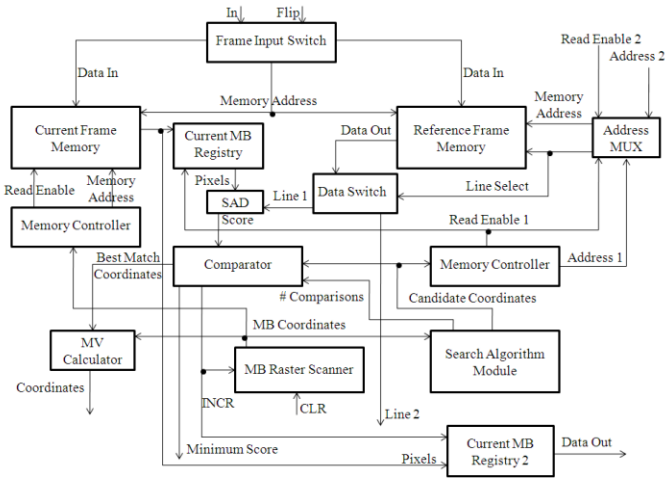

Figure 2 - Motion Estimation CAL Network

The module reads in the pixels of the reference and current frames and stores them in their corresponding memories. The current frame is then scanned by the *Macro-block Raster Scanner* and a best match is selected from a search window calculated by the *Search Algorithm Module*. The selection is done by the *Comparator* and the *SAD*. A motion vector is then calculated and passed, along with the pixels of the current macro-block and the best match, to the motion compensation module.

2.2 Motion Compensation

Figure 3 shows the motion compensation (MC) module of the AVC inter prediction. The module takes as input the motion vector of the best matching macroblock from the ME module. The location of the best matching block is calculated by adding the motion vector to the macro-block location from the *Macro-block Raster Scanner*. The best matching block is retrieved from the ME module by *Memory Controller 1* in order to calculate the compensated frame. The compensated frame and compensation error are calculated and used to reconstruct the current frame which is sent back to the ME module to be used as the new reference.

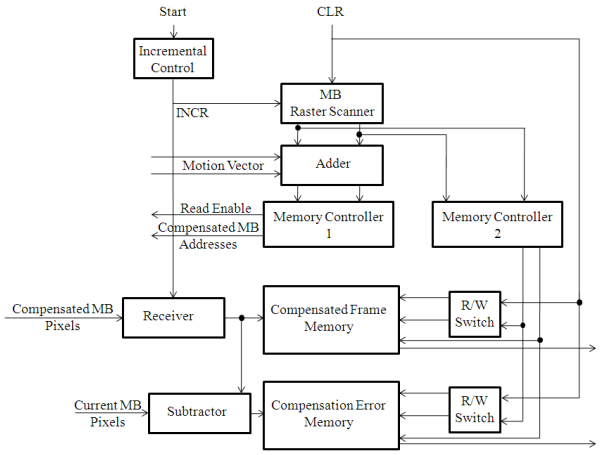

Figure 3 - Motion Compensation CAL Network

3. INTRA FRAME PREDICTION

In intra prediction, predictors are formed from previously encoded and reconstructed MBs. Predictors are formed either for every 4x4 block in a MB (9 modes) or for the MB as a whole (4 modes). The encoder selects the mode which minimizes the difference between the predictor and the block to be encoded.

Figure 4 shows the CAL network for an intra prediction module.

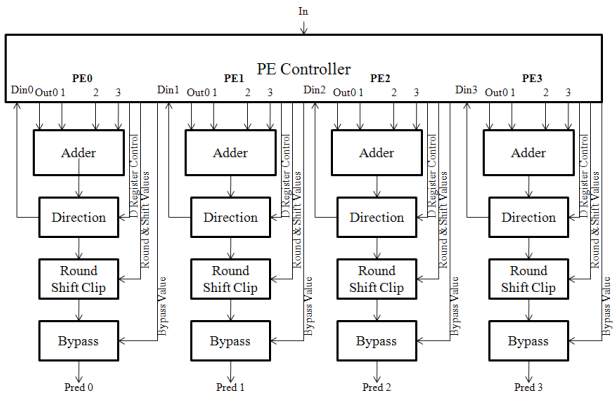

Figure 4- Intra Prediction CAL Network

The design is based on [5], where reconfigurable processing elements (PEs) are used to generate predictors for all the modes instead of one PE for each mode. A PE operates depending on the prediction mode. A four-parallel architecture is used, meaning that four predictors are generated per cycle. This achieves a good tradeoff between area and operating frequency.

The network consists of a controller and four series of adders, registers and shifters, each representing one of the PEs. The main actor in the network is the *PE Controller*. It

first receives tokens serially at its input port for the previously coded and reconstructed samples, which are used for prediction. For modes requiring no calculations (vertical and horizontal extrapolation), it produces a predictor token directly to the *Bypass* actor. In the DC modes, the predictor value is the mean of the surrounding previously coded samples. The *PE Controller* produces tokens to the *Adders* to calculate the sum of the samples. The sum is calculated over more than one cycle and is accumulated using the *Direction* actors, which produce tokens with the values of the intermediate addition results to the *PE Controller*. The division to produce the mean value is done at the *Round Shift Clip*. In other modes, the predictor is a weighted average of some of the samples. Depending on the position of the sample being predicted, its value is calculated as follows:

$$Pred = Clip(((W + X + Y + Z) + Round) >> Shift)$$

The *PE Controller* produces tokens for the *Adders* and those for the *Round Shift Clip* values. At the *Round Shift Clip*, the sum from the *Adders* is added to a round value then shifted to get the average and finally clipped to be between 0 and 255. In plane mode, a linear function is fitted to the previously coded samples. A predictor is calculated as follows:

$$Pred[y, x] = Clip((a + b(x - 7) + c(y - 7)) >> 5)$$

Initial values, in the first row, are calculated using the above formula. The predictors in the other rows are calculated by adding c to each row to produce the next. This is done using the *Direction* actors to accumulate the addition results of each row in order to calculate the predictors for the next.

4. ENTROPY CODING

The AVC baseline profile uses Exp-Golomb to encode all syntax elements except for the residual data that are encoded using Context Adaptive Variable Length Coding (CAVLC).

4.1. Exp-Golomb

Exp-Golomb codes are constructed systematically as variable length binary codes. Figure 5 shows the CAL network implementing the Exp-Golomb.

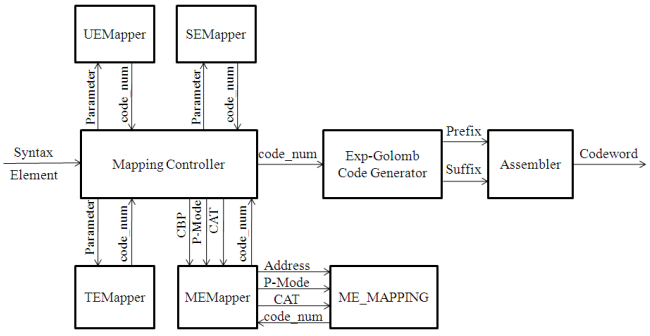

Figure 5 - Exp-Golomb CAL Network

The Exp-Golomb first maps a parameter *k* to a *code_num*. There are four different modes to choose from, based on the parameter type. Each mode produces shorter codewords for values with higher probability of occurrence and longer codewords for less-frequent parameter values. In the implemented CAL network, the four actors performing the different mapping modes are moderated by a *Mapping Controller* actor. The mapped exponent is more involved; it communicates with a dedicated actor storing the lookup tables with the codewords, targeting a ROM implementation upon synthesis. The *code_num* is then used to construct the codeword using the equations provided in [6].

Implementing Exp-Golomb in CAL exploits the abstraction and encapsulation of dataflow programming, allowing for seamless integration of the module within the complete Entropy Coder.

4.2. CAVLC

CAVLC switches different VLC look up tables (LUTs) to encode the different syntax elements based on values of previously coded elements. Each LUT exploits the statistical properties of the syntax elements it is designed to encode. Figure 6 describes the proposed CAL implementation of CAVLC.

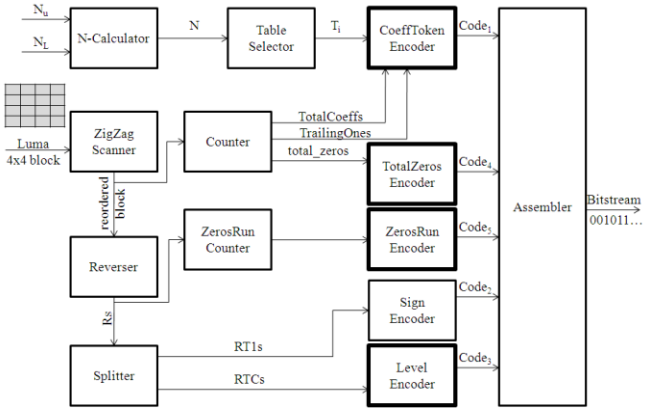

Figure 6 - CAVLC CAL Network

The CAVLC execution is distributed among the different actors shown in figure 6. The actions within an actor fire whenever the tokens they are anticipating are ready in the FIFO channels. Thus, unlike sequential models, the procedure exploits parallelism implicitly. The *Assembler* actor is then responsible for compiling the output tokens from the different actors and producing the encoded stream serially.

5. RESULTS AND ANALYSIS

The RVC framework accompanies its normative description language (CAL) with supporting tools for automatic code generation into both, software (CAL2C) and hardware (CAL2HDL) [7]. Table 1 compares between the proposed CAL implementation, the AVC reference software written in C/C++ [8] and a reference VHDL implementation. Table 2 shows the gain factor for both the productivity (in terms of the average saving in lines of code) and the speedup in development time achieved by using the proposed CAL approach.

Table 1 - Comparison between the different approaches

		CAL	C/C++	VHDL
Lines of Code (LOC)	Inter Prediction[1]	203	356	897
	Intra Prediction	1239	2758	N/A*
	Entropy Coding	922	1762	3784
Development Time (MH)	Inter Prediction	56	N/A*	133
	Intra Prediction	80		N/A*
	Entropy Coding	72		116
Number of Developers	Inter Prediction	1	3	1
	Intra Prediction	1	5	N/A*
	Entropy Coding	1	3	1

[1] The comparison relates to integer pel inter prediction implementing full search and SAD.
* No precise data available at the time of comparison.

Table 2 - The gain from using the CAL implementation

	Average Productivity Gain	Average Speed-up in Development Time
CAL over C/C++	1.96	N/A*
CAL over VHDL	4.20	1.99

* No precise data available at the time of comparison.

6. CONCLUSION

In this paper, A CAL implementation of an AVC baseline profile encoder has been presented. The components presented are potential candidates for addition to the VTL, due to the abstract and encapsulated representation that was made feasible by CAL (refer to [9] for more details). The results presented in section 5 are also echoed in [10]. Such results demonstrate the advantage of using CAL as the RVC standard specification language due to its ability to exploit the reconfigurability, reusability, understandability and maintainability of a dataflow programming language. Furthermore, a CAL implementation needs less development time than its sequential or HDL counterparts and can target both software and hardware architectures.

7. REFERENCES

[1] Jang, E. S., Ohm, J., Mattavelli, M. (January 2008). Whitepaper on Reconfigurable Video Coding (RVC). ISO/IEC JTC1/SC29/WG11 document N9586.

[2] Lucarz, C., Amer, I., Mattavelli, M., (November 2009). Reconfigurable Video Coding: Objectives and Technologies. Proceedings of IEEE International Conference on Image Processing, Special Session on Reconfigurable Video Coding.

[3] Mattavelli, M. (2008). Reconfigurable Video Coding (RVC) a new Specification and Implementation Paradigm for MPEG Codecs. The 12th Annual IEEE International Symposium on Consumer Electronics. [Keynote Presentation].

[4] Yang, W. (2003). An Efficient Motion Estimation Method for MPEG-4 Video Encoder. IEEE Transactions on Consumer Electronics, 49(2).

[5] Huang, Y., Hsieh, B., Tung-Chien C., Chen, L. (2005). Analysis, Fast Algorithm, and VLSI Architecture Design for H.264/AVC Intra Frame Coder. IEEE Transactions on Circuits and systems for Video Technology, 15(3), 378-401.

[6] Silva, T., Vortmann, J., Agostini, L., Bampi, S., Susin, A., (2007). FPGA Based Design of CAVLC and Exp-Golomb Coders for H.264/AVC Baseline Entropy Coding. 3rd Southern Conference on Programmable Logic (SPL07).

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

[8] Joint Video Team (JVT) reference software, version 14.2. [Online] http://iphome.hhi.de/suehring/tml/download/old_jm/jm14.2.zip

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

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