Hardware Design for a Reference Frame Compression Technique for Multiview Video Coding

Rafael Justo[#1], Felipe Sampaio[#2], Sergio Bampi[#3]

[#]*PPGC/PGMICRO, Instituto de Informática, Universidade Federal do Rio Grande do Sul*
Porto Alegre, RS - Brasil
[1]rafael.justo@inf.ufrgs.br
[2]felipe.sampaio@inf.ufrgs.br
[3]bampi@inf.ufrgs.br

***Abstract* — This work presents a hardware architecture design for a reference frame compression technique for Motion and Disparity Estimation (ME/DE) on Multiview Video Coding (MVC). The goal of the compression technique is to reduce the off-chip memory bandwidth during the ME/DE processing, reducing the power consumption related to memory, which is very restrictive in MVC encoders. This work proposes an architecture design to accelerate the compression/decompression of the reference data required for the ME/DE search. The compression technique is based on a (1) simplified intra–prediction, adaptive (2) non-linear quantization and (3) Huffman-based entropy encoding. The hardware design is composed for an intra-prediction module that is responsible for generating the predicted samples, one module that produces the residue using the predicted samples previously generated and a module composed of non linear quantization and Huffman based entropy encoder. The synthesis results shows that the proposed hardware design accomplished the goal of accelerating the reference frame compression technique, arriving at a maximum operating frequency of 372 MHz, being able to process videos in high resolution.**

***Keywords* — Multiview Video Coding, MVC, reference frame compression**.

I. INTRODUCTION

The concept of Multiview Video Coding (MVC) [1] is part of the 3D applications, where multiple independent cameras record the same scene from different observation points. The MVC standard, the state-of-the-art in multiview video coding, provides 20-50% increased coding efficiency in comparison to the H.264/AVC [2]. Together with the Motion Estimation (ME), the Disparity Estimation (DE) represents the most power consuming module in the MVC encoder (more than 92% of the average power consumption) [3].

The ME/DE goal is to search, for each current block of the frame that is being coded, for the most similar block (best match) in one or more reference frames (previously coded frames). The search is performed within a search window, which is accessed from the Decoded Picture Buffer (DPB). The DPB is generally mapped to external memory and the search window is typically stored in a local on-chip video memory. Previous studies prove that 90% of the ME/DE power consumption is related to the external and on-chip memory issues [3]. Due to the battery-powered nature of such devices, the power consumption must be smaller as possible to allow good battery usage with multiview video handling. Therefore, it is necessary minimize the external memory accesses related to the ME/DE on MVC in order to attend mobile devices power and performance restrictions.

Previous works also aimed to implement hardware architectures to reduce the memory power consumption in the ME/DE by compressing the reference samples before storing them in the DPB external memory [4][5][6]. The solution presented in [4] is based on the compression of 2x2 blocks and utilize different prediction modes. This solution introduces extra complexity to the compress process, since the best prediction mode for each 2x2 block must be calculated. The solution proposed by [5] reduces the losses of the MMSQ [6] (that is previously proposed technique) process by storing them, so these errors can be returned to the correspondent blocks.

The goal of this work is to design an efficient hardware architecture for a low-complexity reference frame compression algorithm that aim to reduce the memory traffic in the ME/DE on MVC encoders, during the access of search window in the DPB. With this algorithm in mind, a hardware architecture that efficiently implements this technique is proposed. This hardware implementation is promised to be an acceleration path to not limit the MVC encoder loop. The algorithm and the hardware in this paper are based on a simplified intra-prediction encoder defined [1]. This enables the exploitation of already existing information that was generated during the MVC encoding process. After the intra-prediction, the residues are applied to a content-adaptive compressor path, which is composed of (a) non-linear quantization and (b) Huffman-based entropy encoder.

The hardware architecture was completely described in VHDL using the ISE software tool, developed by Xilinx for synthesis and analysis of HDL designs. Thus, it was possible simulate the designed architecture.

This paper is organized as follows: Section 2 presents a description of the efficient compressor technique based in intra-prediction. The process is shown in steps that are explained one by one. Section 3 shows details of hardware architecture, also presenting each block with the respective explanation of its operation. Section 4 contains a brief analysis of results. Finally, Section 5 concludes the paper.

II. REFERENCE FRAME COMPRESSION TECHNIQUE

The content-adaptive frame compressor technique goal is to compress the samples after they are completely encoded and reconstructed (Inverse Transforms and Quantization, Reconstruction). After that, the reconstructed samples are stored in the Decoded Picture Buffer (DPB), mapping in the external memory, to be used as reference for future ME/DE operations. The proposed content-adaptive compressor technique is described as a pseudo-code in Fig. 1. As inputs, the algorithm expects: (a) the sixteen already selected intra-prediction modes, (b) the original samples of the current 4x4 block and (c) the reconstructed reference samples.

Algorithm: Content-Adaptive Reference Frame Compression

```
1. // inputs: predModes[]: sixteen RDO MVC encoder intra prediction modes
2. //origBlock4x4[]: original 4x4 block samples
3. //reconBlock4x4[]: reconstructed 4x4 block samples
4. // outputs: codedBlock4x4[]: compressed bitstream to send to the DPB
5. function compressBlk4x4(predModes[ ], origBlock4x4[ ], reconBlock4x4[ ])
6.   codedBlock4x4 ← 0
7.   foreach reconSample, origSample in reconBlock4x4, origBlock4x4 loop
8.       predictSample ← intraPrediction(neighbors, predModes[reconSample])
9.       residueSample ← predictSample - reconSample
10.      quantizedSample ← quantization(residueSample)
11.      huffmanSample ← staticHuffman(quantizedSample, huffTable)
12.      codedSample ← packer(huffmanSample)
13.  end loop
14.  return codedBlock4x4
15.end function
```

Fig.1. Content-adaptive compression technique

The algorithm is applied for each sample that composes the entire 4x4 block (line 7). The first step is a simplified intra-prediction (line 8) with the goal of eliminating the spatial redundancies intrinsic to the reconstructed reference samples. The technique uses the best 4x4 intra mode selected by the MVC mode decision, in order to decrease the computational effort, requiring the calculation of only one predicted block. The simplified intra-prediction algorithm for the predicted samples is the same of the H.264/AVC definition for 4x4 blocks, where 9 possible modes using thirteen neighboring samples, at maximum [2]. Then, the residue sample is calculated (line 9) by the difference between the predicted sample and the reconstructed sample. Then, the non-uniform quantization is applied (line 10) to further minimize the range of representation, becoming the residue distribution much more concentrated compared to the reconstructed samples.

By definition, the non-linear quantization applies smaller quantization steps (distance between two quantization levels) for the higher probability regions along the statistical distribution (near the average). Exploiting the concentered distribution of the residue, Huffman based entropy encoder is applied (line 11) with the goal to assign the smaller codes to the most likely symbols. The last step (line 12) is to pack the residue, generated by quantization and Huffman process, and the prediction modes for all 4x4 into fixed-sized packages to be sent to the external memory (line 14).

Experimental results show that the proposed content-adaptive compression scheme is able to reduce the external memory accesses by up to 63% along with negligible losses in the MVC encoder rate-distortion performance. At the best of the authors' acknowledgement, this is the best results when compared to the available related works.

III. HARDWARE ARCHITECTURE

The proposed architectural design for the reference frame compression hardware is shown in Fig. 2. The system is divided into four main stages that will be described with more details in the following subsections.

The architecture receives, as input: (a) the prediction mode previously used during the MVC mode decision, (b) the original neighbors samples, (c) the reconstructed samples and (d) the quantization factor. First, the original neighbors samples are stored in the Neighboring Sample Buffer (NSB) and this happens when the control send the *We* signal, allowing the write operation in the buffer. Then, the next step is to generate the predicted samples. For this, we have two types of intra-prediction (*DC Prediction* and *Sample Prediction*). At the same time that NSB receives the samples, the register bank also receives the eight necessary samples to perform the DC mode. Thus, DC Prediction and Sample Prediction can work in parallel. Then, the control, that has previously received the prediction mode, sends the signal (selPrediction) signal to the multiplexer that chooses the predicted sample valid. Now, the residue is generated by the subtraction between the predict sample and the reconstructed sample. The Huffman tables and the quantization steps were statically defined based on statistics taken by using real video test sequences. They were implemented as a memory which is addressed by the residue sample. After the Huffman encoder and quantization are applied over the residue, the coded residue is packed into 16-bit fixed sized packages and stored in the external memory. In the next sections we have a detailed description of each module in a separated way.

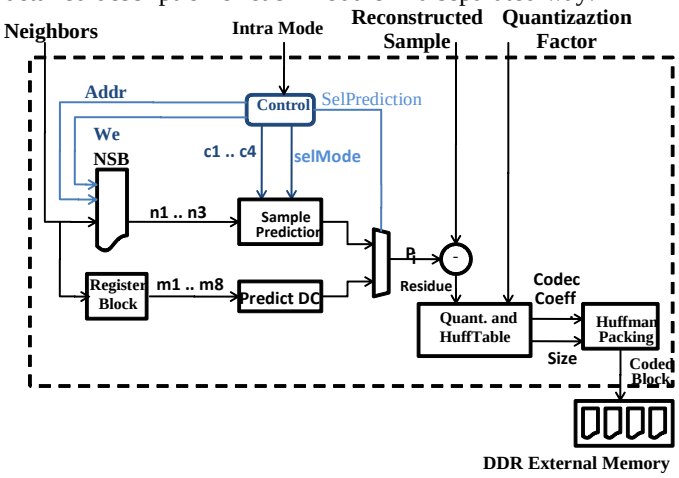

Fig. 2. Overall Architecture Block Diagram

A. *Intra-Prediction Sample and Residue Generator*

The prediction module is designed to be simple. It is a combinational logic composed of adders, multiplexers and a shift module. The Fig. 3 shows the hardware architecture responsible for the intra-prediction of one sample. This module of the architecture needs to be able to calculate nine modes of prediction, depending on the constant received from the control and the neighbors that area available. This hardware is able to calculate every mode, with the exception of DC mode that needs a separate hardware. We chose shifters and multiplexers instead of using multipliers, due to the high delay propagation caused by the multipliers. Fig. 3 also shows the Switch Mode module, which is responsible to deal with non-regular predicted samples that are not generated using the general formula (modes 3 and 8). This module performs this decision based on the *selMode* signal from the control unit. The hardware for DC mode is quite similar. In this module, the mean of the left and upper samples is used to predict the entire block. The Fig. 4 shows the hardware for intra-prediction on DC mode. As it is possible to see on the Fig. 3 and Fig. 4, three pipeline stages are implemented in both modules. This allows that three different predicted samples can be processed at the same time. Besides, the architecture is able to produce results of both at the same time and, causing no time wasting. This hardware was chosen since we have a simple combinational logic with four stages and the sums can be process in parallel (with exception of dependent stages). This way, the results in DC mode or others modes are available at same time and the choice is made by a signal that say if the intra-prediction mode is DC or others.

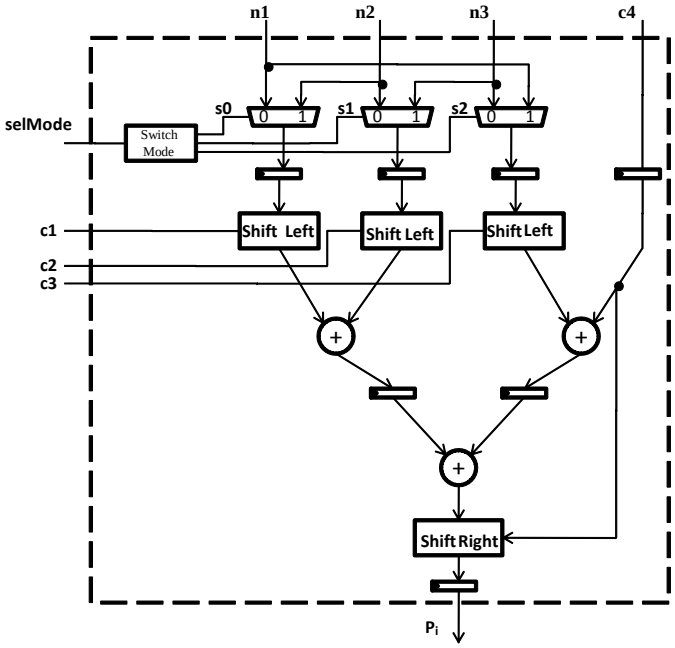

Fig. 3. Predict Sample Module

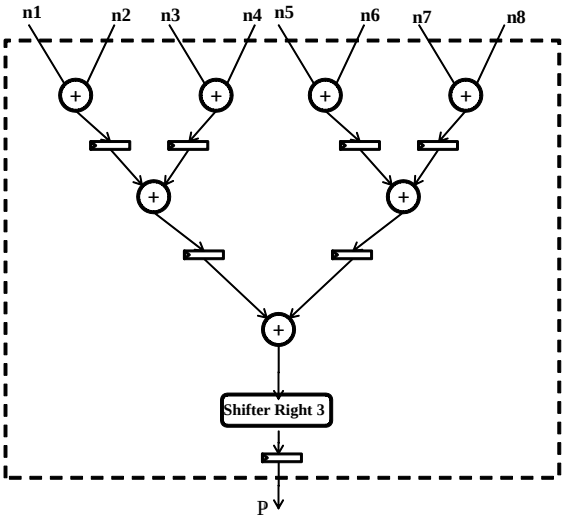

Fig. 4. Predict Sample Module for DC mode

The residue module is a simple module composed by a subtractor that calculates the difference between the predict sample and the reconstructed sample. This stage receives the predicted sample (from prediction sample module) and the reconstructed samples (external input of the compressor architecture). The output residue is immediately sent to be compress as much as possible.

B. *Neighboring Samples Buffer*

The thirteen neighbors required for the processing of one 4x4 block are stored in a memory, where they will be available to be used on during prediction. The Fig. 5a shows the memory design and the position of the neighboring samples of a 4x4 block (Fig. 5a).

Due to the large power consumption of a memory access, both off/on-chip, the reduction of the number of accesses is essential. Then, each memory position is composed of three consecutive neighbors. So, when one access is performed, three neighboring samples are accessed at the same time and using only one cycle. Each memory position in the NSB occupies 24 bits (3 x 8bits).

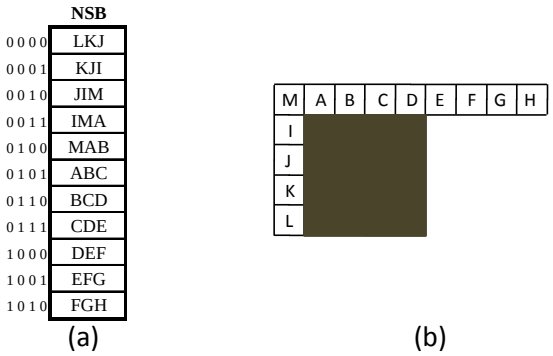

Fig. 5. Neighbors Sample Buffer and 4x4 block

C. *Non-linear Quantization and Huffman Encoder*

The static Huffman relates the input code (residue, in our case) with the fixed conversion table that is predetermined by the values of input code. In this hardware architecture, the quantization and Huffman table was mapped to a memory that was previously loaded with the correspondent codes for each input value. This was possible because both the quantization and Huffman algorithm are static, and then can be mapped in the same memory, first decreasing the range of input (residue) and then applying Huffman process. These two steps happen directly, depending of quantization factor (external output).

The Huffman codes packing, that is the last stage of this compressor, is shown in the Fig. 6.

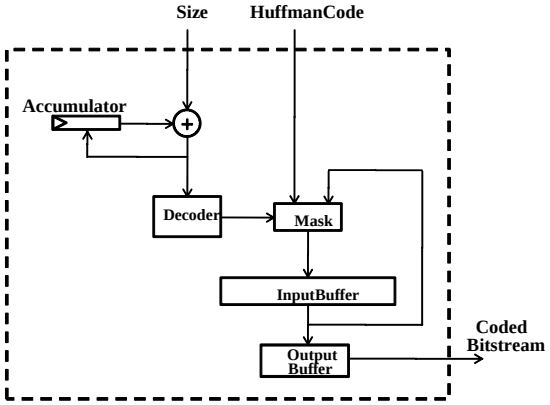

Fig.6. Huffman Packing Process

The Decoder module receives the number of valid bits on the Huffman code, and then this value is added to the accumulator which stores the sum of valid bits, that is send to Mask module. The Mask module receives the Huffman code from Huffman Table, as well as the size of valid code and is responsible for packing these valid codes with others until formed a word of 16 to 32 bits. Then, the 16 most significant bits are sent to the output buffer, delivering them to the external memory. The rest of valid bits (that are stored in 16 least significant bits) are sending to the Mask module and the process begins again.

D. *Control Unit and Pipeline Schedule*

The control unit is a regular state machine that was programmed to send a series of control words, depending of prediction mode and the samples being processed. This

control word has a length of 15 bits and is composed by different information as address of NSB, constants for intra prediction and a signal to DC or others modes.

The composition of a control word is as follows: The 4 most significant bits match the address that will be accessed in NSB. The 8 following bits are the constants (c1, c2, c3 and c4) that are necessary for intra-prediction module. The 2 following bits are used for a small select module that is present inside of intra-prediction module and it chooses the correct input for the prediction, depending of prediction mode. The least significant bit is to choose between the two types of intra prediction (sample prediction and prediction DC).

The Fig. 7 shows the pipeline schedule for the compression of one 4x4 block. Besides, the time that each module of compressor takes is represented in the x-axis.

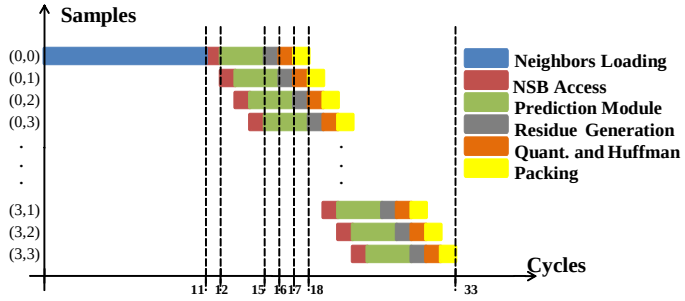

Fig. 7. Pipeline schedule of the proposed hardware architecture.

As can be seen, 11 cycles are taken for the neighboring samples loading. After this, these neighboring samples are going to be used to process the entire 4x4 block. Each sample is completely processed in 7 clock cycles. When the pipeline is full, we have one sample per cycle until the entire 4x4 block has been processed. It also can be noted that only with pipeline, without further parallelism technique, three samples can be processed at the same time during the intra-prediction.

IV. Results and Discussions

The compressor architecture was described in VHDL and synthesized looking for a FPGA implementation using the Xilinx ISE synthesis tool. The target FPGA device was the Virtex 5 XC5VLX50T. The performance evaluation takes into account the number of clock cycles that are required to compress one 4x4 block: 33 cycles. The compressor is applied to all reconstructed blocks before they are stored in the DPB (external memory). In another words, the architecture will process every 4x4 reconstructed block that will be required for future ME/DE operations. Tab. 1 presents the synthesis results considering: the number of (a) LUTs (Look-Up Tables), (b) FFs (Flip-Flops), (c) Slice FFs, (d) BRAMs (Block Random Access Memory) and (e) maximum operation frequency.

Table I
Synthesis Results for the FPGA Implementation

Logic Utilization Summary	
#LUTs	98 (1%)
#FFs	89
#Slice FFs	89
#BRAMs	1
Frequency[MHz]	372

The memories, required for the NSB and the Huffman-quantization design, were mapped to only one BRAM, impacting in a low memory usage. The maximum operation frequency that the hardware implementation is able to achieve is 372 MHz (2,687ns of critical delay). This high frequency was enabled due to the pipeline states that reduces the critical path to one adder plus a shift operation (multiplexer based).

Tab. 2 presents a performance evaluation of the designed hardware architecture. The first part of the evaluation presents the minimum frequency required to reach real time processing (30 frames per second) for several scenarios: (a) 2, 4 and 8 views and (b) VGA (640x480), XGA (1024x768) and HD1080 (1920x1080) video resolution. The rightmost part of Tab. 2 shows the maximum throughput (in frames per second) allowed by the compressor implementation over the FPGA device for the same scenarios.

Table II
Architecture Performance Evaluation

Resolution	Frequency [MHz] for 30fps			Throughput [fps] @max. freq.		
	2-view	4-view	8-view	2-view	4-view	8-view
VGA	38	76	152	293	146	73
XGA	97	194	389	114	57	28
HD1080	256	513	1026	43	21	10

Considering the Tab. 2, it can be seen that the hardware architecture developed on this work is capable to meet real time processing capability for the tested resolutions as follows: (1) 2 views in the HD1080 videos, (2) 4 views for the XGA resolution and (3) 8 views when coding VGA videos. When the compressor works at the maximum operation frequency, it is able to process (1) 8-view VGA video running at 73 fps, (2) 2-view HD1080 videos running at 43 fps, and (3) XGA videos with 4 views at 57 fps and with 8 views at 28 fps (also acceptable for real time applications).

V. Conclusions

This work presented a hardware architecture design for a reference frame compression technique for Motion and Disparity Estimation (ME/DE) on MVC. This hardware was based on a simplified intra-prediction, where the spatial redundancy is exploited. The compression also use a path composed of quantization and Huffman-based entropy encoder in order to obtain a higher compression ratio. The goal was to optimize the hardware architecture in order to achieve better compression ratios and the construction of a reference frame decompression to implement together the compressor presented in this paper. The synthesis results showed that the proposed hardware design obtained the performance expected, arriving at a maximum operating frequency of 372 MHz and is capable of processing video with high resolutions.

References

[1] Joint Draft 8.0 on Multiview Video Coding, JVT-AB204, 2008.

[2] P. Merkle, et al. “Efficient Prediction Structures for Multiview Video Coding.” In: IEEE TCSVT, v. 17, n. 11, pp. 1461-1473, nov. 2007.

[3] B. Zatt, M. Shafique, F. Sampaio, L. Agostini, S. Bampi, J. Henkel, "Run-time adaptive power-aware motion and disparity estimation in multiview video coding", IEEE DAC, pp. 1026-1031, 2011.

[4] Y. V. Ivanov and D. Moloney, “Reference Frame Compression Patterns for H.264|AVC Decoder”. ICDT 2008, pp. 168-173, July 2008, Bucharest, Romania.

[5] A. D. Gupte, et al “Memory Bandwidth and Power Reduction Using Lossy Reference Frame Compression in Video Encoding”, IEEE TCSVT , vol. 21, no. 2, pp.225-230, February 2011, Bengaluru, India.

[6] M. Budagavi, M. Zhou. “Video Coding Using Compressed Reference Frames”. IEEE ICASSP 2008, pp. 1165-1168 May 2008, Las Vegas, NV, USA.