A Hardware Solution for the HEVC Fractional Motion Estimation Interpolation

Henrique Maich[#1], Vladimir Afonso[#2], Denis Franco[#3], Marcelo Porto[#4], Luciano Agostini[#5]
[#]Federal University of Pelotas (UFPel)
Pelotas, Brazil
[1]hdamaich@inf.ufpel.edu.br
[2]vafonso@inf.ufpel.edu.br
[3]denis.franco@ufpel.edu.br
[4]porto@inf.ufpel.edu.br
[5]agostini@inf.ufpel.edu.br

***Abstract*— Nowadays many devices can handle with digital videos, especially with high definition, even for portable devices, as smartphones and tablets. However, high definition videos demand a high amount of information to be represented. The current video coding standards use a set of new techniques to increase its coding efficiency. One of these techniques, used by H.264 and HEVC (High Efficiency Video Coding), is the Fractional Motion Estimation (FME). One of the main steps of the FME is the interpolation step, which is responsible for the generation of the fractional positions. This paper presents a hardware design focusing on the interpolation step of the FME for the emerging HEVC standard. The designed architecture was described in VHDL and synthesized for Altera Stratix III FPGA. The architecture is able to generate the fractional samples for videos with QFHD (3840 x 2160 pixels) resolution in real time at 48 frames per second.**

***Keywords*— FME, Fractional Motion Estimation, FME HEVC, Interpolation HEVC, FME Interpolation**

I. INTRODUCTION

Digital videos with high definition are used in many devices, like DVD and Blu-Ray players, digital TV and smartphones, among others. Since a great amount of data should be processed, it is extremely important to compress these videos. In addition, the real time processing of high definition videos is associated with a high computational complexity, principally due to these compression techniques needed. Considering portable device applications, this high complexity is a great restriction and a dedicated hardware design is an efficient solution for this problem.

A video encoder uses different steps to reduce some types of the data redundancy presented in a video sequence. The inter-frame prediction identifies and reduces the temporal redundancy existing in near temporal frames of a video sequence. In the inter-frames prediction, there is a step called motion estimation (ME), which is the most important step in all video encoder since the ME is the one that brings more compression gains [1]. For the application of the ME, each video frame is subdivided in little blocks, called PUs (Prediction Units), before of coding. After, the ME step identifies in previously encoded frames (called reference frames), which PU are more like the PU that is being encoded. When this PU is identified on the reference frame, the ME generates a motion vector from the PU position allowing its localization. Therefore, it is necessary to calculate the differences from the PUs that are been encoded and the PUs in reference frames. Thus, only the vector and the difference between these PUs are sent to the next step of the encoder.

In the HEVC (High Efficiency Video Coding), PUs can have variable sizes, and the selection of size occurs according the results of image quality and compression rates [2]. Furthermore, it is possible to apply a refinement step in the ME, called Fractional Motion Estimation (FME). The FME can increase ME gains using sub-pixels position beyond integer positions [3]. Finally, the HEVC uses some innovative techniques to compress the motion vectors [2] rising the compression gains.

The FME step can be divided in: (a) the interpolation unit, responsible to generate fractional positions, and (b) the search for best match between the encoded block and previously blocks considering these positions generated. This paper presents the hardware design for the HEVC FME interpolation unit, considering 8x8 blocks.

II. EVALUATION WITH THE REFERENCE SOFTWARE

Since the HEVC inter-prediction complexity is very high, strategies to reduce this complexity are a relevant research topic, mainly for applications with processing and power consumption restrictions. But these strategies must maintain the quality and the compression rate in acceptable levels.

The main part of the HEVC inter-frames computational complexity is related with the variable PU size, since each PU size must be evaluated in terms of the rate-distortion cost [2] to define which the best PU size is. There are 24 PU sizes defined in the current version of HEVC, and this number can rise to 25 when the 4x4 size is enabled [2]. Then, some evaluations were done in this work aiming to discover which

are the most selected PU size in the HEVC encoding process. The evaluations were done using the version HM-8.0rc2 of the HM (HEVC Model) reference software [4].

The test configurations defined by the JCT-VC [5] were used in the evaluation process. The JCT-VC defines 8 conditions to do the tests, with combinations of high efficient and low complexity tools in 3 configurations: (a) intra-only, (b) random-access and (c) low delay [2]. The evaluations were done considering the main profile. All 24 video sequences defined in [5] were encoded for the five classes in the random-access and low-delay configurations. The intra-only configuration was not used since ME is not employed in this configuration. The results of this evaluation have shown that the most used PU size was 8x8 in almost all scenarios, followed by 16x16 and 32x32 PU sizes.

Another test was done to evaluate the losses in terms of image quality and bit-rate when using only the 8x8 PU size in the HEVC ME. This evaluation shows average bit-rate losses lower than 13.18%. The average image quality losses were lower than 0.44dB.

The last test was done to evaluate the relevance of the fractional ME when only the 8x8 block size was used. Then, the FME was disabled and all tests were done again. As result was possible to conclude that the use of FME provides an important gain in bit-rate and quality. The bit-rate gain was near to 9.73% and the quality gain was near to 0.18dB.

Considering the presented results, it was possible to verify that the FME is a very important tool in the HEVC encoder. Other important point about the FME is that it presents a low complexity when all encoder is considered. The FME calculations can be done in parallel with the integer ME of the next block and no other encoder loops are necessary. On the other hand, even providing expressive gains in compression and quality, the use of variable PU sizes increases a lot the encoder complexity, since all PU sizes must be evaluated. When only one PU size is evaluated the complexity is drastically reduced, since only one block size must be processed by further encoder steps and the decision of which is the best block is unnecessary. In this scenario, the global encoder complexity is reduced in more than 24 times, since the HEVC defines at least 24 PU sizes and also many intra-prediction modes. So, the losses in quality and compression are justified, especially for applications with processing and energy consumption restrictions.

After the analysis presented in this section, this work focused in a hardware design of the HEVC FME interpolation unit using a fixed block size of 8x8 samples.

III. Fractional ME in the HEVC

The most recent HEVC FME definitions is present in documents [2] and [6]. Those documents define the process to generate the sub-pixels positions according to the HEVC standard. Fig. 1 shows the samples of luminance in integer positions, as well as in fractional positions for the interpolation with precision of quarter pixel defined in the HEVC standard, considering 8x8 block size.

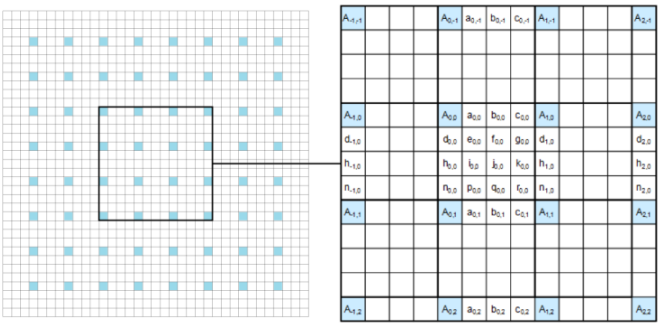

Fig. 1 Samples at integer positions (shaded squares and with uppercase) and at fractional positions (with lowercase)

According with the FME defined in the HEVC [6], it is possible to generate the fractional positions $\mathbf{a_{0,0}}$, $\mathbf{b_{0,0}}$, $\mathbf{c_{0,0}}$, $\mathbf{d_{0,0}}$, $\mathbf{h_{0,0}}$ and $\mathbf{n_{0,0}}$ in Fig. 1, with the application of an 8-tap FIR filter using only the integer positions. The generation of the fractional positions $\mathbf{e_{0,0}}$, $\mathbf{f_{0,0}}$, $\mathbf{g_{0,0}}$, $\mathbf{i_{0,0}}$, $\mathbf{j_{0,0}}$, $\mathbf{k_{0,0}}$, $\mathbf{p_{0,0}}$, $\mathbf{q_{0,0}}$ and $\mathbf{r_{0,0}}$ requires, firstly, the calculation of the fractional positions $\mathbf{a_{0,i}}$, $\mathbf{b_{0,i}}$ and $\mathbf{c_{0,i}}$, where **i** varies from **-3** to **4** in vertical direction. Then, the fractional positions generated in the vertical direction are used to generate these samples also using an 8-tap FIR filter. The used FIR filter coefficients depend on the sample position, as shown in Table I.

TABLE I
FIR FILTER COEFFICIENTS

Fractional Positions	FIR Filter Coefficients
$a_{i,j}$, $d_{i,j}$, $e_{i,j}$, $f_{i,j}$, $g_{i,j}$	{–1, 4, –10, 58, 17, –5, 1, 0}
$b_{i,j}$, $h_{i,j}$, $i_{i,j}$, $j_{i,j}$, $k_{i,j}$	{–1, 4, –11, 40, 40, –11, 4, –1}
$c_{i,j}$, $n_{i,j}$, $p_{i,j}$, $q_{i,j}$, $r_{i,j}$	{0, 1 , –5, 17, 58, –10, 4, –1}

IV. Hardware Design

An analysis of the interpolation algorithm was done in order to reduce the complexity of the hardware design. Firstly, was possible to observe a similarity among some fractional positions generation, where the FIR filter coefficients were the same in some cases. Therefore, in order to reduce the cost of the designed hardware, the sub-pixel positions were divided in 3 groups, according to the FIR filter coefficients and their positions related to the integer positions, following the same distribution presented in Table I. The names of these 3 types of filters are: Up, Middle and Down. Fig. 2, presents the location of these filters considering the interpolated positions.

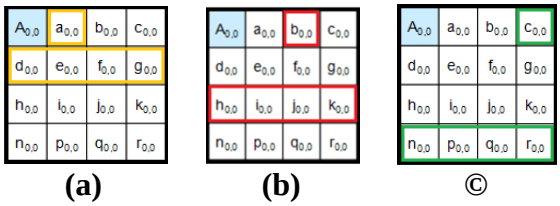

Fig. 2 Filter types (a) Up, (b) Middle and (c) Down

The next step was to optimize the filters with some algebraic manipulations. The multiplications by constants were transformed in shift-adds and common sub-expressions sharing were also considered.

The optimizations applied in the Middle filter are presented in this paper as an example, but the same process was done to the Up and Down filters. The interpolation input and output are represented as **a_0-a_7** and **s**, respectively, as presented in equation (1). The first optimization was to transform the multiplications by constants in shift-adds, where all constants are represented as numbers in base 2, as presented in equation (2). This strategy allows a multiplierless design, decreasing the interpolator hardware cost and increasing its global performance. Since each constant can be replaced by many sets of shift-adds, the selected set of shift-adds considered the lowest use of additions and, consequently, the lowest cost of hardware. Then, the multiplications by the same factors were grouped, as can be seen in equation (3). This optimization allows a reduction in the number of shifts and a reduction in the adders bit width, since the additions are done first and the multiplications are done later. Other optimization is the sub-expression share. In this case, the common calculations R_1 and R_2, presented in equation (4) are done only once and the result is reused as presented in equation (5). Finally, equation (6) presents the result of the optimization process, with the multiplications changed to shifts.

$$S = [-a_0 + 4*a_1 - 11*a_2 + 40*a_3 + 40*a_4 - 11*a_5 + \\ + 4*a_6 - a_7] >> 6 \quad (1)$$

$$s = [-a_0 + 4*a_1 - (8*a_2 + 2*a_2 + a_2) + (32*a_3 + 8*a_3) + \\ + (32*a_4 + 8*a_4) - (8*a_5 + 2*a_5 + a_5) + 4*a_6 - a_7] >> 6 \quad (2)$$

$$s = [1*(-a_0 - a_2 - a_5 - a_7) + 2*(-a_2 - a_5) + 4*(a_1 + a_6) + \\ + 8*(-a_2 + a_3 + a_4 - a_5) + 32*(a_3 + a_4)] >> 6 \quad (3)$$

$$R_1 = a_2 + a_5 \quad \text{and} \quad R_2 = a_3 + a_4 \quad (4)$$

$$s = [1*(-a_0 - a_7 - R_1) - 2*R_1 + 4*(a_1 + a_6) + \\ + 8*(R_2 - R_1) + 32*R_2] >> 6 \quad (5)$$

$$s = \{(-a_0 - a_7 - R_1) - (R_1 << 1) + [(a_1 + a_6) << 2] + \\ + [(R_2 - R_1) << 3] + (R_2 << 5)\} >> 6 \quad (6)$$

The hardware design of the interpolation filters started considering the optimized equations. Three architectural versions were designed considering structural and behavioral descriptions: (a) purely combinational; (b) using 2 pipeline stages and (c) using 4 pipeline stages. All designed architectures consume in one clock cycle the input data necessary to generate one new interpolated sample. Then, the architectures are able to generate one sample per clock cycle (when the pipeline is full for those solutions which use pipeline). The integer input samples are 8 bit-wide but the input fractional positions are 10 bit-wide. This way, as the same filter could receive integer and fractional inputs, the bit width of the filters inputs were defined as 10 bits.

The pipeline versions were designed trying to best balance the stages, splitting the adders accordingly. Fig. 3 shows the architectural design of the Middle filter using 4 pipeline stages. The other filters used the same architectural template.

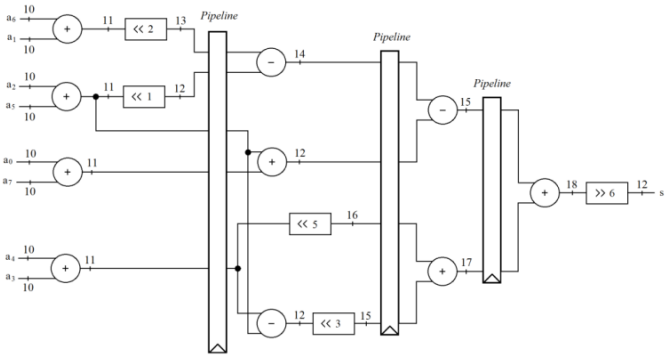

Fig. 3 Hardware architecture for the Middle filter type

The complete architecture designed for the FME interpolation is presented in Fig. 4 and it uses some instances of the three filters presented bellow and 4 buffers. Since the main goal of this work is to design a high throughput FME interpolation unit, the global architecture was designed targeting the generation of one complete line or column of fractional positions per clock cycle.

One buffer is used to store the input samples at integer positions, and the other 3 buffers are used to store the filters outputs in fractional positions. These buffers were implemented using register banks and are presented in Fig. 4, integrated with the interpolation filters. The information stored in the output filters are used in the next FME step, which is the search for the best match inside the interpolated area. The output information of the three filters is reordered inside the H-type (horizontal positions), V-type (vertical positions) and D-type (diagonal positions) buffers to be used in the next FME step. Fig. 4 also presents the filters inputs which are provided by the first buffer, containing the integer position samples, and by the H-type buffer, containing the interpolated positions used in the generation of the next interpolated samples.

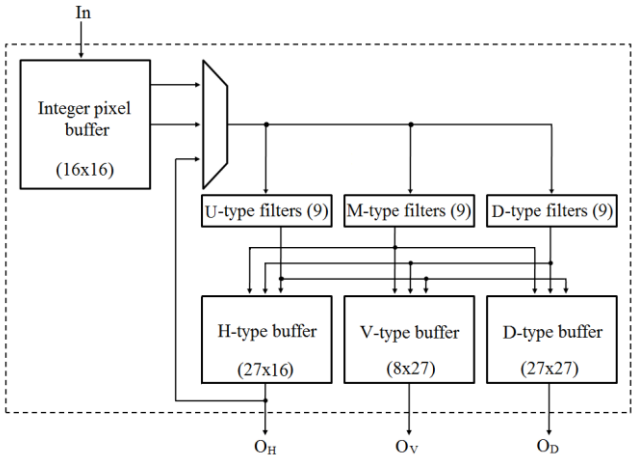

Fig. 4 Hardware architecture for the FME interpolation

The integer pixel buffer presented in Fig. 4 is able to store 256 positions with 8 bits each, because, besides the 8x8 block, it is necessary to store 4 additional samples around the block edge to allow the filtering process. Then the stored blocks with integer positions concerns 16x16 samples.

To allow the desired global architecture throughput, 9 instances of each one of the three filters (Up, Middle and Down) were used, as presented in Fig. 4. Then, it is possible to calculate 9 fractional positions per clock cycle in each group of filters. This means that 27 fractional positions (one line or column) are generated per clock cycle. It is important to notice that one 8x8 block with integer pixels generates 432 horizontal fractional position, 216 vertical fractional positions and 729 diagonal fractional positions.

The interpolation flow starts processing one line of the integer samples, where all horizontal fractional positions are calculated for this line (**a**, **b**, **c** samples in Fig. 1). This process is repeated for the 16 lines stored in the input buffer. The next step is the use of the columns of integer samples to generate the vertical fractional positions (**d**, **h**, **n** samples in Fig. 1) for this column. This process is repeated 8 times, because only 8 columns must be processed. The last step considers the horizontal fractional positions as inputs to generate the last fractional positions, called diagonal positions in this work. In this case, a total of 27 additional columns must be processed. According with the number of lines and columns that must be processed, it is possible to estimate an execution rate of 51 clock cycles to complete the interpolation of an 8x8 block.

V. Synthesis Results and Related Works

Table II presents the synthesis results achieved with the designed hardware architectures for the FME interpolation unit. These results considered all 27 filters used in Fig. 4, with 9 filters of each type (Up, Down and Middle). The VHDL description of these filters considering the three architectural versions designed in this work (combinational, two-stage pipeline and 4-stage pipeline) were synthesized targeting a Stratix III EP3SE50F484C2 FPGA device [7]. The used synthesis tool was the Altera Quartus II [7].

TABLE II
SYNTHESIS RESULTS FOR THE INTERPOLATION UNIT VERSIONS

Interpolation Filter Version	Combinational ALUTs	Total Registers	Frequency (MHz)
Combinational	10,327	16,534	131.91
Pipeline and 2 steps	10,327	17,764	208.42
Pipeline and 4 steps	10,327	19,130	317.36

Based on the operation frequency reached by the different filter versions it is possible to estimate the number of frames per second that can be processed, considering different resolutions. This estimation is presented in Table III considering Full HD (1920x1080 pixels) and QFHD (3840x2160 pixels) resolutions. Considering these results it is possible to conclude that all developed filters can process Full HD videos in real time (at least 30 frames per second), but considering QFHD videos, the combinational version did not reach the needed performance.

TABLE III
ESTIMATION OF FRAMES PER SECOND USING THE ARCHITECTURES DEVELOPED

Developed Architecture	Full HD (1920x1080) (fps)	QFHD (3840x2160) (fps)
With Combinational Filters	79	19
With Pipeline Filters and 2 steps	126	31
With Pipeline Filters and 4 steps	192	48

To the best of our knowledge, there´s no other published work targeting the hardware design for the last HEVC interpolation definitions [6]. The work [8] presents a hardware design for the HEVC FME, but it is based on an older version of the standard. The latest standardization documents presented a lot of modifications in the FME, then a fairly comparison is not possible.

VI. Conclusion

This works presents a hardware solution for the FME step considering the HEVC emerging video coding standard. This hardware design considered PU blocks with 8x8 samples, allowing a reduction in the global encoder complexity.

An analysis was done to find out which optimizations could be made in the interpolation filters. With these optimizations was possible to generate a multiplierless and high throughput architecture. The synthesis results show that this design was able to interpolate QFHD videos (3840 x 2160) in real time.

The next step of this work consists in focus on the full FME architecture, which will use the interpolation architecture design presented in this work and integrate it with the search and comparison step, which is currently at development, also considering real time processing on high definition videos for the HEVC standard.

References

[1] A. Puri, X. Chen and A. Luthra, "Video Coding Using the H.264/MPEG-4 AVC Compression Standard," *Elsevier Signal Processing: Image Communication*, n. 19, pp. 793–849, 2004.

[2] I. Kim, K. McCann, K. Sugimoto, B. Bross and W. Han, HM8: High Efficiency Video Coding (HEVC) Test Model 8 Encoder Description (JCTVC-J1002), 2012.

[3] I. Richardson, H.264 and MPEG-4 Video Compression: Video Coding for Next-Generation Multimedia. Chichester: John Wiley and Sons, 2003.

[4] High Efficiency Video Coding (HEVC) Reference Software. Oct, 2012; https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/

[5] F. Bossen, Common Test Conditions and Software Reference Configurations (JCTVC-J1100), 2012.

[6] B. Bross, W. Han, J. Ohm, G. Sullivan and T. Wiegand, High Efficiency Video Coding (HEVC) text specification draft 8 (JCTVC-J1003), 2012.

[7] ALTERA. FPGA CPLD and ASIC from Altera. Altera Web Site. Oct, 2012; www.altera.com

[8] Z. Guo, D. Zhou, and S. Goto, "An optimized MC interpolation architecture for HEVC," IEEE International Conference on Acoustics, Speech and Signal Processing, March 2012.