Hardware Design for the Interpolation Filters of the VVC Standard Affine Motion Estimation

Denis Maass, Murilo Perleberg, Vladimir Afonso, Luciano Agostini, Marcelo Porto
Video Technology Research Group (ViTech), Group of Architectures and Integrated Circuits (GACI)
Federal University of Pelotas (UFPel), Pelotas, RS, Brazil
denismaass7@gmail.com, {mrperleberg, vafonso, agostini, porto}@inf.ufpel.edu.br

***Abstract*—The interpolation filters used in the Affine Motion Estimation (AME) of the Versatile Video Coding (VVC) standard demand a large memory bandwidth and an intense computational effort. So, this work presents two different dedicated hardware implementations for the 15 interpolation filters used by the AME of the VVC standard. The first implementation adopts the use of full multiplicators, while the second implementation is a multiplierless approach, where the full multiplicators are replaced by adders and shift operations. The ASIC synthesis results for the TSMC 40nm standard cells demonstrate that the implementations using the shift-add method can achieve a decrease of 21% and 24% in total power dissipation and the number of gates respectively, in relation to the implementations using multiplicators. Moreover, the shift-add filters implementations should be able of increasing the maximum operation frequency by 22%.**

Keywords—Affine ME, VVC, Hardware, Interpolation Filters.

I. INTRODUCTION

With the popularization of the internet and the spread of mobile devices, the demand for video content has been increasing year after year. However, due to the vast amount of data necessary to represent these videos, compression techniques are required to allow efficient storage and transmission, maintaining the quality, while dealing with constraints of power and time. Currently, the state-of-the-art in video compression is the Versatile Video Coding (VVC) standard. The VVC was in development since 2015 by the Motion Picture Expert Group (ISO/IEC MPEG) and the Video Coding Experts Group (ITU-T VCEG) [1], being firstly released in July 2020.

To get efficient video coding, it is very important to explore temporal redundancies, that is, the similarity between two frames of the video. In this context, motion estimation (ME) is the main technique. The ME has been explored by almost all previous standards of video coding. However, in the VVC standard, the innovation is the integration of AME, which allows more flexibility in the prediction tool, adapting to non-translational movements, such as rotation and zooming, describing more effectively the motion in natural videos [2].

Inside the AME, an important step is fractional interpolation, which consists of generating intermediate samples between the integer samples, once the motion between two frames is not limited to integer positions [3]. Using these tools, the VVC can reach up to 33% more coding efficiency compared to its predecessor's High-Efficiency Video Coding (HEVC) [1]. On the other hand, while the HEVC supports 1/4 fractional pixel accuracy, the VVC provides a 1/16 fractional pixel accuracy [4], meaning a considerable increase in the computational complexity and demanding a large memory bandwidth. Thus, the coding may take so much time that it makes it impossible to use software implementation of interpolation filters of the VVC for real-time video applications.

An approach to deal with the complexity problem is to perform the encoding using dedicated hardware accelerators. While in the software encoding the CPU performs all the processes, the hardware encoding uses a dedicated media processor, allowing the CPU to be free to do other tasks [5]. As result, the hardware approach can encode a video with very similar quality to the software approach, using much fewer computational and power resources, conditions that make this method appropriate for mobile applications [6] and to reach real-time processing.

In this work, two different dedicated hardware architectures implementations were proposed and designed for the 15 interpolation filters used by the AME of the VVC standard. The two implementations were described in VHDL and synthesized in ASIC. The first implementation uses full multiplicators in the design of the 15 interpolation filters, while the second implementation replaces the full multiplicators with a shift-add circuit, resulting in a multiplierless approach. The ASIC synthesis results indicate that the shift-add architectures decrease by 21% and 24% the power dissipation and cell area, respectively, in relation to the implementation using full multiplicators. Preliminary results also show that the shift-add design should be able to increase the maximum operational frequency by 22%.

II. THE VVC STANDARD

Released in July 2020, the VVC video coding standard aims to improve the coding efficiency of HEVC as well as to support a wide variety of video applications, such as video with resolutions beyond the high definition, ultralow-delay streaming, and 360° immersive video [7]. To improve video compression, the VVC included a set of new tools and refined those already existent in the HEVC. Some of these tools are important to the scope of this work and will be detailed in the next subsections. Subsection II-A describes the AME, while subsection II-B describes the interpolation filters adopted inside the AME.

A. Affine Motion Estimation

The AME can recognize and represent other movements besides the translational ones represented by the traditional ME, such as zooming and rotation. The AME starts after the search that, in previously encoded frames, finds the most similar candidate block to the block being coded. Once found, the block with the best match is mapped by a set of Motion Vectors (MVs) that represents the movement executed by the reference block in relation to the current block [1]. The AME is applied in the Coding Unit (CU) level in blocks with a size of at least 16x16 samples and can use a 4 or 6 parameters model. The 4-parameter model uses the MV of the two control points which are located at the top-left and top-right corners of the CU, while the 6-parameter model uses MV from three control points, which are located at the top-left, top-right, and bottom-left corners [6], as shown in Fig. 1.

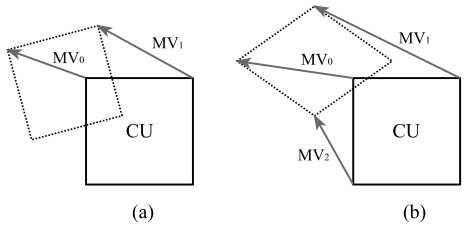

Fig. 1. The representation of the AME model: (a) 4-parameter and (b) 6-parameter (Adapted from [1]).

To reconstruct a CU coded with the AME, the CU is split into subblocks of 4x4 sample size, and each subblock is individually reconstructed. For that, the MV of the central sample of the subblock is inherited from the 4 or 6 parameters, and this MV will be used by all samples of the subblock [1]. However, as mentioned before, the motion between two temporal-neighbor frames may not be between the entire distance of two samples, becoming necessary the use of sub-pixel samples, which are provided by the interpolation filters.

B. Interpolation Filters

To generate the fractional samples to reconstruct a 4x4 subblock, and consequently reconstruct a CU coded with the AME, the VVC uses a set of 15 filters. This set of filters reaches a fractional pixel resolution of 1/16 [7], which means that between two horizontal neighbor pixels are interpolated 15 new horizontal fractional samples, and the same occurs between vertical neighbor pixels. Besides that, are calculated 15x15=225 new diagonal fractional samples, totalizing 255 samples interpolated for one integer sample [4]. The integer pixels and filters used to generate each fractional sample are shown in Fig. 2. Note that from all 255 fractional samples presented in Fig. 2, only one fractional sample pointed by the MV of the central samples of the subblock are required to reconstruct the 4x4 subblock.

The interpolation filters used in the interpolation process of 4x4 subblocks are of 6-taps. Therefore, every interpolation filter needs 6 input samples to generate its fractional sample. The horizontal fractional samples are interpolated using integer pixels from horizontally neighbors, as exemplified in Fig. 3. The vertical fractional samples are obtained from integer pixels of the vertical neighbors. Finally, the diagonal fractional samples are calculated using the horizontal fractional samples previously calculated [3].

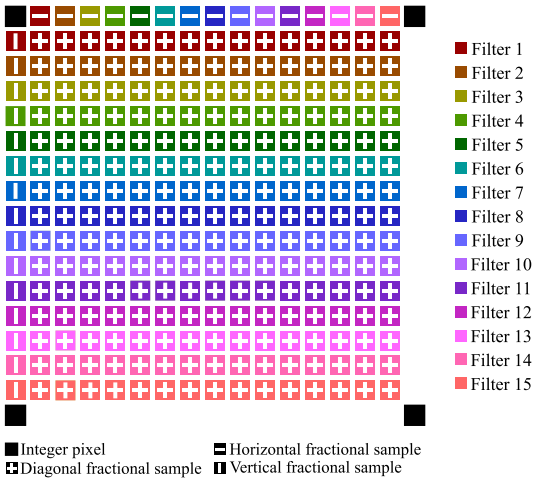

Fig. 2. Integer pixels and fractional samples (Adapted from [4])

Fig. 3. Samples used (A_x) to obtain a horizontal fractional sample (F_8)

The interpolation process occurs by computing the weighted average of the input samples ($A_{-2} \sim A_3$) by different coefficients. So, (1) gives the equation for one generic interpolation filter (F_N), where c_x represents the coefficients for filtering and A_x means the input samples. Due to the rounding properties for the weighted average, it is observed in the reference software of the VVC encoder, the VVC Test Model (VTM), that besides the weighted sum of entries, there is an offset of 32, and after all sums, the obtained value is divided by 64, which is the sum of all coefficients of the filters, resulting in the final weighted average, which is the value of the fractional sample generated.

$$F_N = (c_0A_{-2} + c_1A_{-1} + c_2A_0 + c_3A_1 + c_4A_2 + c_5A_3 + 32) / 64 \quad (1)$$

The value of those coefficients is different for each of the 15 interpolation filters and varies according to the distance between the input samples and the fractional sample to be generated. The coefficients of each input sample for the 15 interpolation filters are shown in Table I. Then, as an example, the final equation of the F_8 filter is given by (2). The equation of all others filters follows the same format, but adopting their own coefficients, as given in Table I.

$$F_8 = (3A_{-2} - 11A_{-1} + 40A_0 + 40A_1 - 11A_2 + 3A_3 + 32) / 64 \quad (2)$$

III. Interpolation Filters Hardware Design

In this work, two different implementations of dedicated hardware architectures for the 15 interpolation filters of the AME of the VVC standard are being proposed and designed. The first implementation used full multiplicators, and the second is a multiplierless approach that uses only shift-add circuits in the architecture of those 15 filters. As this work presents the first hardware implementation of these filters, the comparison of these two implementations provides interesting results of its design space exploration.

In a high-level view, both architecture models receive as the input 6 samples of 10-bit, which are weighted accordingly to the equation of the respective filter, as result, the output of the filter is a 10-bit fractional sample. The match of sizes of the inputs and the output is made because some of the fractional outputs generated may be used as input for another filter. To ensure that the output sample is truly 10-bit size, a

TABLE I. AFFINE FILTER COEFFICIENTS

Filter	C_0	C_1	C_2	C_3	C_4	C_5
F_1	1	-3	63	4	-2	1
F_2	1	-5	62	8	-3	1
F_3	2	-8	60	13	-4	1
F_4	3	-10	58	17	-5	1
F_5	3	-11	52	26	-8	2
F_6	2	-9	47	31	-10	3
F_7	3	-11	45	34	-10	3
F_8	3	-11	40	40	-11	3
F_9	3	-10	34	45	-11	3
F_{10}	3	-10	31	47	-9	2
F_{11}	2	-8	26	52	-11	3
F_{12}	1	-5	17	58	-10	3
F_{13}	1	-4	13	60	-8	2
F_{14}	1	-3	8	62	-5	1
F_{15}	1	-2	4	63	-3	1

clipping procedure is performed before the output, where the most significant bit (the signal bit) and the 9 least significant bits to produce the final sample are concatenated.

The next subsections will be used to describe in more detail the two implementations for the hardware of the interpolation filters. Subsection III-A describes the implementation using multiplicators, while subsection III-B presents the shift-add version. The description focuses on the implementation of the F_8 filter, however, the procedure is very similar for all the other filters.

A. *Multiplicators Implementation*

This architecture is a direct application of the equation of the filter, where each operation (sum, multiplication, or division) is performed by a dedicated operator. The input samples are multiplied by their respective coefficients and then are all summed, jointly with the offset of 32. Finally, the division by 64 is made.

Fig. 4 presents the architecture for the filter F_8. As can be seen in Fig. 4, the architecture for this filter requires six full multiplicators of 10 bits which produce 20-bit outputs, they are all summed to the offset into a 23-bit size number, and then, are divided by a 23-bit divider. The other filters require the same number of operations, just differing from the coefficients adopted as input in the multiplicators.

B. *Shift-Add Implementation*

The shift-add implementation consists of restructuring the filter equation, decomposing its coefficients into a set of sums of power 2 values. Thus, every multiplier operation can be performed by adds and/or binary shifts. This multiplierless approach usually requires less hardware to be implemented. Since all the multiplications and the division are made by constants, and since the multipliers and divisions can be performed with bit shift operations, so, the shift-add implementation replaces the multiplicators by circuits with a set of sums and/or binary shifts. Then, the equation (2) can be rewritten as presented in (3), where each multiplication can be performed by accumulating the results from a set of left bit shifts. Finally, (3) was reorganized so that the operations that require the same bit shift were grouped. Thus, the multiplierless implementation of the equation of the filter F_8 can be represented as (4).

$$F_8 = [(2+1)A_{-2} - (8+2+1)A_{-1} + (32+8)A_0 + (32+8)A_1 - (8+2+1)A_2 + (2+1)A_3 + 32]/64 \quad (3)$$

$$F_8 = [(A_{-2} - A_{-1} + A_3 - A_2) + (A_{-2} - A_{-1} + A_3 - A_2) \ll 1 + (A_0 - A_{-1} + A_1 - A_2) \ll 3 + (A_0 + A_1) \ll 5 + 32] \gg 6 \quad (4)$$

Fig. 5 shows the architecture developed by applying the shift-add method for the F_8 filter. Reorganizing the operations that require the same bit shift allows sharing operations between different multiplications. Thus, the architecture for the F_8 filter requires four 10-bit subtractor operators, four bit-shift operators, and seven adder operators, being one of 10 bits, two of 11 bits, one of 13 bits, one of 16 bits, one of 17 bits, and one of 18 bits. The developed architectures for the other filters followed the same optimization method to obtain shift-add implementation. Although, the number of operations performed for the other filters can have slight variations due to the differences in the filter coefficients.

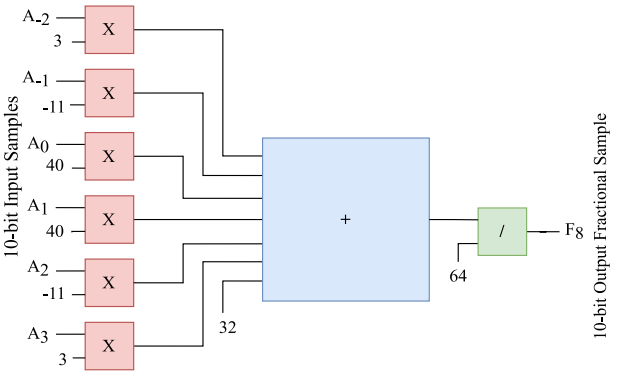

Fig. 4. F_8 filter architecture with the full multiplicators implementation

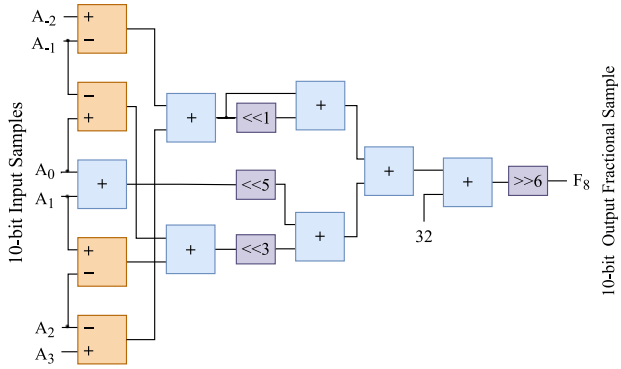

Fig. 5. Shift-add architecture for the F8 filter

IV. Synthesis Results

The 15 filter architectures were designed according to the two implementation methods previously presented and then the 30 filter architectures were described in VHDL and validated using the ModelSim tool. The architectures were synthesized to ASIC using the TSMC 40nm standard-cells library by using the Cadence RTL Compiler. All filters synthesis were made for a frequency of 100MHz, the minimum operating frequency to filter once each of the samples from a 1080p@30fps video. The obtained results, for each of the 15 filters, are presented in Table II. The area results consider the number of equivalent gates, considering the area of a NAND2 ($0.9408mm^2$).

As seen in Table II, when running at 100MHz, the filters designed with shift-add circuits show a decrease of up to 21.56% in the total power dissipation and up to 24.60% in the number of gates in comparison with the multiplicators implementation. On average for the 15 filters, the shift-add architectures present a power reduction of 13.6% and require a number of gates 10% lower. These significant reductions occur by the replacement of the full multiplicators, which requires a set of adders and additional logic gates, for the shift-add circuits.

It is important to mention that this is the first hardware implementation of the AME interpolation filters of the VVC standard in the literature. So, it is impossible to provide a fair comparison with related works in this paper beyond the one presented over the two proposed implementations.

A. *Maximum Frequency*

A preliminary investigation of the maximum operating frequency of the two proposed implementations was also performed. These preliminary results were obtained for the architectures of the filter F_8 when synthesized at their maximum frequency. The synthesis results are shown in Table

TABLE II. SYNTHESIS RESULTS PERFORMED FOR 100MHZ

Filter	Full Multiplicators implementation			Shift-Add Implementation			Δ Gains (%)			
	Leakage Power (mW)	Dynamic Power (mW)	Area (gates)	Leakage Power (mW)	Dynamic Power (mW)	Area (gates)	Leakage Power	Dynamic Power	Total Power	Area
F_1	0.015	0.207	916	0.014	0.182	876	-6.67	-12.08	-11.71	-4.35
F_2	0.016	0.222	984	0.014	0.190	893	-12.50	-14.41	-14.29	-9.30
F_3	0.015	0.211	912	0.016	0.191	942	6.67	-9.48	-8.41	3.22
F_4	0.020	0.290	1206	0.018	0.237	1053	-10.00	-18.28	-17.74	-12.71
F_5	0.017	0.270	1109	0.019	0.264	1114	11.76	-2.22	-1.39	0.48
F_6	0.020	0.283	1211	0.017	0.232	1028	-15.00	-18.02	-17.82	-15.11
F_7	0.022	0.312	1300	0.018	0.259	1086	-18.18	-16.99	-17.07	-16.48
F_8	0.021	0.299	1247	0.015	0.236	940	-28.57	-21.07	-21.56	-24.60
F_9	0.022	0.319	1318	0.018	0.257	1086	-18.18	-19.44	-19.35	-17.59
F_{10}	0.020	0.285	1216	0.017	0.232	1028	-15.00	-18.60	-18.36	-15.48
F_{11}	0.018	0.268	1134	0.018	0.253	1089	0.00	-5.60	-5.24	-4.01
F_{12}	0.020	0.274	1188	0.018	0.239	1076	-10.00	-12.77	-12.59	-9.42
F_{13}	0.014	0.211	911	0.015	0.194	933	7.14	-8.06	-7.11	2.42
F_{14}	0.017	0.227	1003	0.014	0.190	893	-17.65	-16.30	-16.39	-11.02
F_{15}	0.015	0.198	913	0.014	0.183	876	-6.67	-7.58	-7.51	-4.03
Average	**0.018**	**0.258**	**1105**	**0.016**	**0.223**	**994**	**-9.93**	**-13.85**	**-13.60**	**-10.00**

III and indicated that the shift-add architecture is capable to run at a frequency of 1169 MHz, while the architecture implemented with multiplicators can achieve a maximum frequency of 951 MHz. This means that the shift-add architecture of the F_8 filter was able to reach a frequency 22.9% higher. The results of total power dissipation and the total number of gates are also shown in Table III. Note that area and power dissipation results were increased if compared to those in Table II. It happens since to achieve higher frequencies new syntheses are necessary, and faster cells must be chosen, which demands more power and area resources.

V. CONCLUSIONS

This work presents two different dedicated hardware implementations of the interpolation filters of the AME of the VVC standard. The first implementation of the filter uses full multiplicators, and the second one is a multiplierless version only implementing shift-adds circuits. The 15 filters were designed considering both approaches, described in VHDL, and synthesized to ASIC with a TSMC 40nm standard-cells library. The synthesis results show that, at the same frequency, the shift-add architectures require up to 21.56% and 24.60% less power and gates, respectively. Besides that, the shift-add architecture can achieve a maximum frequency 22.9% higher than the multiplicators implementation. In future works, it is intended to join all the 15 filters in an interpolation architecture, being able to generate all the 255 fractional samples necessary to perform the AME.

TABLE III. SYNTHESIS RESULTS OF F_8 FILTER ARCHITECTURES AT THEIR MAXIMUM FREQUENCY

	Multiplicators Implementation	Shift-Add Implementation	Δ Gains (%)
Frequency (MHz)	951.47	1169.59	22.9
Power (mW)	2.139	2.076	-2.9
Area (gates)	1861	1623	-12.8

ACKNOWLEDGMENT

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, and algo by FAPERGS and CNPq Brazilian research support agencies.

REFERENCES

[1] P. Gonçalves. "Um esquema rápido baseado em aprendizado de máquina para a predição interquadros do codificador de vídeo VVC," *M.S. thesis*, CC, UFPel, Pelotas, BR, 2021. [Online]. Available: http://guaiaca.ufpel.edu.br/bitstream/prefix/7787/1/Dissertacao_Paulo_Henrrik_Ribeiro_Goncalves.pdf

[2] L. Li et al., "An Efficient Four-Parameter Affine Motion Model for Video Coding," in *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 8, pp. 1934-1948, Aug. 2018. doi: 10.1109/TCSVT.2017.2699919.

[3] V. Afonso, H. Maich, L. Audibert, B. Zatt, M. Porto, L. Agostini, and A. Susin, "View of Hardware Implementation for the HEVC Fractional Motion Estimation Targeting Real-Time and Low-Energy," *Journal of Integrated Circuits and Systems 2016*, v. 11, n. 2, pp. 106-120, Aug. 1, 2016. doi: https://doi.org/10.29292/jics.v11i2.435.

[4] A. CanMert, E. Kalali and I. Hamzaoglu, "A Low Power Versatile Video Coding (VVC) Fractional Interpolation Hardware," *2018 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, pp. 43-47, 2018. doi: 10.1109/DASIP.2018.8597040.

[5] J. Kufa and T. Kratochvil, "Software and hardware HEVC encoding," *2017 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pp. 1-5, 2017. doi: 10.1109/IWSSIP.2017.7965585.

[6] R. Safin, E. Garipova, R. Lavrenov, H. Li, M. Svinin and E. Magid, "Hardware and software video encoding comparison," *2020 59th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, pp. 924-929, 2020. doi: 10.23919/SICE48898.2020.9240439.

[7] B. Bross et al., "Overview of the Versatile Video Coding (VVC) Standard and its Applications," in *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 10, pp. 3736-3764, Oct. 2021. doi: 10.1109/TCSVT.2021.3101953.