VLSI implementation of the Smooth Interpolation Filter for AV1 codecs

[1]Bruna Suemi Nagai, [2]Daiane Freitas, [2]Guilherme Corrêa, [1]Mateus Grellert
[1]Embedded Computing Laboratory (ECL)) - Universidade Federal de Santa Catarina (UFSC), Brazil
[2] Graduate Program in Computing (PPGC) - Universidade Federal de Pelotas (UFPel), Brazil
bruna.nagai@grad.ufsc.br, dffreitas@inf.ufpel.edu.br, mateus.grellert@ufsc.br, gcorrea@inf.ufpel.edu.br

***Abstract*—This work proposes a hardware architecture for the Smooth fractional interpolation filters defined in the royalty-free AOMedia Video 1 codec. The filters involve several multiplication by constants, which were optimized using HCub Multiplierless Constant Multiplication algorithm. Different levels of input parallelism were described in order to evaluate resource consumption scales and the associated gain in throughput. The proposed architecture supports luminance and chrominance interpolation and was synthesized to an Altera Arria 10 FPGA and is capable of processing 302.28 Mega samples per second.**

***Index Terms*—interpolation filter, AV1, video coding, VLSI**

I. Introduction

The past years have shown exponential growth in the use of video on demand and video communication services, mainly in the period of social isolation due to Covid-19 [1] [2]. According to the Cisco VNI Forecast [3], by 2022 it is predicted to have about 4.8 billion internet users and 82 percent of that global internet traffic will be used for videos, gaming and multimedia. At the same time, an increase in video resolution and quality is being demanded by consumers as technology evolves. This has motivated the constant advances in video coding technologies.

A previously released codec called VP9 [4], developed by Google is still in large use nowadays due to its compatibility with modern web browsers and was primarily used by YouTube. It was developed in 2013 as a royalty-free solution opposing its rival, the HEVC/H.265 video coding standard [5]. HEVC was also released in 2013, but it was developed by the Joint Collaborative Team on Video Coding (JCT-VC) to replace its predecessor H.264 and carries royalties for its use [6]. The objective of HEVC was to use 50% less bitrate and maintain the same quality for the user.

As it became even more essential to have better video encoders quality and keep it as a royalty-free code, in 2015 the Alliance for Open Media (AOM) gathered scientists and companies willing to develop an advanced video codec aiming to provide high-quality video to consumers using open-source technology. Big companies like Google, Amazon, Netflix, Intel, Cisco, and Microsoft had the intention to develop a more complex algorithm that could be scalable, optimized for hardware, capable of real-time streaming, for commercial and non-commercial purposes [7]. However, one of the most important targets was to increase imaging quality using less bitrate than HEVC.

A key point to make it applicable is to deliver very high-quality imaging and at the same time, make sure the data is compressed enough to grant an efficient internet transmission and storage. However, as the technology of encoders improves, their algorithms become more complex and require more energy and computational efforts. For instance, AV1 supports up to 90 filters including luminance and chrominance samples [8], which is a considerable increase compared to HEVC with 10 filters [5]. This represents a large computing overhead for two steps of the encoding/decoding process that require interpolation: Motion Estimation (ME), present in encoders, and Motion Compensation (MC), present in encoders and decoders.

Therefore, dedicated hardware for video decoders became even more expected to evolve and make video streaming services accessible even for low internet bandwidth cases. This is specially important for the Motion Compensation step because videos are played (decoded) much more often than recorded (encoded).

With that in mind, this paper proposes a hardware architecture of Smooth interpolation filter for the Motion Compensation step AV1 video codec. All the 15 filters of the Smooth family were implemented using the HCub Multiple Constant Multiplication (MCM) algorithm to minimize area [9].

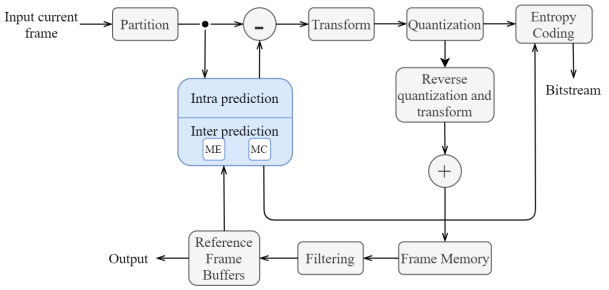
This paper is organized as follows: Section II discusses the main processes of the AV1 video encoder and interpolation filters and the proposed architecture is explained in Section III. The results and comparison with another work approach for interpolation filters are shown in Section IV.

II. Background

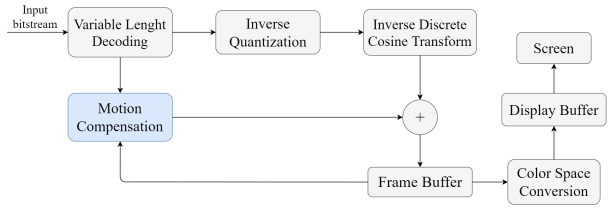
The AV1 codec follows the same traditional steps as the previous encoders, depicted in Fig. 1, although the main difference is the complexity aggregated to it due to additional options to take when encoding a video. The basis of the algorithm comes from its predecessor VP9, which was developed by Google in 2013 [6].

The codec can process video formats of 4:0:0, 4:2:0 and 4:4:4 and allows input pixels representations of 8, 10 and 12 bits [10]. The algorithm expanded the partition tree to support 10 ways of block sizes. In this step, a frame can be divided into blocks sizes ranging from 4x4 pixels to 128x128 pixels, which are the largest blocks supported by AV1, called superblocks

(SB). Rectangular shapes (tile) are also a possible option for the partitioning, although they can not be further subdivided.



(a) Encoder



(b) Decoder (Adapted from [11])

Fig. 1: Overview of video encoding and decoding processes.

.

During the block partitioning, all coding blocks are processed in intra and/or inter frame prediction modes. The first option works based on spatial extrapolation between previously decoded frame blocks and the current one. For the AV1 codec, upgrades like more granularity of directional extrapolation and enhancement of coherence chroma and luma signals were applied [12]. The inter prediction mode is based on a block matching procedure between blocks of pixels from the reference frame and the current frame.

In the AV1 inter-frame algorithm, a total of seven reference frames are required. This process is the most computational-consuming one and is composed of two steps: ME and MC [13]. The first one analyses the movement of objects between frames and obtain Motion Vectors (MV) to represent an estimated motion of them. It is usually executed in two steps: integer search (IME), where integer pixels position are compared, and fractional search (FME). This last one needs to interpolate fractional pixels positions generated around the integer samples to accomplish the search. After the ME is done, the MC step is responsible for recreating the block pointed by the MV, which includes another interpolation step if the MV has a fractional precision. The reconstructed block is combined (added) with the residue block to compensate the error generated in the inter-prediction.

Since AV1 supports $1/16$ pixel motion accuracy for chrominance samples and $1/8$ for luminance samples, sub-pixels positions are generated among integer pixels positions during ME and MC for a more precise processing [14]. This is illustrated in Fig. 2, where the integer pixels positions are shown in grey. The 63 fractional samples for one integer pixel position are also represented, both vertical (V) and horizontal (H) pixel orientations. Note that the vertical samples in the middle region require previously interpolated horizontal samples (normally called 2nd order vertical 2OV), causing a data dependency that impedes a completely parallel implementation.

The AOM codec uses an adaptive interpolation filtering structure, which is applied independently in vertical and horizontal directions. It applies a total of 4 different families of filters: Regular (6-taps filters based on Lagrange), Smooth (6-taps filters based on Hamming window), Sharp (8-taps filters based on discrete cosine transform) and Bi-linear (coding or fast decoding filters) [10], [15].

Overall, the algorithm can implement 90 interpolation filters of FIR and Bi-linear type. A recent paper indicates that for 1920x1080 resolution videos, the Sharp and Smooth filters were responsible for a significant encoded pixel count for vertical interpolation filter, particularly for 19-27 and 24-32 quantization parameter ranges [15]. In cases where the coding block dimensions are 4x4 or less, there are two additional 4-tap filters modifications of the Smooth and Regular filters, which corresponds to the middle-four coefficient taps [14]. The Smooth filter coefficients are shown in Table I. Equation (1) shows the $F0$ filter description.

$$F0 = (2a_0 + 28a_1 + 62a_2 + 34a_3 + 2a_4 + 64) >> 7 \quad (1)$$

Each filter has a rounding term (+64) and is right-shifted by 7 positions to efficiently implement a division by 128. It is possible to notice that the filter outputs are essentially multiplications by constants (i.e., coefficients) added together which is a great characteristic for hardware architectures because this type of operation can be computed using adders and shifts, saving a lot of resources compared to general-purpose multipliers.

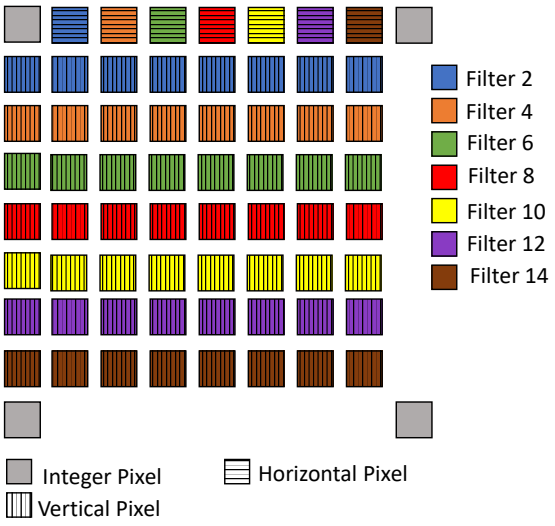


Fig. 2: Interpolation process of luminance samples in AV1 standard. (Source: [15])

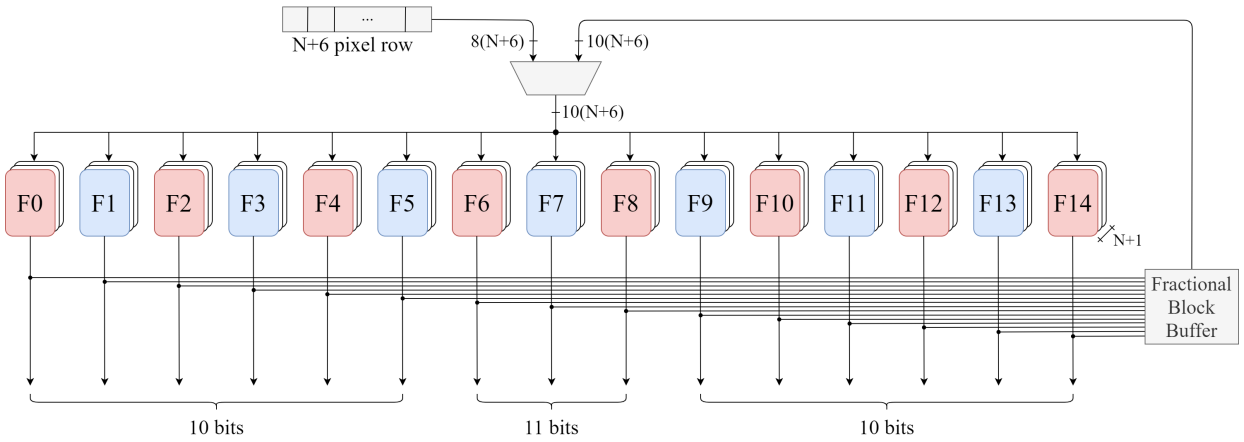


Fig. 3: Diagram of the proposed AV1 Smooth interpolation filter architecture.

TABLE I: Smooth filters Coefficients

Filter	Y	CbCr	Coefficients					
F0	**-**	**1/16**	2	28	62	34	2	0
F1	**1/8**	**2/16**	0	26	62	36	4	0
F2	**-**	**3/16**	0	22	62	40	4	0
F3	**2/8**	**4/16**	0	20	60	42	6	0
F4	**-**	**5/16**	0	18	58	44	8	0
F5	**3/8**	**6/16**	0	16	56	46	10	0
F6	**-**	**7/16**	-2	16	54	48	12	0
F7	**4/8**	**8/16**	-2	14	52	52	14	-2
F8	**-**	**9/16**	0	12	48	54	16	-2
F9	**5/8**	**10/16**	0	10	46	56	16	0
F10	**-**	**11/16**	0	8	44	58	18	0
F11	**6/8**	**12/16**	0	6	42	60	20	0
F12	**-**	**13/16**	0	4	40	62	22	0
F13	**7/8**	**14/16**	0	4	36	62	26	0
F14	**-**	**15/16**	0	2	34	62	28	2

III. Designed Architecture

This paper proposes a hardware description of the interpolation Smooth filter, used in the inter prediction step. The architecture processes all 15 filters of the Smooth family in parallel and supports all block sizes (4x4, 8x8, 16x16, 32x32, 64x64 and 128x128). Fig. 3 shows the combinational diagram proposed where the blue filters indicate the generation of interpolated luminance samples. Only the outputs of filters $F6$, $F7$ and $F8$ have 11-bit width, all the others 12 outputs have 10-bit width.

The input for the architecture proposed is a row of $N+6$ pixels of 8-bit width, where N is the parallelism level of integer pixels, and the extra 6 pixels represent the padding zone. This padding zone has $T/2$ extra pixels on each end, where T is the number of taps of the filter. An input line for N = 4 ($N_3N_2N_1N_0$), is formed by 10 pixel samples which corresponds to the 4 original pixels and 3 complementary samples ($c_0c_1c_2$, $c_3c_4c_5$) required as padding-zones on the beginning and end of the line. The same logic is applied to other values of N since it always will need more 3-sample padding zones on each end.

To process all the row in parallel, $N+1$ filters are required for each of the 15 set of coefficients. Although the input row contains samples of 8-bit width, the second data pass is inputted with the output of the first pass, which is a 10-bit width in the worst case. So a depth input of $10*(N+1)$ was implemented to support both passes. The architecture has to process all the horizontal and vertical fractional samples inside and surrounding the $N \times N$ region. Considering that one row is processed each cycle, the following equation shows how many cycles it takes to process an $N \times N$ block:

$$C_N = C_H + C_V + C_{2OV} = (N+6) + (N) + (15*(N+1)) \quad (2)$$

In (2), C_H, C_V, and C_{20V} respectively stand for the amount of cycles spent on the horizontal, vertical and 2nd order vertical samples, respectively. The extra 6 cycles spent on the horizontal part are due to the padding region required to interpolate pixels near the borders of the input block.

Since the coefficients of each filter are constant, efficient multipliers using adders and shifts can be used instead of costly general-purpose multipliers. To efficiently implement these multiplications with some degree of reuse, the HCub Multiple-Constant Multiplication (MCM) algorithm was employed [9]. The MCM has a constraint that all the constants must be multiplied by a single input value, which is not the case for our filters, so we applied it on the transposed form. This is better illustrated in Fig. 4, where the first (A0) and last (A5) modules of optimized circuits are shown as examples, but note that there are six circuits ranging from A0 to A5. The arithmetic operations are also presented in Fig. 4 in which the yellow blocks are shifts and the red blocks are negative representations. Afterward, the MCM outputs that belong to the same filter have to be summed up and shifted (following the equations in Section II).

IV. Results

The proposed architecture was described in Verilog and compiled with Quartus Prime Pro Edition for an Altera Arria 10 device, which supports a total of 427,200 adaptive logic

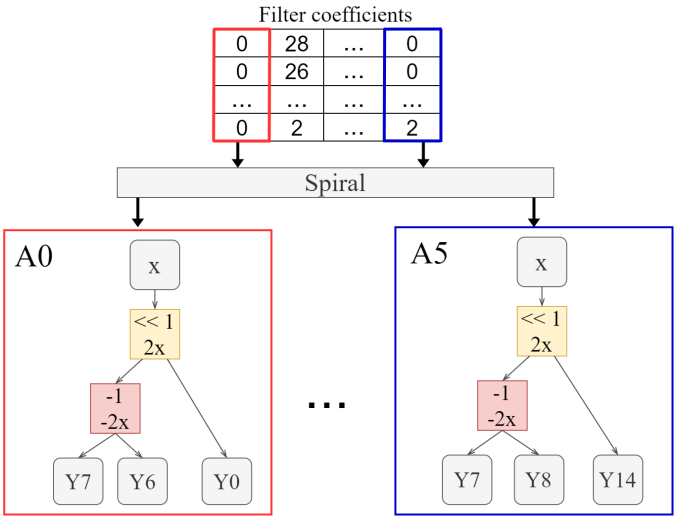


Fig. 4: Filter coefficients as inputs to Spiral tool, generating optimized circuits.

modules (ALM). As there are no Altera FPGA devices available with 21,077 (for the worst case scenario) physical pins or more, virtual pins were set in order to compile the hardware description files. The results were obtained for all parallelism levels ($N = 4, 8, 16, 32, 64$ and 128). A Python script was created to verify the architecture's correctness by comparing simulated values in Altera ModelSIM and the output from the script. Since there are not many AV1 implementations in literature, mainly if considering only the Smooth filter, it was not possible to fairly compare this paper to other works.

Table II presents the obtained synthesis results, considering each of the different levels of parallelism (N) adopted, as well as throughput results for MC operations considering a 4:2:0 chroma subsampling. It is notable that both logic utilization and total number of pins utilized increase with N. For $N = 128$, the logic utilization is almost 25× higher when compared to the parallelism level $N = 4$. Similarly, for $N = 4$ the total number of pins required for implementation is approximately 24× less when compared to $N = 128$. The number of cycles required to process a block of size N×N (C_N) was calculated according to (2). To process a block of size 8×8, 157 cycles are required, while to process a block of size 64×64, 1109 cycles are required, which results in a 7× longer cycle. The required frequency (in MHz) is inversely proportional to the value of N. For $N = 4$, the operating frequency is 76.88 MHz, 1.89× higher than the frequency required for $N = 128$ (40.53 MHz). The throughput is increased, the higher the value of N. When $N = 4$, the throughput is 13.83 M samples per second. For $N = 128$, the throughput achieved is 22× higher. The maximum resolution supported by the architecture is 2560×1600@45 fps, achieved at the parallelism level $N = 128$.

V. Conclusion

This paper features a hardware architecture of the 15 Smooth interpolation filters for the AV1 codec standard.

TABLE II: Synthesis results for the Smooth filter and throughput for MC considering a 4:2:0 chroma subsampling.

N	Logic (ALMs)	Total Pins	C_N	Freq. (MHz)	Through. (Msmp/s)	Supported MC resol.
4	3155	865	89	76.88	13.82	480p@30
8	5509	1517	157	74.01	30.17	480p@50
16	10217	2821	293	67.00	58.54	720p@50
32	19633	5429	565	64.72	117.31	1080p@30
64	38465	10645	1109	66.82	246.80	1600p@30
128	78373	21077	2197	40.53	302.28	1600p@45

The architecture was optimized using HCub algorithm and described with Verilog. All the 6 levels of parallelism implemented were synthesized to obtain the results for logic utilization and total usage of pins. The highest supported resolution was 1600p@45 fps for a block size of $N = 128$, with throughput of 302.28 Msamples per second. For further works, a multiple filter interpolation architecture is being developed.

Acknowledgment

The authors thank CNPq, CAPES Finance code 001, and Propesq/UFSC Brazilian Agencies for R&D support for scholarships and financial support to this research.

References

[1] "5 things you need to know about the acceleration of the video streaming wars," Jun 2020. [Online]. Available: https://www.thinkwithgoogle.com/consumer-insights/consumer-trends/video-streaming-wars/
[2] M. Frost, "Aom decoder," *AOM*, Mar 2021. [Online]. Available: http://aomedia.org/newsletters/aom-decoder-q1-2021/
[3] U. Cisco, "Cisco annual internet report (2018-2023) white paper," 2020.
[4] D. M. et al., "The latest open-source video codec vp9 - an overview and preliminary results," in *2013 Picture Coding Symposium (PCS)*, 2013, pp. 390–393.
[5] G. J. S. et al., "Overview of the high efficiency video coding (hevc) standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, 2012.
[6] F. Zhang, A. V. Katsenou, M. Afonso, G. Dimitrov, and D. R. Bull, "Comparing vvc, hevc and av1 using objective and subjective assessments," 2020.
[7] A. for Open Media. "home". [Online]. Available: http://aomedia.org/
[8] R. D. et al., "High-throughput multifilter interpolation architecture for AV1 motion compensation," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 66, no. 5, pp. 883–887, May 2019.
[9] Y. Voronenko and M. Püschel, "Multiplierless multiple constant multiplication," *ACM Transactions on Algorithms*, vol. 3, no. 2, 2007.
[10] P. de Rivaz and J. Haughton, "AV1 bitstream & decoding process specification. version 1.0.0 with errata 1," *The Alliance for Open Media*, p. 681, 2018.
[11] G. S. et al., "Accelerate video decoding with generic gpu," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 5, pp. 685–693, 2005.
[12] C. et al., "An overview of core coding tools in the av1 video codec," in *2018 Picture Coding Symposium (PCS)*, 2018, pp. 41–45.
[13] Chakrabarti, *Motion Estimation for Video Coding*. Switzerland: Springer International Publishing, 2015.
[14] H. et al., "A technical overview of av1," *Proceedings of the IEEE*, pp. 1–28, 2021.
[15] D. Freitas, R. da Silva, Í. Siqueira, C. M. Diniz, R. A. Reis, and M. Grellert, "Hardware architecture for the regular interpolation filter of the av1 video coding standard," in *2020 28th European Signal Processing Conference (EUSIPCO)*. IEEE, 2021, pp. 560–564.