Ultra-High Definition AV1 FME Interpolation Architectures Exploring Approximate Computing

William Kolodziejski, Robson Domanski, Marcelo Porto, Bruno Zatt, Luciano Agostini.

Federal University of Pelotas (UFPel) - Pelotas, Brasil
Postgraduate Program in Computing (PPGC)
Video Technology Research Group (ViTech)
wkolodziejski@inf.ufpel.edu.br

Abstract—Modern video encoders like the AOMedia Video 1 (AV1) implement several complex tools to allow the required high level of compression efficiency. The Fractional Motion Estimation (FME) is one of these complex tools, and AV1 FME defines 42 different interpolation filters. To handle such complexity, hardware acceleration using approximate computing has become an interesting alternative to be explored. This paper presents three optimized approximate architectures for the AV1 FME interpolation filters. The architectures reach real-time interpolation for UHD 4K videos at 30 frames per second in a low cost, low power, and memory-efficient design. The architectures were synthesized for a 40nm TSMC standard-cells technology reaching power gains up to 83% when compared to a precise architecture, and up to 20% when compared to a previously published approximate solution. The area gains were also expressive: up to 83% and 40%, respectively. The architectures also allow a memory bandwidth reduction of up to 59.5%, compared to state-of-the-art solutions. The approximations implied minor coding efficiency degradation of 0.54% and 1.25% in BD-BR. The presented architectures have the best results found in the literature when considering the trade-off among hardware cost, power dissipation, processing rate, memory bandwidth, and coding efficiency.

Index Terms—Video Coding; AV1; FME; interpolation; hardware design; approximate computing.

I. INTRODUCTION

The last decade has brought significant growth in digital video consumption, especially for video-on-demand services, live streaming, and conferences [1], and the COVID-19 Pandemic has pushed even more this growth. One clue to this assumption is the decision of Netflix, Amazon Prime, and YouTube to reduce the video quality to guarantee their quality of service [2]. Cisco estimates that by 2022 the Internet will count 5 billion users, where 70% will be mobile devices [1]. These devices will correspond to 90% of the global traffic, or about 77 EB per month (1 EB is 1,048.576 TB), and about 79% of this traffic will be related to digital video content, i.e., 61 EB [3]. Statista states that by January 2021 the global mobile data volume related to digital video streaming was about 66% [4].

This way, video streaming over the Internet is a challenging task as it requires high bandwidth and real-time delivery. This scenario increased the need for solutions that can efficiently decrease the data used to represent a video and then, video compression techniques are mandatory. The state-of-the-art video compression techniques bring a very high compression (or coding) efficiency, but they come with a very high computational cost. In this context, coding efficiency is a relation between the compression rate and the reached image quality.

As the demand for high-performance video compression continued to grow in the last decades, new video codecs emerged, such as HEVC [5], VVC [6], VP9 [7], and AV1 [8]. AV1 was released in 2018 and it is the first codec from the Alliance for Open Media (AOM), a consortium of big players in the technology market, like Google, Netflix, Facebook, Cisco, Mozilla, and others. AOM was formed in 2015 with the objective of developing an open-access and royalty-free technology with a high compression efficiency [8]. AV1 is strongly based on Google VP9 and included contributions from Cisco Thor [9], and Mozilla Daala codecs [10].

The AV1 developers reported that by 2018 AV1 allowed for about 30%-bit rate reduction over its predecessor, the VP9 [11], whereas a second work [12] estimates that the gains are about 23%, but with an impressive encoding time increase of 56 times. A more recent work [13] shows that the AV1 encoding time has significantly decreased in comparison to [12], coming to be 2.2 times faster than before. The work in [13] also compares AV1 with HEVC, showing that AV1 has an important increase in complexity when compared with HEVC, even with a smaller coding efficiency.

Given this scenario, it is not easy to develop real-time AV1 encoders with support for all its encoding tools. This is especially true for high-resolution videos. This scenario is even most challenging if the encoding process is running on battery-powered devices, where the energy consumption also must be considered. Then, the main challenge is to find a good balance between computational costs and coding efficiency, developing solutions with lower computational costs, but with minor impacts on coding efficiency. A common approach to handle this challenge is the use of approximate computing [14] since many steps of a video encoder are resilient to controlled accuracy losses [15]. Another common approach to handle the high computational costs of the video coding process is the development of dedicated hardware architectures, which can deliver solutions with much higher throughput and much lower energy consumption than the ones required by software implementations running in general-purpose processors.

The usage of approximate computing in video coding is not a novelty, and this approach has been widely explored, in other related works targeting interpolation filters, like in [16], [17], [18], and [19]. The work [16] presents an architecture for the AV1 Motion Compensation (MC), the work [17] presents an architecture for the VP9 FME, and the work
This paper is organized as follows. The next section shows an overview of AV1 improvements over the VP9 codec. Then, a detailed explanation of the Inter-frames Prediction step is presented in Section III. Section IV presents the related works. In Section V, the proposed approximations are presented and explained, and, finally, Section VI presents the proposed architectures, explaining their operation, synthesis results, and comparisons with related works. Section VII concludes this paper.

II. AV1 OVERVIEW

In an effort to create an open video format, Google launched the VP9 video codec in 2013, competing in coding efficiency with the state-of-the-art royalty-bearing HEVC codec. However, as the demand for high-efficiency video applications was still growing and diversifying, it soon became indispensable to continue advances in compression performance [25]. The AV1 is the VP9 successor, extending and improving the existing tools while proposing new tools, capable of handling diverse aspects of modern videos. AV1 also included some tools from Cisco Thor [9], and Mozilla Daala [10], as previously discussed. This section discusses some of the AV1 main concepts and tools.

Firstly, AV1 divides the frames into Super Blocks (SB) and each SB can have 128×128 or 64×64 luminance samples associated with the chrominance samples. The SB is then partitioned using a 10-way partition tree down to the 4×4 level. The partitions can be split into two vertical partitions, two horizontal partitions, 4:1/1:4 rectangular partitions, or four squared recursive partitions. The rectangular partitions cannot be further subdivided. Thus, the supported block sizes are 128×128, 128×64, 64×128, 64×64, 64×32, 64×16, 32×64, 16×64, 32×32, 32×16, 32×8, 16×32, 8×32, 16×16, 16×8, 16×4, 8×16, 4×16, 8×8, 8×4, 4×8, and 4×4 [11].

The AV1 is a hybrid encoder, like other current encoders, following a general coding flow, as shown in the diagram in Figure 1. This means that it has a prediction step (using intra-frame or inter-frames predictions), transforms (T in Figure 1), quantization (Q in Figure 1), entropy encoding, and in-loop filtering. AV1 included many novelties in each one of these steps. To guarantee that the encoder and decoder will have the same references, the encoder also has the inverse quantization (Q⁻¹ in Figure 1) and the inverse transforms (T⁻¹ in Figure 1) steps. This is necessary because quantization is a lossy process [26].

The intra-frame prediction tools are extended from VP9, exploring the potential of intra-coder: upgrading the granularity of directional extrapolation; enriching non-directional predictors by considering gradients and evolving correlations; exploiting coherence of luminance and chrominance channels; developing new tools targeting synthetic video content [25].

AV1 has a more powerful inter-frames coder, in comparison to VP9, which largely extends the pool of reference frames and motion vectors, breaks the limitation of block-based translational prediction, and enhances compound prediction by using highly adaptable weighting algorithms as well as sources [25]. In terms of fractional motion estimation, the focus of this work, AV1 has increased the number of supported filters and the subpixel precision. Since AV1 inter-frames prediction is the focus of this work, this module will be detailed in Section III.

AV1 also allows the use of transforms with multiple sizes, from 4×4 to 64×64, and supports rectangular transforms with 2:1/1:2, and 4:1/1:4 relations. AV1 also defines the use of different transforms in horizontal and vertical directions and allows the use of four transforms: (i) Discrete Cosine Transform (DCT), (ii) Asymmetrical Discrete Sine Transform (ADST), (iii) flipADST, which is an ADST in reverse order, and (iv) Identity Transform (IDTX). Then, a total of 16 combinations of vertical and horizontal transforms are allowed [11]. The AV1 quantization is similar to the VP9 quantization.

![Fig. 1 General Video Coding Diagram.](image-url)
For entropy coding, AV1 uses a symbol-to-symbol adaptive multi-symbol arithmetic coder, in contrast to VP9, which uses a tree-based Boolean non-adaptive binary arithmetic encoder to encode all syntax elements. To properly capture the coefficient distribution in the vast cardinality space, AV1 also uses a level map design for sizeable transform coefficient modeling and compression [25].

Finally, AV1 defined the use of three in-loop filters which are applied successively to a decoded frame allowing for more flexibility in signaling separate filtering levels horizontally and vertically for luminance and each chrominance plane. The AV1 in-loop filters are: (i) Deblocking Filter (DBF), (ii) Constrained Directional Enhancement Filter (CDEF) and (iii) Switchable Loop Restoration Filter (SLRF) [27].

III. AV1 INTER-FRAMES PREDICTION

AV1 inter-frames prediction has two main steps, like other video formats: Motion Estimation (ME) and Motion Compensation (MC). Within these steps, some tools are included, like (i) Warped Motion Compensation (WMC), which can handle non-translational motions through affine transformations; (ii) Advanced Compound Prediction, which can combine two predictions in different ways: compound wedge prediction, difference-modulated masked prediction, frame distance-based compound prediction, and compound intra prediction; and (iii) Overlapped Block Motion Compensation (OBMC), which explores the motion information of neighbor blocks to improve the prediction quality for pixels near its top and left boundaries [11].

Motion Estimation is a process that searches for a block, inside previously encoded frames (called reference frames), that best matches the block being processed in the current frame. When this block is found, a Motion Vector (MV) is generated pointing to the position of this block inside the reference frames, as shown in Figure 2. Then, the MC uses all MVs generated by the ME to reconstruct the predicted frame. The differences between the predicted and the original frames, called residues, are processed by the other encoder steps. The AV1 supports up to seven reference frames, as independently encoded at the bitstream [29].

Then, the AV1 FME must define, among the available filter combination, which one is the best to encode each video block, considering the coding efficiency. The number of evaluated filtering options depends on the block size. For 4×4 blocks, only Regular and Smooth filters with 4-taps are evaluated. Then, in this case, four filtering combinations must be evaluated. For blocks higher than 4×4, the filter families Regular (6-taps), Sharp (8-taps), and Smooth (6-taps) must be evaluated, in a total of nine possible combinations of filtering. In all cases, the filter family evaluation is done in three main steps: (i) evaluate Regular filters for both vertical and horizontal directions, (ii) fix the horizontal filter to be Regular and search for the best vertical filter, and (iii) fix the vertical filter to be the best filter from step (ii) and
search for the best horizontal filter [29]. The Bilinear filter is used only for fast operations, called "speed features" in AV1, and, in this case, the other filters are not available.

At this point, it is important to highlight that the interpolation processes used in FME and MC are different. The FME is applied only over luminance blocks while the MC is applied over luminance and chrominance blocks. Then the MC also has specialized filters for the chrominance samples, totaling 96 interpolation filters (48 for luminance and 48 for chrominance). On the other hand, the FME supports 48 filters, focusing only on luminance blocks. However, it is also important to emphasize that the MC and FME efforts are much different. FME must evaluate all filter possibilities to encode one block, and this means that many interpolations must be done to define which one will be used in each case. The MC, on the other hand, only needs to do the filtering process once, since the FME indicates exactly which filter must be used to reconstruct the block. Then, even though the MC supports more filters, the FME has a much higher computational cost.

Another AV1 FME novelty is that the first filter of each family is an "identity" filter, which means that when this filter is selected, then no changes are done to the filtered samples in that direction. This is useful to allow the filtering in only one direction (vertical or horizontal). Since the identity filter is only a bypass, then, in fact, there are 42 different filters in the AV1 FME implementation (7 filters for each family).

The filtering can be defined as follows [29]. Given a filter kernel $F$ and a subsample position $(w, z)$ derived from a motion vector, a prediction block can be generated with two interpolations, one horizontal and one vertical. Firstly, the horizontal interpolation generates an intermediate horizontal filtered block $H$ by applying the filter $F_w$ to the corresponding integer position block $I$ in the previously reconstructed frame, given by (1), where $n$ is the number of filters taps and $f = \frac{n}{2} - 1$:

$$H_{x,y} = \sum_{k=0}^{n} I_{(x-f+k),y} \times F_{w,k} = I_{(x+\frac{f}{2}),y} \quad (1)$$

Then, the horizontal block $H$ is used to generate the final block $V$, through the vertical interpolation, given by (2):

$$V_{x,y} = \sum_{k=0}^{n} H_{x,(y-f+k)} \times F_{z,k} = I_{(x+\frac{f}{2}),(y+\frac{z}{2})} \quad (2)$$

The horizontal blocks $V$ and $H$ correspond to the subsample position in the reference block $I$. Let $1/\epsilon$ be the precision of the block, since the FME interpolates only luminance samples, its precision goes up to $1/8$ of samples, thus $\epsilon = 8$.

An experiment was done to show the most selected filters in each direction and the results are shown in Figure 4 (a) and (b). The experiment consists in extracting, from the bitstream, the filters used to reconstruct all blocks of all sequences (presented in Table III), on the decoder side, then executing a script to count how many times the filter families were used for each direction. This experiment grouped the two Regular and two Smooth filter families. The results are presented for different quantization levels, defined by the constrained quality (CQs) with values 20, 32, 43, 55. Figure 4 also shows the results for different resolutions. In all scenarios, Regular filters tend to be more used in both horizontal and vertical directions, and the use of these filters increases according to the CQ increase.

The global usage of Regular filters, considering an average of horizontal and vertical filtering over all CQs, showed that this family is used 71.4% of the time. Considering the resolution, the use of Regular horizontal filtering decreases as the resolution increases, and the opposite behavior happens for vertical filtering. The second most used filter family is the Smooth and the third is the Sharp family, considering average results of CQs and resolutions. The Bilinear filters were not transmitted in the bitstream since the speed features were not enabled during the encoding process.

### IV. Related Works

Several related works in the literature propose hardware architecture for interpolation tools of different video formats, such as AV1, VP9-10, HEVC, and VVC. Some works propose precise architectures whereas some others propose approximate architectures. Since no similar work was found in the literature, i.e., a hardware architecture exploring approximate computing for the AV1 FME filters, the works briefly presented in the Introduction and this section were used for comparisons.

The work [16] proposes two architectures targeting the AV1 MC, considering all 96 interpolation filters defined by AV1. The architectures explore different levels of parallelism reaching a processing rate enough to process UHD 4K@30fps and UHD 8K@30fps videos. In this case, no approximations are used. Moreover, the work in [16] is focused on the AV1 MC, which is much less complex than the AV1 FME, as previously discussed. Then, the required specifications for frequency and throughput are completely different, avoiding a fair comparison.

In [21] the authors propose an architecture targeting the Sharp filters of AV1 FME and MC. The architecture processes up to UHD 4K@30fps videos but also does not use approximations. The work in [22] has the same authors as work [21], but in this case, two architectures targeting the Regular filters of AV1 FME and MC are presented. The architectures explore different levels of parallelism processing up to UHD 4K@30fps and UHD 4K@120fps and do not
use approximations. These two works focus also on the AV1 FME, as our work, but support only one family of filters each (15 filters [21] and 15 filters in [22]), whereas the solutions proposed in this paper support all AV1 filters families, in a total of 48 filters.

The work in [17] proposes two architectures targeting the VP9 FME. The architectures process videos at 720p@30fps and FHD@30fps, also without approximations. Although VP9 is the predecessor of AV1 and has the same filter names, the kernels were completely changed in the AV1 format, with an important increase in complexity. Thus, neither the filters are similar, nor the target resolutions are the same, avoiding a fair comparison.

The work in [23] proposes an architecture for the VP9-10 FME and MC, processing videos up to UHD 8K@30fps with precise filters. In the same way as the VP9 work presented in [17], the filters differ from the AV1 specifications, making impossible a fair comparison. This work presents synthesis results for UHD 4K and UHD 8K, along with real results of power dissipation.

The work in [18] proposes two interpolation architectures for the HEVC FME, reaching up to UHD 4K@60fps and UHD 8K@60fps. This work explores approximations in both filter coefficient values and the number of filter taps, being the most similar related work found in the literature but focusing on a completely different encoder. The architecture work reduces the filter taps to seven and six, reaching memory communication reductions and reducing hardware costs. However, HEVC has only three filter families, making the hardware much simpler than the one required for AV1 FME. Once more, a fair comparison is not possible.

The work in [24] proposes a reconfigurable interpolation architecture for the VVC MC, which can process up to UHD 4K@66fps and UHD 8K@60fps. This work explores approximations in both filter coefficient values and the number of filter taps, being the most similar related work found in the literature but focusing on a completely different encoder. The architecture work reduces the filter taps to seven and six, reaching memory communication reductions and reducing hardware costs. However, HEVC has only three filter families, making the hardware much simpler than the one required for AV1 FME. Once more, a fair comparison is not possible.

Finally, our previous works [19] and [20] propose two approximate architectures targeting the AV1 FME. Both works can process up to UHD 8K at 30fps videos. The work [19] was the first one in the literature to explore approximate computing in the AV1 FME, where the filter taps were approximated to hardware-friendly values. The work in [20] also explores such approximations at the taps’ values, but with an additional approximation, which reduces the number of taps in the filters. In this case, the number of filter taps was reduced from eight to only four, leading to a substantial impact in terms of area, power, and I/O bandwidth reduction. On the other hand, the coding efficiency was also impacted considerably.

V. EXPLORED APPROXIMATIONS

To reduce the complexity of the AV1 FME, two major approximation approaches were proposed in this paper: (i) simplify the filter coefficients and (ii) reduce the number of filter taps. The proposed implementations considered the seven filters for each family, in a total of 42 filters, supporting also the six identity filters. The FIR filtering process is essentially a group of multiplications of the input samples by a constant. The number of constants (and input samples) will define the number of filter taps, and the values of each group of constants vary according to the filter family and type. Approximating the coefficients allows for easy conversion of the multiplications into shift-adds, and according to the approximation level, it also allows for a reduction in the number of required additions. Reducing the number of taps will lead to a reduction in the number of required multiplications, which also reduces the required input bandwidth.

A. Approximated Filter Coefficients

The approximation explored in the filter coefficients is intended to reduce, as much as possible, the number of calculations required to generate the filtered samples. To achieve this, the coefficients were transformed in power of two \(2^n\) values, with \(n\) ranging from 0 to 7. In this case, the multiplications were eliminated and only a single shift is applied over each input sample. Negative numbers are also considered since a controlled subtraction over the shift outputs is also available. However, this solution resulted in a prohibitive loss in coding efficiency, with losses up to 4.8% in BD-BR [30] for FHD and UHD 4K resolutions, on average, according to experiments conducted. Then, an alternative solution was investigated.

The second solution considered the use of two additions or subtractions over the shifted samples to represent the coefficients. However, this solution was not used in all coefficients, due to the increase in the area caused by the addition of extra hardware. Only the two central coefficients are defined by these two additions or subtractions, whereas the other coefficients are defined by a single power of two, as in the first solution. Comprehending the interpolation as a weighted average, the central coefficients are the most important to the final results, since they have the highest values in amplitude, and multiply the most important samples (the ones closest to the sample being filtered). This way, the external taps realize the multiplications through a single shift, whereas the two central taps can use different combinations of shifts to be added or subtracted.

One important aspect must be highlighted at this point. The sum of all coefficient values in each filter must be equal to 128. This is required since after the multiplication and accumulation steps a division by 128 is made, keeping the interpolation gain equal to one. This way, all the approximate filters respect this constraint. The value 128 is defined by the AV1 as part of its standard and corresponds to the sum of all filter coefficient values.

One example of the approximations applied at the central samples calculations is done for the precise coefficient value 76, which was approximated to 84. In this case, let \(p\) be a pixel sample value. The operation is computed as:

\[
84 \times p = (((p \ll 2) + p) \ll 2) + p \ll 2
\]  

(3)

In this case, an imprecise result was reached since the final result is \(p \times 84\) instead of \(p \times 76\). Some operations are less complex, requiring only one shift, such as multiplying \(p\) by 2, which can be precisely computed as \(p \ll 1\).
The values available for the filter coefficients were manually selected, intending to better balance the approximation losses and the computational cost. This solution was presented in the authors’ previous work [19]. For simplicity’s sake, this architecture was called Approximate Multiplierless with 8-taps (AM8), in this work. Table I shows a comparison with one filter for each family, considering the 4/8 precision. A precise version and its respective approximate version in the AM8 are presented. Unfortunately, due to space restrictions, it is not possible to present all proposed approximated filters.

B. Reduced Filter Taps

The second approximation investigated in this paper was to reduce the number of taps in the FME filters with the highest number of taps. This idea was firstly investigated by the authors’ previous work [20]. This approximation was applied over the AM8 approximation and it is the biggest improvement over the architecture proposed in the authors’ previous work [19]. As explained before, the number of taps in a filter defines the number of required multiplications and the number of required inputs. Then, the filters with a higher number of taps will require large I/O bandwidth for the complete FME.

In general, to filter a \( w \times h \) block of samples, a filter with \( n \) taps will require \( (w + n - 1) \times (h + n - 1) \) samples to perform the interpolation. Figure 5 presents an example of samples required to process a 4\( \times \)4 row when an 8-tap filter is applied. To generate one interpolated sample eight samples are required. However, since there are four samples to interpolate at each row, 11 samples will be required, due to the samples’ offsets. To process a complete 4\( \times \)4 block in horizontal and vertical directions, then an 11\( \times \)11 input matrix (or 121 samples) is required \( (n = 8 \text{ and } w = h = 4) \). If the number of taps of this filter is reduced to 6 \( (n = 6 \text{ and } w = h = 4) \), then the reference matrix is reduced to 9\( \times \)9 (or 81 samples), which implies an I/O bandwidth reduction higher than 33\%. In this case, the required multiplications are also reduced from 128 to 96, considering this 4\( \times \)4 block.

Two versions with reduced taps were evaluated in this work, one considering the maximum use of 6-taps and the other considering the usage of 4-taps as maximum. The most aggressive solution, using only 4-taps, was firstly presented in [20]. The solution with the best balance between hardware cost reduction and coding efficiency losses, allowing filters with 6-taps, is a novelty of this work. The solution published in [20] was named, in this work, Approximate Multiplierless with 4-taps (AM4), and the novel solution presented in this work was called Approximate Multiplierless with 6-taps (AM6). These architectures extend the AM8 strategy, used in the authors’ previous work [19], with the two central samples of each filter being defined as two additions or subtractions and shifts, and the other ones defined using only one shift. Moreover, the number of filter taps was reduced, in order to reduce I/O communication. However, for both new solutions, AM6 and AM4, some filter coefficients needed to be redefined. The two central coefficients remain the same, whereas the other coefficients are adapted to maintain the filter gains equal to one.

The reduction in the I/O bandwidth tends to have a proportional impact on the system memory bandwidth since the required samples are typically stored in memories. However, estimating the final impacts in terms of memory bandwidth reduction is not easy, since many variables must be considered, including the used technology to implement memory, and the used memory hierarchy, among others.

Both AM6 and AM4 solutions do not have negative impacts on filter families Regular with 4-taps, Smooth with 4-taps, and Bilinear. The restriction to use a maximum of 6-taps also does not impact the families Regular with 6-taps and Smooth with 6-taps, only impacting on family Sharp with 8-taps. On the other hand, when the number of taps is restricted to four, the families Regular with 6-taps, Smooth with 6-taps, and Sharp with 8-taps are all impacted, since in all cases the taps are reduced to four. Figure 6 shows an example of the input matrix reduction when using only six or four taps instead of the original eight taps required by the Sharp filter.

Considering the use of only 4-taps filters, the Regular and Smooth filters, which have two families of filters, one with 6-taps and the other with 4-taps, will be reduced to only one family of filters with 4-taps. This happens because the approximation process applied over the 6-taps filters will generate the same filters of the 4-taps families. This means that the number of different filters is reduced from 42 to 28.

<table>
<thead>
<tr>
<th>Filter</th>
<th>Version</th>
<th>TAPS</th>
<th>Coefficients</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular</td>
<td>Precise</td>
<td>6</td>
<td>2 -14 76 76 -14 2</td>
</tr>
<tr>
<td></td>
<td>AM8</td>
<td></td>
<td>4 -32 84 84 -16 4</td>
</tr>
<tr>
<td></td>
<td>Precise</td>
<td>4</td>
<td>-12 76 76 -12</td>
</tr>
<tr>
<td></td>
<td>AM8</td>
<td></td>
<td>-32 84 84 -8</td>
</tr>
<tr>
<td>Sharp</td>
<td>Precise</td>
<td>8</td>
<td>-4 12 -24 80 80 -24 12 -4</td>
</tr>
<tr>
<td></td>
<td>AM8</td>
<td>8</td>
<td>-2 8 -32 84 84 -16 4 -2</td>
</tr>
<tr>
<td></td>
<td>AM6</td>
<td>6</td>
<td>-4 -32 84 84 -16 4</td>
</tr>
<tr>
<td></td>
<td>AM4</td>
<td>4</td>
<td>-32 84 84 -8</td>
</tr>
<tr>
<td>Smooth</td>
<td>Precise</td>
<td>6</td>
<td>-2 14 52 52 14 -2</td>
</tr>
<tr>
<td></td>
<td>AM8</td>
<td>6</td>
<td>4 16 44 44 16 4</td>
</tr>
<tr>
<td></td>
<td>Precise</td>
<td>4</td>
<td>12 52 52 12</td>
</tr>
<tr>
<td></td>
<td>AM8</td>
<td>4</td>
<td>32 44 44 8</td>
</tr>
<tr>
<td>Bilinear</td>
<td>Precise</td>
<td>2</td>
<td>56 72</td>
</tr>
<tr>
<td></td>
<td>AM8</td>
<td></td>
<td>44 84</td>
</tr>
</tbody>
</table>
Table II presents the number of samples required to interpolate polated block sizes using all available filter taps (as in the precise or in the AM8 versions) and with approximate filters (AM6 and AM4), as well as the percentage savings from such approximations. The rectangular block sizes were omitted, but they have proportional results. As can be observed, the smaller the block size, the higher the percentage impact with the reduction in the number of taps.

C. Evaluation of Approximation Impacts

The first evaluation of the approximated filter impacts was made with an experiment to get the frequency responses of the original filters and the three approximated filter versions: AM8, AM6, and AM4. The reached results are presented in Figure 7. The curves were generated for the 4/8 filters precision, as presented in Table I. As can be observed, the approximated versions reach the same results for three filter families: Regular with 4-taps, Smooth with 4-taps, and Bilinear. This occurs because these filters are not impacted by the approximations added at the AM6 and AM4 versions, as previously discussed. Then the curves for AM8, AM6, and AM4 are identical for these filter families. Observing the results in Figure 7, it is possible to conclude that the defined approximations do not significantly affect the frequency response of the major filter families. The exception is the two Smooth families where the approximations caused an erratic distortion in the filter responses.

A second evaluation considered the coding-efficiency impacts of the proposed approximations when compared to the original filters. To do so, experiments using the AV1 reference software, the libaoam version 2.0.0 [27], were made. The approximated filters were implemented in the libaoam software, and the evaluation was executed using 11 video sequences from Mozilla and Netflix data set, following the Video Codec Testing and Quality Measurement (VCTQM) [31]. Four CQ parameters \{20, 32, 43, 55\} were used and all AV1 encoding tools were enabled. Then, four sets of results were generated, one for the original filters, and one for each version using approximated filters: AM8, AM6, and AM4. Then, a total of 176 executions were done, considering the different sequences, CQs, and algorithmic versions.

The coding efficiency evaluation used the Bjøntegaard Delta Bit Rate (BD-BR) [30] metric. This metric indicates the percentage impacts, in the amount of data required to represent a video (bitrate), for two different encoder implementations to reach the same objective quality (PSNR). Positive values indicate losses whereas negative values indicate gains. Table III presents the results of coding efficiency, for each proposed approximation in comparison to the original filters, grouping the videos by resolution, as well as the average results.

The first observation from Table III is that the BD-BR results do not impact highly the coding efficiency, since even with the most aggressive approximation (AM4), the average impacts in BD-BR are 1.25%. When considering the other versions (AM8 and AM6), the overall BD-BR losses are only 0.54%. Moreover, some results are negative, indicating that, for that specific video sequences, the approximations improved the coding efficiency.

Another observation from Table III is that the BD-BR results for AM6 and AM8 versions are identical. The reason for such behavior is that the reduction from 8 to 6-taps affects only the Sharp filter, since all other filter families have up to 6-taps, as can be observed in Table I. Since the Sharp filter family is not used frequently, as can be observed in Figure 4 (a) and (b), this small approximation did not affect the coding efficiency at all.

Table III also shows that UHD 4K videos presented the best results, with the lowest BD-BR impacts. This is an interesting result, since the UHD 4K sequences are the ones with the highest usage of the Smooth filters, as can be observed in Figure 4 (a) and (b). Then, even with the worst frequency response of these filters, as presented in Figure 7, the results were not negatively impacted.

Finally, the relation between objective quality (PSNR) and bitrate (bps) is presented in Figure 8, extracted from the se-

Table II. Samples Required per Block Size

<table>
<thead>
<tr>
<th>Block Size</th>
<th>8-taps</th>
<th>6-taps</th>
<th>4-taps</th>
<th>6-taps</th>
<th>4-taps</th>
</tr>
</thead>
<tbody>
<tr>
<td>128×128</td>
<td>18.225</td>
<td>17.689</td>
<td>17.161</td>
<td>2.94</td>
<td>5.83</td>
</tr>
<tr>
<td>64×64</td>
<td>5.041</td>
<td>4.761</td>
<td>4.096</td>
<td>5.55</td>
<td>18.74</td>
</tr>
<tr>
<td>32×32</td>
<td>1.521</td>
<td>1.369</td>
<td>1.225</td>
<td>9.99</td>
<td>19.46</td>
</tr>
<tr>
<td>16×16</td>
<td>529</td>
<td>441</td>
<td>361</td>
<td>16.6</td>
<td>31.75</td>
</tr>
<tr>
<td>8×8</td>
<td>225</td>
<td>169</td>
<td>121</td>
<td>24.8</td>
<td>46.22</td>
</tr>
<tr>
<td>4×4</td>
<td>121</td>
<td>81</td>
<td>49</td>
<td>33.0</td>
<td>59.5</td>
</tr>
</tbody>
</table>

Table III. Coding Efficiency Results

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Resolution</th>
<th>BD-BR (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AM8</td>
<td>AM6</td>
</tr>
<tr>
<td>Arena of Valor</td>
<td>0.31</td>
<td>0.31</td>
</tr>
<tr>
<td>Market Place</td>
<td>1.41</td>
<td>1.41</td>
</tr>
<tr>
<td>Square and Time-lapse</td>
<td>1.33</td>
<td>1.33</td>
</tr>
<tr>
<td>Tunnel Flag</td>
<td>2.84</td>
<td>2.84</td>
</tr>
<tr>
<td>Foreman</td>
<td>0.27</td>
<td>0.27</td>
</tr>
<tr>
<td>Coastguard</td>
<td>-0.16</td>
<td>-0.16</td>
</tr>
<tr>
<td>Cactus</td>
<td>0.19</td>
<td>0.19</td>
</tr>
<tr>
<td>Bar Scene</td>
<td>-1.54</td>
<td>-1.54</td>
</tr>
<tr>
<td>Boxing</td>
<td>0.88</td>
<td>0.88</td>
</tr>
<tr>
<td>Face</td>
<td>0.27</td>
<td>0.27</td>
</tr>
<tr>
<td>Motorcycle</td>
<td>0.22</td>
<td>0.22</td>
</tr>
<tr>
<td></td>
<td>FHD</td>
<td>1.47</td>
</tr>
<tr>
<td>Average</td>
<td>UHD 4K</td>
<td>-0.07</td>
</tr>
<tr>
<td></td>
<td>UHD 8K</td>
<td>0.24</td>
</tr>
<tr>
<td></td>
<td>Overall</td>
<td>0.54</td>
</tr>
</tbody>
</table>

Fig. 7 Filters Responses.
quence Tunnel Flag, which presents the highest coding efficiency losses when using the proposed method. As can be observed, the curves are very close, indicating that even with the impacts on coding efficiency, the quality and bitrate do not vary much from the original curve. A visual example is also presented, in Figure 9. In this figure, a zoom is done over the red rectangle of the first image to compare the results for the three approximated versions and the original one. As can be observed, the visual changes are almost imperceptible.

VI. DESIGNED ARCHITECTURES

Three architectures are presented in this paper, exploring the three levels of approximations previously discussed: AM8, AM6, and AM4. The AM8 and AM4 are architectures previously proposed by the authors’ works [19] and [20], respectively. As previously discussed, the main ideas explored with these approximations are: to change the filters coefficient values to multiplier-less hardware-friendly ones; and change the number of taps in the filters to reduce the required input samples. All three architectures follow the idea of splitting the input block into $4 \times 4$ sub-blocks.

A. Definition of the Basic Partition Size

One challenge to implement the AV1 FME is the number of different block sizes supported. A total of 22 block sizes are supported by the AV1 FME, ranging from $128 \times 128$ down to $4 \times 4$, with different rectangular partitions, as explained in Section II. To deal with it, each block can be decomposed into smaller sub-blocks, allowing for regular processing and memory access [23]. The best sub-block size should take into account parameters such as memory bandwidth, samples to process, and parallelism level. Considering the need to support bigger block sizes composed of small sub-blocks, the use of the smallest block size supported by the AV1 ($4 \times 4$) is the best option.

Using this block size, all bigger block sizes can be processed by correctly splitting the input matrix into $n \times 4 \times 4$ sub-blocks. Figure 10 shows an example of an $8 \times 8$ input block split into four $4 \times 4$ sub-blocks. An important notice is that the input matrix, in this case, $15 \times 15$, will be split into four $11 \times 11$ matrices, where some samples will overlap among the blocks, and this overlapping can be explored with local memories to avoid extra communication bandwidth.

B. Basic Filtering Architectures

The architectures were designed to compute the filtering of one sample in one direction, in a multifilter fashion, supporting all 48 filtering possibilities (implementing the 42 filters and supporting the bypass required by the identity filter). Then, the architectures were called Multi filter AM8 (MF-AM8), Multi filter AM6 (MF-AM6), and Multi filter AM4 (MF-AM4).

As explained before, the approximation explored in AM8 and AM4 was firstly explored in the authors’ previous works [19] and [20]. However, the architectures MF-AM8 and MF-AM4, as well as the architecture MF-AM6, proposed in this work are used to compose a new complete FME architecture, which explores different levels of parallelism, operational frequency, and throughput, when compared to the solutions in these previous works.

To design a complete AV1 FME interpolation architecture, many instances of the MF architectures must be grouped. The number of instances is a function of the desired parallelism level and throughput, and can easily be combined to obtain different settings of frequency, area, and power. The next section will present three complete AV1 FME interpolation architectures using the multifilter designs presented in this section.

Figure 11 presents the three multifilter architectures. As can be noticed, the number of inputs is a function of the number of taps allowed for each solution. Another observation is that the four central samples are processed by the same hardware in the three architectural versions, then the MF-AM4 is included inside the MF-AM6, which is included inside the MF-AM8. On the other hand, the peripheral samples of MF-
AM8 and MF-AM6 are processed differently in the function of the applied optimizations.

The three multifilter architectures are completely combinatorial and were implemented in a multiplier-less fashion, as observed in Figure 11. All filters are computed using sub-expression sharing with combinations of single shifts, shift-adds and/or shift-subs. Some outputs have a controlled subtractor to change the output signal according to the filter definition, and these operations are signalized as C2 modules in Figure 11.

The accumulation of the partial results is made with an adder tree. Since the number of partial results is a function of the number of allowed taps, the adder tree will have different depths. The MF-AM8 will have an adder tree with three levels and seven adders, the MF-AM6 will also have an adder tree with three levels but using five adders and the MF-AM4 will have an adder tree with two levels and with only three adders. The adder trees were omitted in Figure 11 for simplicity, so they are represented with a single big adder. The final division by 128 is made with a right-shift of seven bits, as presented in Figure 11.

C. Complete AV1 FME Interpolation Architectures

Three versions of complete AV1 FME interpolation architectures were designed to explore the three multifilters presented in the last section. These architectures were called AV1 FME Interpolation using AM8 (AFI-AM8), AV1 FME Interpolation using AM6 (AFI-AM6), and AV1 FME Interpolation using AM4 (AFI-AM4). The designed architectures are presented in Figure 12.

The input matrix, represented by black boxes in Figure 12, is given by $n + 3$, where $n$ is the number of taps used in the filters. The architectures were designed to process $4 \times 4$ blocks, filtering one row of this block per clock cycle. Since the AV1 defines a horizontal filtering followed by a vertical filtering, four instances of the multifilter architectures will be required for horizontal filtering, and four for vertical filtering. In Figure 12, the H and V that precedes the multifilter architecture name were used only to identify horizontal (H) and vertical (V) filtering. The architectures are identical.

A Shift-Register-Chain (SRC) is used between each horizontal and vertical multifilter architecture to synchronize the operations. Let $n$ be the number of taps in the filters, the SRCs have the same size of $n$: eight, six, or four positions, depending on which is the used multifilter solution. Each SRC stores the $n$ horizontally filtered results, receiving one sample per clock cycle, and, after $n$ cycles, the vertical filtering can start.

The number of cycles required to completely interpolate a $4 \times 4$ block is defined as follows. The horizontal filtering requires one cycle to filter or to bypass each input matrix row, in all cases. Then, let $n \in \{11, 9, 7\}$ be the width and the height of the input matrix, the horizontal filtering architectures will take $n - 3$ cycles to fill the SRCs. The other three cycles will be necessary, in all cases, to finish the vertical filtering. Then, $n$ cycles are necessary to finish the $4 \times 4$ block interpolation.

In this case, the AFI-AM8 architecture will require 11 cycles to process one $4 \times 4$ block, the AFI-AM6 version will use 9 cycles to interpolate the same input block and the AFI-AM4 version will use 7 cycles.

The parallelism of the designed architectures can be easily explored to increase the desired throughput. In this case, as many instances as necessary of the AFI-AM8, AFI-AM6, and AFI-AM4 architectures can be used, each one processing one $4 \times 4$ input block. Then, the presented architectural solutions can reach diverse processing rates, targeting distinct video resolutions and frame rates.

Fig. 11 Multifilter Interpolation Architectures Using AM8 (MF-AM8), AM6 (MF-AM6), and AM4 (MF-AM4).

Fig. 12 AV1 FME Interpolation Architectures Using MF-AM8 (AFI-AM8), MF-AM6 (AFI-AM6), and MF-AM4 (AFI-AM4).
D. Synthesis Results and Related Works

The synthesis was made considering four architectures, the AFI-AM8, AFI-AM6, and AFI-AM4, and also a precise version using multipliers, to allow for a better comparison. The precise architectural version follows the same architectural template used in AFI-AM8, where the multilfilter instances were replaced with precise versions.

All architectures were described in VHDL and synthesized for a 40nm TSMC standard-cells technology with 1.1V [32] using the Cadence RTL Compiler tool [33]. The power results were generated using the default tool switching activity (20%). The gate count was calculated based on 2-input NANDs size (0.9408µm²).

Three syntheses were made targeting different resolutions and frame rates, to better demonstrate the flexibility of the proposed solution. The three target throughputs were: (i) FHD videos at 30 frames per second, (ii) UHD 4K videos at 60 frames per second, and (iii) UHD 4K videos at 30 frames per second.

Using only one instance of the designed FME interpolation architecture, an operation frequency of 1.3GHz is necessary to process FHD videos at 30 frames per second. Then, this was the first synthesis for the three architectural versions (AFI-AM8, AFI-AM6, and AFI-AM4). Now, using the same operating frequency of 1.3GHz, but with eight instances of the designed FME interpolation architectures, it is possible to reach a processing rate enough to process UHD 4K videos at 60 frames per second. This was the second synthesis. Finally, using the same eight instances of the designed FME interpolation architectures, but reducing the operation frequency to 650MHz, it is possible to process UHD 4K videos at 30 frames per second.

An important observation is the relationship between the number of instances and frequency. Since the UHD 4K resolution is four times larger than FHD, to achieve 4K processing it is required four times the number of instances of the AFI-AM architecture, considering the same frequency of 1.3GHz. Since the objective of this work is to keep the frequency, area, and power similar to related works found in the literature, it is possible to reduce the operating frequency by half just by doubling the number of AFI-AM instances. This way, it is possible to use a lower frequency, while keeping the area similar to the one presented in the related works.

The syntheses’ results are presented in Table IV. The first important conclusion when observing the results in Table IV is that the different levels of imprecision explored in this work caused an impressive impact in terms of area and power, independently of the explored parallelism level of operation frequency. The area gains vary from 72.0% to 83.2%, and the power gains vary from 79.0% to 83.4% depending on the explored approximation. At this point, it is important to emphasize that the power results are not considering the expressive I/O reduction reached by the AFI-AM6 and AFI-AM4 architectural versions.

Another interesting conclusion is that, even with the same encoding efficiency results, the AFI-AM6 architecture can reach a power reduction of up to 20.8% if compared with the AFI-AM8 architecture and an area reduction of up to 14.1%. AFI-AM6 architecture also presented the lowest power dissipation among all explored solutions.

Table IV also shows that increasing the number of architecture instances leads to a proportional increase in throughput, but with important impacts on power and used area. When halving the operation frequency, the area and power were reduced in all cases, but not in the same proportion.

Table V shows the comparisons with related works. To simplify, only the synthesis results of the three architectural versions designed in this work using eight instances (8×AFI-AM8, 8×AFI-AM6, and 8×AFI-AM4) and targeting 650MHz are presented in Table V. These versions were selected since these solutions are able to process UHD 4K videos (as in most of the related works) with the best power results among the solutions explored in this work. Table V also shows the results reached by the related works.

As explained in Section IV, the related works [16], [21], [22], [17], [23], [18], and [24] cannot be fairly compared with our architectures, since the number of filters supported, the complexity of these filters and the used technology do not match with the defined in our work.

The work in [16], presents a precise architecture for the AV1 MC tools, then half of the filters presented in [16] are the same presented in this work. However, since the work in [16] is focused on MC, then only one filtering must be done for each input block, according to the FME decision. The AV1 FME, on the other side, must evaluate all filters, pushing up the processing requirements, which naturally will require more area and more power. Even with this important difference, the use of approximation computing allows our AFI-AM4 version to reach a power dissipation only 2.4 higher than the work in [16], using almost the same number of gates.

The works proposed in [21] and [22] show precise architectures for the AV1 FME and MC interpolation, targeting the Sharp with 8-taps and Regular 6-taps filters, respectively. Since only one interpolation filter family is supported, these works do not implement a complete MC or FME. Observing both architectures in [21] and [22], the throughput of these works is the same as our work and the area is higher than 8×AFI-AM6 and 8×AFI-AM4 versions, even focusing only on one filter family. On the other hand, power results are smaller than the ones reached by all versions designed in this work. However, it is important to highlight that our architectures support more filters and implement the complete interpolation defined in AV1 FME.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Freq. (GHz)</th>
<th>Gates (K)</th>
<th>Power (mW)</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>1×Precise</td>
<td>1.3</td>
<td>82.98</td>
<td>76.63</td>
<td></td>
</tr>
<tr>
<td>1×AFI-AM8</td>
<td>1.3</td>
<td>21.02</td>
<td>14.05</td>
<td>FHD</td>
</tr>
<tr>
<td>1×AFI-AM6</td>
<td>1.3</td>
<td>19.03</td>
<td>13.31</td>
<td>30fps</td>
</tr>
<tr>
<td>1×AFI-AM4</td>
<td>1.3</td>
<td>14.20</td>
<td>13.30</td>
<td></td>
</tr>
<tr>
<td>8×Precise</td>
<td>1.3</td>
<td>539.42</td>
<td>781.62</td>
<td></td>
</tr>
<tr>
<td>8×AFI-AM8</td>
<td>1.3</td>
<td>136.66</td>
<td>140.65</td>
<td>4K</td>
</tr>
<tr>
<td>8×AFI-AM6</td>
<td>1.3</td>
<td>129.42</td>
<td>129.10</td>
<td>60fps</td>
</tr>
<tr>
<td>8×AFI-AM4</td>
<td>1.3</td>
<td>90.91</td>
<td>144.43</td>
<td></td>
</tr>
<tr>
<td>8×AFI-AM8</td>
<td>0.65</td>
<td>427.44</td>
<td>499.84</td>
<td></td>
</tr>
<tr>
<td>8×AFI-AM6</td>
<td>0.65</td>
<td>119.49</td>
<td>104.72</td>
<td>4K</td>
</tr>
<tr>
<td>8×AFI-AM4</td>
<td>0.65</td>
<td>102.62</td>
<td>82.91</td>
<td>30fps</td>
</tr>
</tbody>
</table>
The work in [23] targets only the interpolation filters of the VP9-10 format, in a total of 60 filters supported. The synthesis results in [23], even focusing only on the interpolation filters, can be evaluated as the MC, since all filters are implemented. In this case, the evaluation of the different filter families required by the FME was not considered. Then, the area and power are lower than the ones reached by our solution, as expected. In that case, real video samples were used to estimate the switching activity, then a much lower power was observed since the default value of 20% is a pessimistic value for video applications. Also, the work in [23] targeted a lower frequency and used less area, also contributing to reducing the reached power.

The work in [17] focuses on the 28 VP9 interpolation filters and even targeted a smaller resolution, reached the worst coding efficiency, and used 11 times more hardware and almost seven times more power than our $8 \times AFI$-AM6 version.

The work in [18] focuses on the three interpolation filters of the HEVC and even focuses on a much simpler problem, the compression efficiency losses are almost the same as reached by $8 \times AFI$-AM6 version, but with higher throughput and lower power and area, as expected, since only three filters are supported.

Finally, work [24] focuses on 15 VVC interpolation filters. Once more, since the number of filters is smaller, the reached area and power are also smaller than the one reached by our work. In this case, it is important to highlight the power results, since the work in [24] reached a similar power result to our $8 \times AFI$-AM6 version, even with fewer filters.

### Table V. Synthesis Results and Comparison with Related Works

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Related Works</th>
<th>This Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>[16] [21] [22] [17] [23] [18] [24]</td>
<td>$8 \times AFI$-AM8 $8 \times AFI$-AM6 $8 \times AFI$-AM4</td>
</tr>
<tr>
<td>Freq. (MHz)</td>
<td>256.6 441.0 448.4 248.9 413.9 466.5 357.0</td>
<td>650.0 650.0 650.0</td>
</tr>
<tr>
<td>Gates (K)</td>
<td>40.7 104.3 106.1 1,171.0 71.2 87.9 11.7</td>
<td>119.49 102.62 71.59</td>
</tr>
<tr>
<td>Power (mW)</td>
<td>16.8 63.1 56.3 569.9 2.34 32.3 77.0</td>
<td>104.72 82.91 93.5</td>
</tr>
<tr>
<td>BD-BR (%)</td>
<td>- - - 2.94 - 0.52 -</td>
<td>0.54 0.54 1.25</td>
</tr>
<tr>
<td>Technology</td>
<td>TSMC 40nm STMicro 65nm STMicro 65nm Nangate 45nm Nangate 45nm Nangate 45nm TSMC 90nm</td>
<td>TSMC 40nm</td>
</tr>
<tr>
<td>Throughput</td>
<td>4K 30fps 4K 30fps 4K 30fps 4K 30fps</td>
<td>8K 30fps 8K 30fps 4K 95fps 4K 30fps</td>
</tr>
<tr>
<td>Filters</td>
<td>90 8-taps 30fps 6-taps 28 60 3 15 42</td>
<td></td>
</tr>
<tr>
<td>Tools</td>
<td>MC FME MC FME MC FME MC FME</td>
<td></td>
</tr>
<tr>
<td>Standard</td>
<td>AV1 AV1 AV1 VP9 VP9-10 HEVC VVC AV1</td>
<td></td>
</tr>
</tbody>
</table>


tions using a fully combinational and multiplier-less implementation. Since the number of filter taps was reduced in two architectural versions, then, the required I/O bandwidth was also reduced by up to 59.5%, in comparison with a precise solution. The inserted imprecision caused a small average coding efficiency degradation of 0.54% in BD-BR for the AM8 and AM6 solutions and of 1.25% for the AM4 solution, when compared to the original AV1 filters.

Finally, three versions of complete AV1 FME interpolations architectures were also presented in this paper, one for each approximated level: AFI-AM8, AFI-AM6, and AFI-AM4. All three architectures considered the same architectural template and were designed to process $4 \times 4$ blocks. As many instances as necessary of these architectures can be used to reach the desired throughput, according to the target application. The designed architectures, when compared with a precise version, presented area gains varying from 72.0% to 83.2% and power gains varying from 79.0% to 83.4%, depending on the approximation level.

### VII. Conclusion

This paper presented three optimized approximate architectures for the AV1 FME interpolation filters. Different levels of approximation were explored, and three solutions were presented: AM8, AM6, and AM4. The explored approximations changed the filter coefficients to hardware-friendly values and reduced the number of taps in the interpolation filters.

Three multilayer architectures were designed, one for each approximated solution: MF-AM8, MF-AM6, and MF-AM4. These architectures can process all 48 AV1 interpolation options using a fully combinational and multiplier-less implementation. Since the number of filter taps was reduced in two architectural versions, then, the required I/O bandwidth was also reduced by up to 59.5%, in comparison with a precise solution. The inserted imprecision caused a small average coding efficiency degradation of 0.54% in BD-BR for the AM8 and AM6 solutions and of 1.25% for the AM4 solution, when compared to the original AV1 filters.

Finally, three versions of complete AV1 FME interpolations architectures were also presented in this paper, one for each approximated level: AFI-AM8, AFI-AM6, and AFI-AM4. All three architectures considered the same architectural template and were designed to process $4 \times 4$ blocks. As many instances as necessary of these architectures can be used to reach the desired throughput, according to the target application. The designed architectures, when compared with a precise version, presented area gains varying from 72.0% to 83.2% and power gains varying from 79.0% to 83.4%, depending on the approximation level.

### REFERENCES


