High-Throughput Hardware Design for the AV1 Decoder
Switchable Loop Restoration Filters

Roberta Palau\textsuperscript{1,2}, Wagner Penny\textsuperscript{1,2}, Ramiro Viana\textsuperscript{2}, Jones Goebel\textsuperscript{2}, Marcelo Porto \textsuperscript{2},
Guilherme Corrêa\textsuperscript{3} and Luciano Agostini\textsuperscript{2}

\textsuperscript{1}Sul-rio-grandense Federal Institute - Pelotas, Brazil
\textsuperscript{2}Video Technology Research Group (ViTech), Graduate Program in Computing (PPGC), Federal University of Pelotas - UFPEL, Brazil
e-mail: robertapalau@ifsul.edu.br

Abstract—This paper presents a high-throughput hardware design for the Switchable Loop Restoration Filter (SLRF) of the AV1 video format. This hardware includes the two filters defined at the AV1 SLRF: the Separable Symmetric Normalized Wiener Filter (SSNWF) and the Dual Self-Guided Filter (DSGF). The SLRF is the last step in the AV1 loop restoration filters, and it is used to attenuate blurring artifacts, improving the subjective video quality and the coding efficiency. The designed hardware targeted the AV1 decoder and is able to process up to 4K Ultra-High Definition (4K UHD) videos (with 3840x2160 pixels) at 60 frames per second (fps) in real-time. In order to cover different scenarios, two other target throughputs were also evaluated: 4K UHD at 30fps and Full HD (FHD) (with 1920x1080 pixels) at 30fps. The architectures were synthesized for standard cells using the 40 nm TSMC library. The SSNWF and DSGF architectures used 37.38 Kgates and 177.58 kgates in all evaluated scenarios. Depending on the evaluated scenario, the SSNWF power dissipation varied from 8.25 mW to 26.95 mW and the DSGF power varied from 57.19 mW to 115.02 mW. This is the first paper in the literature presenting a hardware design for the AV1 SLRF with its two filters.

Index Terms—AV1; Switchable Loop Restoration Filter; Separable Symmetric Normalized Wiener Filter; Dual Self-Guided Filter; Hardware Design.

I. INTRODUCTION

With the recent technological innovations that led to the popularization of consumer electronic systems for data interchanging, the use of video resources has become common in our daily lives. Digital videos are currently used for entertainment, work, and education, among other applications. These technologies became even more essential after the COVID-19 pandemic, allowing life to continue even at a distance with social isolation.

Considering the increasing demand for high video resolutions and frame rates, in recent years many encoders have been developed, such as the High-Efficiency Video Coding (HEVC) [1], the Google VP9 [2], the AOMedia Video 1 (AV1) [3] and the Versatile Video Coding (VVC) [4]. HEVC is a video standard released in 2013 that has doubled compression rates while maintaining the same video quality when compared to its predecessor, the H.264/AVC (Advanced Video Coding) [5]. On the other hand, VP9 is an encoder released by Google in 2013, focused on a solution designed to be royalty-free, also doubling the compression rate when compared to its previous version, the VP8 [6]. AV1 and VVC are the current state-of-the-art encoders, released in 2018 and 2020, respectively. VVC is the successor of HEVC, also developed by experts from ITU-T and ISO.

Regarding the mentioned encoders, the main focus of this paper is the AV1 codec. AV1 is a video format based on Google VP9 [2], and it has incorporated technologies from other encoders such as Thor from Cisco [7] and Daala from Mozilla [8], besides the technologies already introduced by the Alliance for Open Media (AOM). It is a video format specified by the AOM and developed by a consortium that includes the biggest technology companies in the area, such as the previously mentioned Google, Cisco, and Mozilla, as well as other giants like Microsoft, Netflix, Nvidia, Samsung Electronics, Facebook, and Intel, who came together to build an efficient, open-source and royalty-free codec [3].

When defining AV1, the AOM’s purpose was to create an encoder for web-based video applications, such as video-on-demand services, live streaming, conferencing, and virtual reality. It also has, among other functions: the consistent delivery of high-quality, real-time video; scalability for electronic devices to multiple bandwidths; viability, and hardware optimization. Moreover, AV1 can achieve more than 30% reduction in terms of bitrate when compared to VP9, when considering the same video quality [9].

The AV1 encoding process is based on a hybrid encoder model, the widely employed scheme used by previous encoders, composed of these eight stages: intra-frame and inter-frame predictions, transforms, quantization, entropy encoding, inverse quantization, inverse transform, and the In-loop filters [9].

As well as other previous codecs, AV1 presents an In-loop Filtering stage, which is responsible for reducing artifacts caused by the video compression process. The filtering improves the subjective quality of the final image and increases the video coding efficiency [10]. The AV1 In-loop Filtering is composed of three independent filters, used sequentially: (i) the Deblocking Filter (DBF), (ii) the Constrained Directional Enhancement Filter (CDEF), and (iii) the Switchable Loop Restoration Filter (SLRF). The SLRF is the main focus of this work and aims to attenuate the blurring effects. It comprises two independent filters: (i) the Separable Symmetric Normalized Wiener Filter (SSNWF), and (ii) the Dual Self-Guided Filter (DSGF).

The encoders have the common goal of achieving high coding efficiency rates while maintaining a trade-off between video quality and bitrate, but normally leading to a significant computational effort. Then, as higher is the coding efficiency of an encoder, as higher tends to be the required comp-
putational effort to encode a video. To deal with this challenge, new solutions are needed to enable the use of these encoders and decoders in real scenarios. The use of Graphics Processing Units (GPUs), Image Signal Processors (ISPs), and/or dedicated hardware, are some examples of these solutions [11]. However, when focusing on battery-powered devices and embedded systems, the use of dedicated hardware is mandatory due to power constraints. In any case, encoding and decoding a video requires much more than an efficient program running on a powerful General-Purpose Processor (GPP).

The AV1 In-loop Filter, which is the focus of this work, presents its particularities and challenges for the development of dedicated hardware. These filters are mandatory elements for the current encoders and decoders.

The H.264/AVC [5] In-loop filtering at the decoder side is represented by the DBF, which is considered a complex stage since it consumes 30% of the decoder computational effort (entropy decoding with 19%, inverse quantization with 20%, predictions with 9% and others with 22%) [12]. In HEVC, only the DBF (which is not the only In-loop filter in this standard) consumes 20% of the decoder computational effort [13]. In VVC, In-loop filtering is the least complex step on the encoder side, but it is the most complex step on the decoder side. This is explained by the fact that the loop filters algorithms apply almost the same operations at the encoder and decoder sides, while other tools, like predictions and transforms, require a much higher computational cost at the encoder than at the decoder [14]. No works were found in the literature about the AV1 In-Loop filtering computational effort at the decoder side. For that reason, a quick experiment was carried out considering 10 frames of five 4K UHD video sequences (Netflix Boxing Practice, Netflix Dancers, Netflix Ritual Dance, Netflix Toddler Fountain, and Netflix Bar Scene), available in the IETF-NETVC-testing [15] recommendations, using the four recommended values of constant quantization (CQ): 20, 32, 43 and 55. The experiments were run in a server with an Intel Xeon E5-4650 v3 2.10 GHz processor and 512 GB of DDR4 memory with SIMD optimizations active. The results indicate that the In-Loop filters together require up to 29% of the decoder computational effort. The In-Loop Filters are more complex on the decoder side because, when compared to other tools, the filtering operations are the same on the encoder and decoder side. As an example, the inter-frames prediction stage is much more complex on the encoder side because it must evaluate all possibilities of block sizes and coding modes in terms of bit-rate and distortion (quality) to define which combination is the best decision to encode each video frame region. This means that dozens of combinations must be evaluated to select only one. The decoder must only decode the single decision defined by the encoder. This also occurs at the intra-frame prediction and transforms stages, for example. The high computational cost of the decoder In-Loop filters is one of the main motivations for developing optimized architectures for these filters.

The implementation of dedicated hardware is necessary considering all AV1 stages, to enable the wide use of this video format in current and future video applications. But even with this relevance, there are still a few published works for AV1 hardware implementations. For the intra-frame prediction step, there are only six works:[16], [17], [18], [19], [20] and [21]. For the inter-frame prediction step, four works were identified: [22], [23], [24] and [25]. For the entropy encoding step, two works [26] and [27] were found. For the transform and quantization steps, no published works were found. Six works have been published focusing on the In-loop filtering stage, and five of them were developed in our previous work: [28] and [29] are solutions targeting the CDEF, [30] target the DBF filters, and [31] and [32] target the SLRF filters, respectively for the SSNWF and the DSGF. The sixth work presents a hardware design for the SSNWF targeting the encoder side [33].

This work presents a hardware design targeting the AV1 Switchable Loop Restoration Filter (SLRF), including its two main filters: the Separable Symmetric Normalized Wiener Filter (SSNWF) and the Dual Self-Guided Filter (DSGF). These designs are focused on the decoder side. Since most of the applications able to handle digital videos tend to run on battery-powered devices, like smartphones, the hardware designs targeting the decoder are even more relevant, since video reproduction is much more used than a video recording.

The literature presents only two related works targeting the AV1 SLRF at the decoder side ([31] and [32]), which are previous works from the authors. The main contributions presented in this paper are:

- This is the first paper in the literature presenting detailed information about the AV1 SLRF filters operation processes, including equations and examples of operation.
- This is the first paper in the literature presenting hardware solutions for the two filters defined at the SLRF filtering stage of the AV1 decoder: the SSNWF and the DSGF.
- This paper also presents novel synthesis results for three different scenarios, with different throughput restrictions, allowing an evaluation of the filters’ behavior when running at different operating frequencies.

The rest of this paper is organized as follows. The next section briefly presents the AV1 In-Loop filters. Section III presents, in detail, the AV1 Switchable Loop Restoration Filter (SLRF), including its two filters: the Separable Symmetric Normalized Wiener Filter (SSNWF) and the Dual Self-Guided Filter (DSGF). Section IV presents the designed architectures for the SSNWF and DSGF filters. Section V presents the synthesis results and conclusions are presented in Section VI.

## II. THE AV1 IN-LOOP FILTERS

The In-Loop Filtering is a stage within the encoder and decoder responsible for reducing artifacts caused by the video coding process. The filtering improves the subjective video quality and enhances the video coding efficiency. The AV1 In-Loop Filtering is composed of three independent filters presented in Fig. 1: (i) the Deblocking Filter (DBF), (ii) the Constrained Directional Enhancement Filter (CDEF), and (iii) the Switchable Loop Restoration Filter (SLRF). These

---

(iii) the Switchable Loop Restoration Filter (SLRF). These

---

(iii) the Switchable Loop Restoration Filter (SLRF). These
filters are sequentially applied. The SLRF comprises two independent filters: (i) the Dual Self-guided Filter (DSGF), and (ii) the Separable Symmetric Normalized Wiener Filter (SSNWF), which are the focus of this work.

Almost all AV1 encoding process considers a frame partition, where the basic unit is called a superblock. A superblock can have sizes of 128x128 or 64x64 pixels and each superblock can be recursively divided into smaller blocks with different shapes and sizes, where the smallest allowed block size is 4x4 pixels. The prediction, transforms, quantization, entropy coding, and the DBF and CDEF In-loop filters are all applied over these superblock partitions. Nevertheless, the SLRF is an exception. This filter is applied over a specialized structure called Loop Restoration Unit (LRU). Each LRU can have sizes of 64x64, 128x128, or 256x256 pixels and includes a variety of encoded blocks and even more than one superblock [34].

As previously discussed, the SLRF filters aim to attenuate the blurring effects. These effects are caused by the coding process, specifically from linear filters used to reduce the blocking effects [34] and the ringing effects [8]. The blurring artifact presents a reduction of the sharpening object edges and a loss of spatial details, especially in regions with high spatial activity [35].

III. SWITCHABLE LOOP RESTORATION FILTER (SLRF)

The Switchable Loop Restoration Filter (SLRF) is the last filter in the loop restoration of AV1, and it is applied over the results of the CDEF filter. This filter is the main AV1 novelty in the In-loop filters when compared to the previous codecs. This new tool presents more than 2% bit-rate savings when enabled [36]. Also, as previously discussed, SLRF comprises two independent filters: the DSGF and SSNWF.

During the SLRF stage, the frame is divided into LRUs. The SLRF acts directly in the LRU, which is evaluated, and, based on that evaluation, there is a decision about the filtering operation and the definition of which filter will be applied: the DSGF or the SSNWF. As the SLRF name suggests, there is a possibility of switching one filter to another filter at the same frame, i.e., both filters can be applied at the same frame, but over different LRUs. The frame is flagged by four parameters: (i) restore none, (ii) restore wiener, (iii) restore Sgrproj, and (iv) restore switchable. These parameters indicate the need for the filtering operation at frame or LRU level [37].

The \textit{restore none} parameter is used at both levels (frame and LRU) and indicates that no filtering was applied at that frame or LRU. The \textit{restore wiener} parameter at frame level signals that the LRUs in that frame were filtered using the SSNWF or were not filtered. In this case, each LRU is flagged to indicate if the filter was used or not, using the \textit{restore wiener or restore none parameters}, respectively. The \textit{restore Sgrproj} parameter signals that the LRUs in a frame were filtered using the DSGF or were not filtered. Again, the LRUs signalize if they were filtered or not, through the \textit{restore Sgrproj or restore none parameters}. Finally, \textit{restore switchable} indicates that the frame was filtered by both filters, SSNWF and DSGF, but over different LRUs. Then, one of the three LRU parameters (\textit{restore wiener, restore Sgrproj or restore none}) must be flagged for each LRU [37].

Figs. 2 and 3 present two frames from the sequence “MINECRAFT 60f 420” [15], encoded using a constant quantization (CQ) \textit{CQ = 63}. Both figures were generated using the Analyzer AOM tool [38]. Lines in red in Fig. 2 and Fig. 3 depict the LRU partitions and lines in light blue depict the frame division into 64x64 superblocks.

Fig. 2 was encoded enabling both filters and the frame parameter \textit{restore switchable} was flagged. Fig. 3, on the other side, was encoded using only the SSNWF, then the \textit{restore wiener} parameter is signalized at the frame level. In both figures, LRUs in purple were filtered by the DSGF (and flagged with \textit{restore Sgrproj} parameter), LRUs in green were filtered with the SSNWF (and flagged with \textit{restore wiener} parameter), and the other LRUs were not filtered (and flagged with \textit{restore none} parameter). As one can notice, since the \textit{restore wiener} parameter is enabled at the frame presented in Fig. 3, only LRUs filtered with SSNWF or without filtering are presented.

Another important observation about Fig. 2 and Fig. 3 is that the frames have LRUs with different sizes. As mentioned before, the LRU selectable sizes are 64x64, 128x128, and 256x256. The size 256x256 (with 4x4 superblocks of 64x64) was selected to generate Fig. 2 and Fig. 3. As one can notice in the figures, there are different sizes of LRU at the bottom of the frame (LRUs with 4x5 superblocks) and on the right side (LRUs with 2x5 superblocks). This difference is allowed when the frame size is not a multiple of 256.

The decision about the use of SLRF and which filter will be applied (DSGF or SSNWF) is defined in the encoding process. The decision between the two filters is based on the sum of squared error (SSE) when comparing the original LRU and the filtered one. Thus, both filters are applied in the evaluated LRU to define if the SLRF is advantageous and, if so, the filter with the best result for each LRU is selected. Those encoder decisions are signalized in the bitstream to the decoder [39]. On the decoder side, all the filtering operations are repeated based on the decisions of the encoder side.

Before starting the filtering operation, the SLRF uses some definitions to handle frame border regions, where the neighbor samples are not available. In this case, the frame is padded three samples up and three samples to the left [39], copying the original samples at the frame border.

The SSNWF and DSGF characterize a restore process because even the LRU presenting large sizes, the filters operate...
sample by sample inside the LRU. The SSNWF, when chosen, modifies the LRU samples through a 7-tap Finite Impulse Response (FIR) filter for luminance components and a 5-tap FIR filter for chrominance components [39]. The DSGF applies, for each LRU, two self-guided filters supporting a sliding window sized as 3x3 or 5x5 around a central sample. The output of the two self-guided filters is combined with two multipliers \( \alpha \) and \( \beta \), which smooth the difference between the reconstructed LRU and the original one, contributing to a final restored LRU [36]. Since these filters are the focus of this paper, they are detailed in the next sections.

A. Separable Symmetric Normalized Wiener Filter (SSNWF)

The AV1 Separable Symmetric Normalized Wiener Filter (SSNWF) is a 7-tap FIR filter for luminance components and a 5-tap FIR filter for chrominance components. It acts separately to the vertical and horizontal directions, focusing on reducing the process complexity. Horizontal and Vertical filters are symmetric. The main idea of these filters is that all degraded samples from the frame could be reconstructed from the samples that surround it [36]. The filter taps are calculated in the coding process and signalized into the bit-stream to the decoding process.

The filtering operation starts from the horizontal direction and then goes to the vertical direction, using specific taps for each direction [39].

The SSNWF filtering operation on the horizontal direction is presented in (1) to (4). Equation (1) presents the first filtering operation for both filters (7- and 5-taps) where \( X_n \) are the filter taps and \( p_n \) are the image samples. When the 5-taps filter is used, then \( X_0 \) and \( X_6 \) are zeroed. Each tap is a two-complement value with seven bits and then the SSNWF filters could have taps ranging from (-64 to +63) [39]. The input samples are 8-bits wide. In (1), the central sample \( p_3 \) is the one that is being filtered and the \( h \) in the \( p'_3 \) is to indicate that this is a partial result from horizontal filtering.

\[
p'_3 = X_0 \times p_0 + X_1 \times p_1 + X_2 \times p_2 + X_3 \times p_3 + \]
\[
X_4 \times p_4 + X_5 \times p_5 + X_6 \times p_6 \tag{1}
\]

After the filtering operation, equations (2) to (4) are applied to round and clip the result. The final value will be delimited between 0 and 8192. When this process is done for all samples of a LRU, then the vertical filtering process can start.

\[
p''_3 = p'_3 + (p_3 \times 128) + 16384 \tag{2}
\]
\[
p''_3 = (p''_3 + 4)/8 \tag{3}
\]
\[
p''_{3,final} = clipping(p''_3, 0, 8192) \tag{4}
\]

Equation (5) presents the first vertical filtering operation, and it is very similar to (1). The differences are in the used taps \( X_n \) to vertical directions and in the input samples, that now are the horizontally filtered samples \( p''_3 \) and are 14-bits wide. In the sequence, equations (6) to (8) are applied to round and clip the result, which is represented between 0 and 255 (8-bits per sample) as the original range of the input samples.

\[
p''_3 = X_0 \times p''_0 + X_1 \times p''_1 + X_2 \times p''_2 + X_3 \times p''_3 + \]
\[
X_4 \times p''_4 + X_5 \times p''_5 + X_6 \times p''_6 \tag{5}
\]
\[
p''_3 = p''_3 + p''_3 \times 128 - 262144 \tag{6}
\]
\[
p''_3 = (p''_3 + 1024)/2048 \tag{7}
\]
\[
p''_{3,final} = clipping(p''_3, 0, 255) \tag{8}
\]

The vertical filtering must be done for all horizontally filtered samples inside the LRU. This process is repeated for every sample in each frame LRU, until the frame is completely restored and ready to serve as reference to the next steps of the coding process.

B. Dual Self-Guided Filter (DSGF)

The Dual Self-Guided Filter (DSGF) is one of the two independent filters within the SLRF, as previously mentioned. The DSGF is based on two self-guided images [40] with distinct parameters used to compose a final restored image. These parameters are \( r \) and \( e \), which are defined during the encoding process. The parameter \( r \) can assume values 0, 1, 2 [39] and it is used to define the window size in the filtering operations as \((2r+1)x(2r+1)\). When \( r = 0 \), the filtering is disabled. The parameter \( e \) is used to update some values for the second filtering stage, as will be further discussed. The first guided image is defined by parameters \( r_1 \) and \( e_1 \), called Set 1 in this work. The second guided image is defined by parameters \( r_2 \) and \( e_2 \), called Set 2 in this work. The encoder also defines two weight factors \( \alpha \) and \( \beta \) used to improve the
The values of \( r_1, r_2, e_1, e_2, \alpha \) and \( \beta \) are signaled by the encoder in the bitstream [41].

The generation of the two self-guided images starts with a computation around a degraded sample, named as central sample (CS). In this first step, intermediary values \( a \) and \( b \) are calculated through nine sliding windows around the CS (including the CS), as showed in Fig. 4.

The \( a \) and \( b \) values calculations are defined in equations (9) and (10), respectively, where \( p_{i,j} \) represents each sample in each sliding window. If \( r = 1 \), then each sliding window will have 3x3 samples and if \( r = 2 \), each nine sliding windows will have 5x5 samples. This means that 25 samples around CS will be required when \( r = 1 \), and 49 samples around CS will be required when \( r = 2 \). In both cases, this process results in 3x3 matrices of \( a \) and \( b \) values, called \( A \) and \( B \). The operations are done based on \( r_1 \) for Set 1 and \( r_2 \) for Set 2. Finally, each CS of each guided image will have their own \( A \) and \( B \) matrices to be used in the next filtering steps.

\[
a = \sum_{i=0}^{2r} \sum_{j=0}^{2r} (p_{i,j})^2 \\
b = \sum_{i=0}^{2r} \sum_{j=0}^{2r} p_{i,j} \tag{9,10}
\]

Fig. 4 presents an example of \( a \) and \( b \) calculation when \( r = 2 \) (5x5 sliding windows in red). In this case, 49 samples (a 7x7 matrix) will be used in this process. Each 5x5 sliding window is used to calculate one \( a \) and one \( b \) values (\( a, b \) in blue in Fig. 4). At the end of these operations, the \( A \) and \( B \) matrices (highlighted in purple in Fig. 4) are ready to be used in the next filtering steps.

The second filtering stage is an update of the calculated \( a \) and \( b \) values based on the \((e_1, r_1)\) and \((e_2, r_2)\) parameters for Set 1 and Set 2, respectively. The encoder sets the values of these parameters [36] and the \( a \) and \( b \) adjustment is done to improve the filtering efficiency. This filtering operation stage has a high complexity, since it has a group of multiplications and divisions which manipulate values with a large bit width.

After the \( a \) and \( b \) operations around CS using the values of \((e_1, r_1)\) for the Set 1 and \((e_2, r_2)\) for Set 2, the updated values for \( a \) and \( b \) will be generated [39].

To update an \( a \) value, the results of \( b \) value are used for the same sample, besides a sequence of operations. To better understand these operations, it is considered that the value being updated is the \( a \) and \( b \) values of CS. However, these operations will be performed for all samples.

The update process of these values is defined from a sequence of operations that will be presented below, considering that some values vary according to the \( r \) value. For example, when \( r = 1 \) the window is 3x3 and has nine elements, then the variable \( j \) will be equal to nine. On the other hand, when \( r = 2 \), the window will be 5x5 and will have 25 elements, in this case, \( j \) will be equal to 25.

The next equations present the necessary operations to update the \( a \) value. To begin, equation (11) is presented.

\[
O = \begin{cases} 
\alpha \times j - b^2, & \alpha \times j - b^2 \geq 0 \\
0, & \alpha \times j - b^2 < 0
\end{cases} \tag{11}
\]

The equation (11) result is rounded off and multiplied by the \( e \) value for both sets, as defined in (12).

\[
R_a = [(O \times e) + 524288] / 1048576 \tag{12}
\]

Finally, the \( a \) value is updated from the equation (13), generating the new \( a \) value, named as \( a_n \).

\[
a_n = R_a \times 256 + (R_a / 2) / (R_a + 1) \tag{13}
\]

The \( a_n \) value will be in the range between 1 and 256 through a clip operation. With the result of an \( a \) updated value, the \( b \) value can also be updated through the operations defined in (14) and (15).

\[
R_b = (256 - a_n) \times b \times ((4096 + (j / 2)) / j) \tag{14}
\]

\[
b_n = (R_b + 2048) / 4096 \tag{15}
\]

The equation result in (15) is the value of \( b \) updated and, after updating all values of \( a \) and \( b \), the filters of Set 1 and Set 2 can start their operations, generating each of the self-guided samples.

Therefore, in the third stage, considering the updated \( A \) and \( B \) matrices, the filtering operation is performed according to equations (16) to (24) to generate \( v_1 \) and \( v_2 \) parameters that will be required for the final CS sample reconstruction, according to the first \((e_1)\) or the second \((e_2)\) guided image [39].

The \( v_1 \) definition, used to generate the first guided image (Set 1) sample, depends on the CS position inside the LRU. If CS belongs to an even row, the filtering is done through equations (16) and (17), but if CS belongs to an odd row, the filtering operation is defined by equations (18) and (19). In all cases, \( A_{i,j} \) identifies the value \( a \) in position \( i, j \) of matrix \( A \). The same occurs for matrix \( B \).

\[
m_{1e} = (A_{0,1} + A_{2,1}) \times 6 + (A_{0,0} + A_{0,2} + A_{2,0} + A_{2,2}) \times 5 \tag{16}
\]

\[
n_{1e} = (B_{0,1} + B_{2,1}) \times 6 + (B_{0,0} + B_{0,2} + B_{2,0} + B_{2,2}) \times 5 \tag{17}
\]
This section presents the designed hardware architecture for the AV1 Symmetric Normalized Wiener Filter (SSNWF). The high-level block diagram of this architecture is presented in Fig. 5 and it has two main modules (detailed in Fig. 5): the Horizontal Filtering, composed by the horizontal filter, and the Vertical Filtering, composed by four vertical filters. These modules are connected through two groups of registers: register barrier (RB) and four shift registers (SRs) – SR1 to SR4 in Fig. 5. All these modules will be detailed in the next paragraphs.

The architecture inputs are the seven horizontal and seven vertical taps, each one with seven bits, and ten LRU samples, with 8 bits per sample. Fig. 5 shows these inputs, including the ten LRU samples ($p_0$ to $p_9$), the seven taps used during the horizontal filtering ($tap_{0}h$ to $tap_{6}h$) and the seven taps used during the vertical filtering ($tap_{0}v$ to $tap_{6}v$). The outputs are four filtered samples with 8 bits each.

The SSNWF architecture was designed to process 64x64 LRUs, which is the worst-case scenario for this filter since this is the smallest input block size when considering SSNWF operation. Bigger LRUs are also supported without any change in the designed hardware. Processing smaller LRUs is considered the worst-case for hardware performance. This occurs because as smaller the size of the LRU as higher the number of LRUs inside a frame. Since the hardware must process all LRUs, the higher the number of LRUs inside a frame, as lower the hardware processing rates in terms of frames per second.

The designed architecture can process, in parallel, four neighbor input samples. Since the SSNWF, in the worst case, uses a 7-tap FIR filter (when processing luminance samples), to filter each central sample the SSNWF will require six other neighbor samples from each side of the central sample. Then the processing of four neighbor samples in parallel will require 10 input samples, as presented in the first line of Fig. 6. In this case, the four orange samples will be filtered, and additional gray samples will be required in the filtering process. This solution was designed to improve the reached throughput and to reduce the I/O. The I/O is reduced since many samples can be reused instead of being read again from the memory system. Only to exemplify, if the four samples were processed independently, then a total of 28 samples (7x4) would need to be read from the memory system. Our solution makes the same process reading only 10 samples. Fig. 6 presents, in gray, the samples used to filter each orange sample.

Furthermore, the architecture was designed using two stages of the pipeline to increase the reached throughput. One stage is used for each filter module. Registers SR1 to SR4 and RB work also as pipeline registers.

The horizontal filtering operation is defined by equations from (1) to (4) and vertical filtering is defined by equations (5) to (8), as previously presented. These equations were designed in a fully combinational way using four instances to allow the processing of four samples in parallel and explore hardware reuse and the share of sub-expressions. The first group of equations (1) to (4) are implemented at the Horizontal Filtering.
Fig. 5 Block diagram of the AV1 SSNWF designed architecture presenting its main two modules: Horizontal Filtering and Vertical Filtering.

As previously presented, the 7-tap FIR filter is applied over luminance samples and the 5-tap FIR filter is applied over chrominance samples. The developed architecture processes one component at a time, and one 7-tap FIR filter was implemented to deal with both luminance and chrominance samples. In this case, depending on the sample being processed (luminance or chrominance), the developed filter adapts the number of taps accordingly.

The Horizontal Filtering and Vertical Filtering modules are connected through two groups of registers: (i) four shift registers (SR1 to SR4) and (ii) one register barrier (RB), as presented in Fig. 5. One shift register is used for each output of the Horizontal Filtering module. Since to perform the vertical filtering, at least seven samples must be already filtered by the horizontal filter, thus, each one of these four shift registers will have seven positions and each position will have 14 bits, which is the bit width of the Horizontal Filtering module output. When the SRs are filled, then all seven horizontally filtered samples of each SR are delivered in parallel to the Vertical Filtering module, in a total of 28 filtered samples. RB is used to synchronize the architecture and to allow pipeline operation. These seven registers are used to store the seven vertical filter taps from the architecture inputs since these taps are used only after the horizontal filter operation. Each register has 7 bits (bit width of the filter taps).

The architecture has a control unit, designed as a finite state machine (FSM) with nine states, which synchronizes the modules and the registers. This FSM controls the activation of the register barrier and the shift registers. The control of these registers is of utmost importance to allow the required synchronism.

The architecture latency is nine clock cycles. One for the Horizontal Filtering process, seven to fill the shift registers, and one for the Vertical Filtering process. After this initial latency, four filtered samples are delivered at each cycle. These four samples are the restored samples that will compose the LRU and, later, the reconstructed frame.

**B. Dual Self-Guided Filter Architecture**

The proposed DSGF decoder architecture was designed to operate with the smallest LRU size (64x64), i.e. the worst-case scenario for the SLRF. The architecture was designed to deliver four processed CS samples in parallel. The inputs are the image samples and the 6-tuple \( r_1, e_1, r_2, e_2, \alpha, \beta \), which is generated by the encoder and signaled in the bitstream. Fig. 7 presents the DSGF architecture, which is composed of six main modules which were named: (i) Calc A B, (ii) UP A B Set 1, (iii) UP A B Set 2, (iv) FO Set 1, (v) FO Set 2 and (vi) Restore. These modules are fully combinational. The acronyms used to represent the main modules were used as a reference in Fig. 7. The Calc A B represents the calculations around A and B mentioned in equations (9) and (10), which results in A and B matrices presented in Session III.
Subsection B. The UP A B Set 1 and the UP A B Set 2 represent the values update of those A and B matrices including both sets as explained for the second filtering stage of the DSGF. The FO Set 1 and the FO Set 2 represent the Filtering Operation for Set 1 and Set 2 as explained for the third filtering stage and finally, the Restore module finalizes the filtering completing the restoration.

The DSGF architecture was designed to process, in parallel, four neighbor CS samples inside the LRU and to calculate one row of a and b values per cycle, since (refer to Fig. 4 for more details) each input row from the matrix is available at each clock cycle. This approach allows the sharing of common input samples and the common calculations among these four CS samples filtering, reducing the external memory communication, area, and power.

As discussed before, depending on the r value, 25 or 9 samples are necessary to calculate each a and b pair, in a total of 49 or 25 samples for the nine iterations. When four CS samples are independently processed, a total of 196 samples (49x4) must be read from the memory in the worst case (r = 2). Using our approach (with four samples filtered in parallel) the amount of input samples is reduced to 50, which implies a reduction of 3.92 times in the required memory bandwidth.

The DSGF architecture was designed as a pipeline with two macro-stages. The first macro-stage comprises the Calc A B, the UP A B Set 1 and the UP A B Set 2 modules in Fig. 7, while the second macro-stage comprises the FO Set 1, FO Set 2 and Restore modules in Fig. 7. Fig. 7 also presents a set of five Shift-Registers (SR A1, SR B1, SR A2, SR B2, SR CS) used to synchronize the pipeline and feed the second macro-stage. SR CS stores the four CS input samples that will be required in FO Set 1, FO Set 2 and Restore modules. SR A1, SR B1, SR A2 and SR B2 are used to store the values generated by the UP A B Set 1 and UP A B Set 2 modules that are required by the FO Set 1 and FO Set 2 modules.

The architecture was designed to be used together with an input buffer, as presented in Fig 7. This buffer receives 10 LRU samples per clock cycle. After five cycles, with a matrix with 10x5 input samples available, Calc A B module can start its calculations. This buffer was not included in the presented design.

The first three modules (from the first pipeline macro-stage) are the most complex ones because of the high number of multipliers and dividers required. Besides that, those modules are designed to process in parallel six a and b values for Set 1 and Set 2. In other words, they can continually deliver 24 outputs per clock cycle, while 12 outputs are related to Set 1 and the others are related to Set 2. As the Calc A B module was designed exploring common sub-expression sharing, the operations used for \( r = 1 \) are completely reused when processing \( r = 2 \) and, as much as possible, the multipliers from equation (1) were shared between CS1, CS2, CS3 and CS4 operations. These simplifications allowed savings of 94 multipliers and 96 adders in this module, but 110 multipliers and 288 adders are still required to reach the desired throughput. It is important to notice that the Calc A B module computes in parallel a and b for both window sizes, according to \( r_1 \) and \( r_2 \). Furthermore, the control (omitted in Fig. 7) selects the outputs according to the r value for both UP A B Set 1 and UP A B Set 2 modules, through Mux A and Mux B in Fig. 7.

The modules UP A B Set 1 and UP A B Set 2 are similar and update the a and b values according to \( e_1 \) and \( e_2 \), as previously presented. Each one of these modules uses five multipliers, four dividers and five adders for each CS sample. Then, 30 multipliers, 24 dividers and 24 adders are used in each module. If the four CS samples were processed in an independent way it would be necessary to use the double of operators to reach the same throughput. Besides the high complexity of the operations, the required bit widths are also high when compared to other modules where up to 41 bits are required in the internal calculations.

At each UP A B Set 2 and UP A B Set 2 modules it was possible to reduce 6 adders and 12 divisors, which results in 12 adders and 24 divisors less for each set, just avoiding unnecessary operations when compared to the way it was defined by the reference software [39]. In this case, some values were fixed, based on the operation \((4096 + (j/2))/j\) which is part of the equation (14). Where \( j \) is the window size and is defined by \( r \). Therefore, for an example, when \( r = 1 \), part of the equation used in (14) corresponding to \((4096 + (j/2))/j\), can be replaced by the value “455” when the window \( j = 9 \) and when \( r = 2 \) by the value “164” when the window \( j = 25 \). These adjustments enabled the reductions mentioned above.

Calc A B, UP A B Set 1 and UP A B Set 2 modules require, together, three clock cycles to generate the updated \( a \) and \( b \) values for Set 1 (12 values) and Set 2 (12 values) for the next macro-stage. These results are stored in SR A1, SR B1, SR A2 and SR B2. In this case, 18 values are stored in each SR, six per clock cycle. The processing of four CSs together allows a reduction in SR sizes from 36 values to 18 values. When the complete matrices are stored in the SR modules, the last three modules can be started.

FO Set 1 and FO Set 2 modules are fully parallel and utilize, in one cycle, all values stored in the SRs. These modules process all calculations defined in equations (3) to (11) for the four CS samples. FO Set 1 process the calculations defined in equations (3) to (8), delivering four \( v_1 \) values in parallel, one for each CS. FO Set 2 processes the calculations defined in equations (9) to (11) and delivers, also in parallel, the four \( v_2 \) values, one for each CS. FO Set 1 module was designed with 20 multipliers, four dividers and 48 adders and FO Set 2 module was designed with 20 multipliers, four dividers and 60 adders, to support the required parallelism.

Finally, the Restore module concludes the filtering operation using \( \alpha \), \( \beta \) and CS global inputs and \( v_1 \) and \( v_2 \) values calculated in the previous modules to generate the four filtered CS samples according to equations (12) to (14). This module uses eight multipliers, four dividers and 16 adders to reach the desired throughput.

The FO Set 1, FO Set 2 and Restore modules are fully combinational, so the four filtered samples are delivered in one clock cycle when the SRs are filled with valid samples.

As one can conclude from the previous discussions, the
DSGF architecture has a latency of nine clock cycles to generate the first four filtered CS samples: five cycles for the input buffer, three cycles for the first macro-pipeline stage and one cycle for the second macro-pipeline stage. After that, at each new cycle, four filtered samples are delivered at the output.

The processing of the LRU samples starts with all 64x64 luminance samples, followed by the two channels of chrominance samples. The latency of nine cycles must be paid when changing the color channel and when starting a new LRU filtering. The DSGF architecture has a control module (omitted in Fig. 7) that was designed as a FSM with nine stages, which is used to synchronize all modules operations.

V. SYNTHESIS RESULTS

The architectures were described in VHDL using the Quartus Prime 16.1 Lite Edition tool and validated using the ModelSim - Intel FPGA Starter Edition 2021.1 (Quar- tus Prime Pro) software [42], with real input data extracted from the AV1 reference software. It was synthesized in ASIC using the standard cell library TSMC [43] 40 nm at 0.9 V and 25°C. The syntheses were performed using the Cadence Genus Synthesis Solution tool and the power results were estimated using the default switching activity setting (20%) of the tool. The gate count was calculated based on a two-input NAND gate (0.9408 $\mu$m$^2$).

The minimum operating frequencies for the two designed architectures were defined based on equation (28), where $p$ is the number of pixels in a frame, $q$ is the frame rate (frames per second), $s$ is the color subsampling rate, $c$ is the number of cycles required to process an input block and $b$ is the number of samples in the input block.

$$f = \left( \frac{p \times q \times s \times c}{b} \right)$$

The worst case in terms of operating frequency is when the architecture processes 4K UHD frames ($p = 8,294,400$ pixels per frame, or $3840 \times 2160$) at 60 frames per second ($q = 60$ frames per second) and considering that all LRUs are filtered by the SSNWF or the DSGF. In all cases, a color subsampling of 4:2:0 was considered (thus, $s = 1.5$). The worst case in terms of input block for both architectures is the smallest LRU size, or 64x64 samples ($b = 4096$, or 64x64 samples). Finally, the value $c$ varies in the designed solutions.

The minimum frequency for the SSNWF hardware when considering the worst case was defined as 207.03 MHz since a 64x64 LRU is completely decoded in 1,136 cycles ($c$ is 1,136 cycles) by the SSNWF hardware. On the other hand, the minimum frequency for DSGF in the worst case was 212.86 MHz, since a 64x64 LRU is completely decoded in 1,168 cycles by the DSGF hardware.

For the other targeted resolutions and frame rates, other operation frequencies were defined. In these cases, $s$, $c$ and $b$ values remain the same, and $p$ and $q$ were accordingly updated.

Considering the minimum operation frequencies for all evaluated scenarios, the values were rounded up and the frequencies below were defined for the synthesis process of both filters: (i) FHD @ 30 fps: 30 MHz; (ii) 4K UHD @ 30 fps: 110 MHz; (iii) 4K UHD @ 60 fps: 215 MHz.

A. SSNWF Synthesis Results

Table I presents the synthesis results for the SSNWF architecture, also showing the results of its main modules: Horizontal Filtering (H-Filters in Table I) and Vertical Filtering (V-Filters in Table I).

The SSNWF architecture presented an area of 37.38 Kgates for the three evaluated operation frequencies, which means that the maximum evaluated operating frequency did not exceed the critical path defined by the synthesis tool in its default configuration. This also occurs for the SSNWF main modules, the Horizontal Filtering and Vertical Filtering with an area of 13.34 Kgates and 21.96 Kgates, respectively. The difference in area between these symmetric modules occurs as a function of the higher bit width of the samples processed by the Vertical Filtering module. Note that a simple sum of results from H-Filters and V-Filters does not lead to the global SSNWF results, since other structures are instantiated outside the Horizontal and Vertical Filtering, thus making SSNWF results higher than the sum of its main modules.

When considering power, the SSNWF hardware reached a total power dissipation varying from 8.24 mW (frequency of 39 MHz) to 26.95 mW (frequency of 215 MHz). Since the Vertical Filtering module uses a larger area than the Horizontal Filtering module, the Vertical Filtering module presented a higher power dissipation in all scenarios, with almost the
double of the power consumption than the Horizontal Filtering module.

Also considering the power results presented in Table I one can conclude that, as expected, as lower is the target operation frequency, as lower is also the power consumption.

There are only two works in the literature targeting a hardware design for the SSNWF AV1 filter.

The first work was presented in [33], but as previously discussed, they focused on the encoder side and focused on UMC 45nm technology. The synthesis results reported an area of 1,988,927.2 $\mu$m$^2$, when our solution used only 35,167.1 $\mu$m$^2$. The work in [33], process FHD (1920x1080) videos at 42.4 fps when running at 75.24 MHz and consuming 12.8 mW. For the same resolution, but with a little smaller frame rate of 30 fps, our SSNWF architecture must run at only 30 MHz, consuming 8.25 mW. Then, the work in [33], uses 56.5 times more area, requires 2.5 times higher frequency and consumes 1.6 times more power than our solution to process the same video resolution.

The second paper in the literature is our previous work [31], which was used as a basis for the architecture presented in Section IV.A.

B. DSGF Synthesis Results

Table II presents the synthesis results for the DSGF architecture, also showing the results of its main modules and considering the same three scenarios previously discussed.

The synthesis of the DSGF architecture and its main modules presented the same area for the three evaluated target frequencies, as in the SSNWF hardware and for the same reasons. The complete DSGF hardware used an area of 177.58 kgates. When looking for the synthesis results, one can notice that the modules UP A B Set 1 and UP A B Set 2 presented the larger area with 48.2 kgates each. The third module with the largest area is the Calc A B, presenting 35.8 kgates. These three modules together represent more than 77.6% of the total architecture area. This happens because a high number of operators (including multipliers and dividers) with large bit widths are required in each one of these modules.

The DSGF architecture presented a total power dissipation varying from 57.19 mW to 115.02 mW considering the minimum and maximum evaluated operation frequencies. The three modules with the higher area also presented the higher power dissipation, as expected. In this case, the modules UP A B Set 1, UP A B Set 2 and Calc A B, as expected, presented the most significant power dissipation for all targeted throughputs, representing more than 82% of the total DSGF hardware power for the three evaluated scenarios.

Again, as expected, as lower is the target operation frequency, as lower is also the power consumption.

The only work published in the literature targeting a hardware design for the DSGF AV1 filter is our previous work [32], which was also used as a basis for the architecture presented in section IV.B. Thus, no comparisons can be presented for the DSGF results.

VI. CONCLUSIONS

Hardware design in the video coding area is essential and gained relevance over the years specially for the decoder, since the decoding process is much more frequently employed than the encoding process. As AV1 is a promising and new video coding format, presenting efficient dedicated hardware solutions targeting its tools contributes to achieving a broader adoption of the format.

This paper presented a high-throughput dedicated hardware design for the AV1 Switchable Loop Restoration Filter (SLRF) considering its two filters: the Separable Symmetric Normalized Wiener Filter (SSNWF) and the Dual Self-Guided Filter (DSGF), both focusing on the AV1 decoder.

The SLRF architectures were evaluated over three video scenarios: FHD at 30 fps, 4K UHD at 30 fps and 4K UHD at 60 fps. For that, three different operation frequencies were defined as targets to the synthesis process: 30 MHz, 110 MHz and 215 MHz. The synthesis targeted the TSMC 40 nm library and the results showed that the DSGF hardware requires much more area and power than the SSNWF hardware. This is due to the complex calculations over high bit widths that are required by the DSGF hardware.

The results also showed that the architectures can be used in AV1 decoders in hardware targeting different video resolutions and supporting real time processing even for 4K UHD videos at 60 frames per second.

This is the first paper in the literature presenting hardware designs for the complete SLRF filters of the AV1 decoder, including the DSGF and SSNWF filters.

ACKNOWLEDGEMENTS

This work was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brazil (CAPES) – Finance Code 001. It was also supported by the CNPq and FAPERGS Brazilian agencies and by the Brazilian Microelectronics Society (SBMicro) through the Programa de Apoio a Projeto de Circuitos Integrados em Universidades (APCI).

REFERENCES


