Adaptive Loop Filter Hardware Design for 4K ASIC VVC Decoders

Ibrahim Farhat, Wassim Hamidouche, Adrien Grill, Daniel Menard, Olivier Deforges

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCE.2022.3146272, IEEE Transactions on Consumer Electronics

Adaptive Loop Filter Hardware Design for 4K ASIC VVC Decoders

Ibrahim Farhat, Wassim Hamidouche, Adrien Grill, Daniel Ménard, and Olivier Déforges

***Abstract*—Versatile video coding (VVC) is the next generation video coding standard released in July 2020. VVC introduces new coding tools enhancing the coding efficiency compared to its predecessor high efficiency video coding (HEVC). These new tools have a significant impact on the VVC software decoder with a complexity estimated to two times HEVC decoder complexity. In particular, the adaptive loop filter (ALF) introduced in VVC as an in-loop filter increases both the decoding complexity and memory usage. These concerns need to be carefully addressed regarding the design of an efficient hardware implementation of a VVC decoder. In this paper, we present an efficient hardware implementation of the ALF tool for VVC decoder. The proposed solution establishes a novel scanning order between Luma and Chroma components that reduces significantly the ALF memory. The design takes advantage of all ALF features and establishes an unified hardware module for all ALF filters. The design uses 26 regular multipliers in a pipelined architecture with a fixed throughput of 2 pixels/cycle and fixed system latency regardless of the selected filter. This design operates at 600 MHz frequency enabling to decode on ASIC platform a 4K video at 30 frames per second in 4:2:2 chroma sub-sampling format.**

***Index Terms*—VVC, In-loop Filter , ALF and ASIC.**

I. Introduction

THE versatile video coding (VVC) is the next generation video coding standard jointly developed by ISO/IEC Motion Picture Experts Group (MPEG) and ITU-T Video Coding Experts Group (VQEG) under the Joint Video Experts Team (JVET). The ITU-T H.266 | MPEG-I - Part 3 (ISO/IEC 23090-3) VVC [1, 2] standard introduces several new coding tools enabling up to 40% of coding gains beyond the high efficiency video coding (HEVC) standard for similar objective video quality [3, 4]. These coding tools have different complexity and quality enhancement trade-offs [2, 5, 6]. The hardware design of the VVC transform module including both multiple transform selection (MTS) and low frequency non-separable transform (LFNST) blocks has been well studied in the literature for field-programmable gate array (FPGA) [7–9] and application-specific integrated circuit (ASIC) [10–13] platforms. In these solutions, the authors mainly leverage the transform features such as butterfly decomposition of the discrete cosine transform (DCT) type II and the linear relationship between discrete sine transform (DST)-VII and DCT-VIII to reach a high throughput while minimizing hardware resources. One of the tools that significantly contributed to enhance the overall coding performance is the adaptive loop filter (ALF) [14]. The ALF was firstly considered as HEVC tool candidate but finally it was not included in the standard [15, 16]. The ALF was then standardized as one of the in-loop filters in VVC [17].

In-loop filters are located in the decoding loop of the encoder and aim to enhance the perceived quality of the decoded video sequence by eliminating the blocking, ringing or blurring artifacts generated by previous decoding stages. The VVC in-loop filter block relies on three filters: deblocking filter (DBF) [18], sample adaptive offset (SAO) [19] and ALF. The encoder estimates the optimal filter parameters that maximizes, the most, the objective quality of a block. These parameters are then transmitted to the decoder so that the in-loop filters of the decoder can use them to efficiently filter the reconstructed frame. The role of the three filters are defined as follows. The first one is the DBF whose aim is to remove the blocking artifacts that appear at the edges of the coding unit (CU). It will be mainly in charge of applying a smoothing filter to edges in order to remove blocking artifacts. A different type of smoothing filter can be applied to a block according to the properties of its neighboring blocks. The strength of the filter coefficients will be determined by the specific values of the edge pixels and thequantization parameter (QP). The SAO comes second with the objective to reduce the undesirable visible artifacts such as ringing. The SAO filter classifies pixels of a coding tree unit (CTU) into two different categories. The reconstructed samples in a specific category with a smaller value than the original ones, a positive offset is added to reduce the existing error. On the other hand, if samples in a specific category have higher values than their corresponding original samples, a negative offset will be applied. The final block of in-loop filters in VVC is the ALF. The main role of the ALF is to reduce visible artifacts such as ringing and blurring by reducing the mean absolute error between the original image and the reconstructed one. ALF is the main purpose of this paper. Fig. 1 illustrates the VVC in-loop filters carried out at the decoder. The in-loop block is fed by a reconstructed picture, which is then processed by the three filters in the order illustrated in Fig. 1. Finally, the ALF filter delivers the reconstructed and filtered picture to the decoded pictures buffer for temporal prediction and rendering at the decoder side. The ALF can also be applied at the post-processing stage of any decoded video to enhance its visual quality

Ibrahim Farhat, Wassim Hamidouche, Daniel Menard and Olivier Déforges are with Univ. Rennes, INSA Rennes, CNRS, IETR - UMR 6164, 20 Avenue des Buttes de Coesmes, 35708 Rennes, France. E-mails: Ibrahim.Farhat@insa-rennes.fr, wassim.hamidouche@insa-rennes.fr, Daniel.Menard@insa-rennes.fr, Olivier.Deforges@insa-rennes.fr.

Ibrahim Farhat and Adrien Grill are with VITEC, 99 rue Pierre Semard, 92320 Chatillon, France. E-mails: ibrahim.farhat@vitec.com, adrien.grill@vitec.com

Manuscript submitted on Septemer 2021.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCE.2022.3146272, IEEE Transactions on Consumer Electronics

before the display. The filter parameters are in this case sent as metadata or supplemental enhancement information (SEI) massage to enhance the overall video quality on consumer electronic devices such as TVs, smartphones, tablets, and VR headsets.

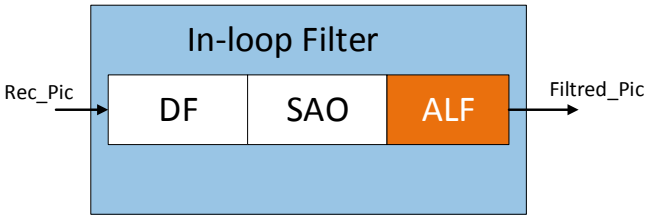


Fig. 1: VVC In loop filters

This paper addresses a hardware implementation of the ALF filter. Unlike previous hardware solutions, our design is the first to address both the memory and gradient stages of the ALF. In addition to ALF computational complexity, the most challenging part of its hardware implementation is related to the memory organization and the input scanning. In this paper, we propose an optimized scanning order between Luma and Chroma components that enables the most efficient memory management and usage. The design supports all ALF filters including 7×7 diamond shape filter (DSF) for Luma, 5×5 DSF and 3×4 cross-component adaptive loop filter (CCALF) diamond filter for the Chroma. These filters are implemented in a unified design that uses 26 regular multiplier (RM) to sustains a fixed throughput of 2 pixels/cycle and a fixed system latency in a pipelined architecture. The main contribution of this paper is to propose an optimized hardware design with minimal memory usage for ASIC decoder featuring many advantages:

- Optimized scanning order between Luma and Chroma components.
- Optimized memory structure and memory region usage.
- Area efficient design by taking advantage of all the symmetrical features of the ALF filters.
- Sustain a fixed throughput and a fixed system latency regardless of the ALF filter size.
- Use a fixed number of RMs in a unified and pipelined architecture.

The rest of this paper is organized as follows. Section II presents the background of the ALF followed by the existing ALF hardware implementations in Section III. The proposed hardware implementation of the ALF module is presented in Section IV. In Section V, the performance of the proposed hardware module is assessed in terms of hardware area. Finally, Section VI concludes the paper.

II. Background on VVC Adaptive Loop Filter

A. VVC adaptive loop filter

ALF uses Wiener filters for its core operation. Wiener filters are designed to minimize the mean squared error (MSE) between the original samples and the filtered samples of a signal. The original samples refer to the original frame from the video sequence and the filtered samples to the reconstructed picture after the ALF (ALF's output). The ALF is divided into several processing stages. The first one performs block partitioning, where the CTU is divided into multiple elementary macro blocks of size 4×4. These macro blocks are then filtered using the same set of coefficients. The filter's coefficients selection is detailed separately in the next section. ALF uses different filters for Luma and Chroma components. For the Luma component, it uses a 7×7 DSF. The filter coefficients of the 7×7 DSF can be fixed or signaled in the adaptation parameter set (APS). For the Chroma components, the ALF uses two filters. The first one is a 5×5 DSF filter and, unlike the 7×7 DSF, the filter coefficients can only be signaled in the APS. The second filter is the CCALF diamond filter. This latter uses the co-located Luma samples to filter the Chroma ones. Like the 5×5 DSF, the coefficients can only be signaled in the APS. Fig. 2 shows the ALF process as defined in VVC specification.

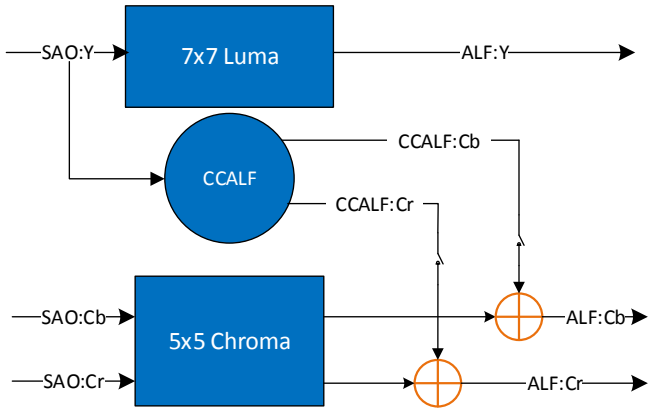


Fig. 2: ALF Luma and Chroma filters in the VVC decoder.

The remaining of this section will be divided in two parts. The first part addresses the Luma filter and its coefficients' derivation while the second part investigates the Chroma filtering.

B. Luma filter

1) Block classification: The ALF classification process relies mainly on the Luma component. First, the ALF divides the CTU into multiple macro blocks of size 4×4. Each macro block is categorized into one out of 25 classes. The classification index C is computed by Equation (1) based on the macro block directionality D and a quantized value of local pixels activity A.

$$C = 5\,D + \widetilde{A} \tag{1}$$

To calculate D and $\widetilde{A}$, the sum of horizontal g_h, vertical g_v and two diagonal g_{d1}, g_{d1} gradients are first computed using a 1-D Laplacian as follows:

$$g_h = \sum_{k=i-2}^{i+5} \sum_{l=j-minY}^{j+maxY} H_{k,l}, \qquad H_{k,l} = |R_{k-1,l} - 2\,R_{k,l} + R_{k+1,l}|\,. \tag{2}$$

$$g_v = \sum_{k=i-2}^{i+5} \sum_{l=j-minY}^{j+maxY} V_{k,l}, \qquad V_{k,l} = |R_{k,l-1} - 2\,R_{k,l} + R_{k,l+1}| \tag{3}$$

$$g_{d1} = \sum_{k=i-2}^{i+5} \sum_{l=j-minY}^{j+maxY} D1_{k,l}, \qquad D1_{k,l} = |R_{k-1,l-1} - 2\,R_{k,l} + R_{k+1,l+1}|\,. \tag{4}$$

$$g_{d2} = \sum_{k=i-2}^{i+5} \sum_{l=j-minY}^{j+maxY} D2_{k,l}, \qquad D2_{k,l} = |R_{k-1,l+1} - 2\,R_{k,l} + R_{k+1,l-1}|\,, \tag{5}$$

where indices (i, j) refer to the coordinates of the upper left sample within the 4×4 macro block and $R_{i,j}$ indicates a reconstructed sample at coordinate (i, j). $maxY$ and $minY$ are derived based on the macro block position relative to the virtual boundary. The maximum values that can take $minY$ and $maxY$ are 2 and 5, respectively.

To reduce the complexity of the block classification, the sub-sampled 1-D Laplacian calculation is applied. As shown in Fig. 3, the same sub-sampled positions are used for gradient calculation in all directions. This means that Equations (3), (2), (4) and (5) are only applied when both k and l are even numbers or both are odd numbers. This reduces the computation complexity by half. The blue pixels in Fig. 3 refer to the current 4×4 macro block to be filtered. All samples of this macro block share the same class value C.

After computing the sum of gradient in all directions, the maximum and minimum values of the macro block between the horizontal and vertical directions, and between the two diagonal directions, are computed by Equation (6)

$$\begin{aligned} g_{hv}^{max} &= max(g_h, g_v), \\ g_{hv}^{min} &= min(g_h, g_v), \\ g_{d}^{max} &= max(g_{d0}, g_{d1}), \\ g_{d}^{min} &= min(g_{d0}, g_{d1}). \end{aligned} \tag{6}$$

According to its direction, each 4×4 macro block will be assigned an integer value D in the interval $[0,\ 4]$. The value D will be then used to select the macro block class. To determine the value of the directionality D, the g_{hv}^{max} and g_{d}^{max} are compared against each other with two thresholds t_1 and t_2 as follows:

- **Step 1**: if both $g_{hv}^{max} \le t_1\, g_{hv}^{min}$ and $g_{d}^{max} \le t_2\, g_{d}^{min}$ are true, then D is set to 0.
- **Step 2**: if $g_{hv}^{max}/g_{hv}^{min} > g_{d}^{max}/g_{d}^{min}$ then continue from step 3, otherwise continue from step 4.
- **Step 3**: if $g_{hv}^{max} > t_2\, g_{hv}^{min}$, D is set to 2, otherwise D is set to 1.
- **Step 4**: if $g_{d}^{max} > t_2\, g_{d}^{min}$, D is set to 4, otherwise D is set to 3.

This process is summarized as a pseudo code in Algorithm 1.

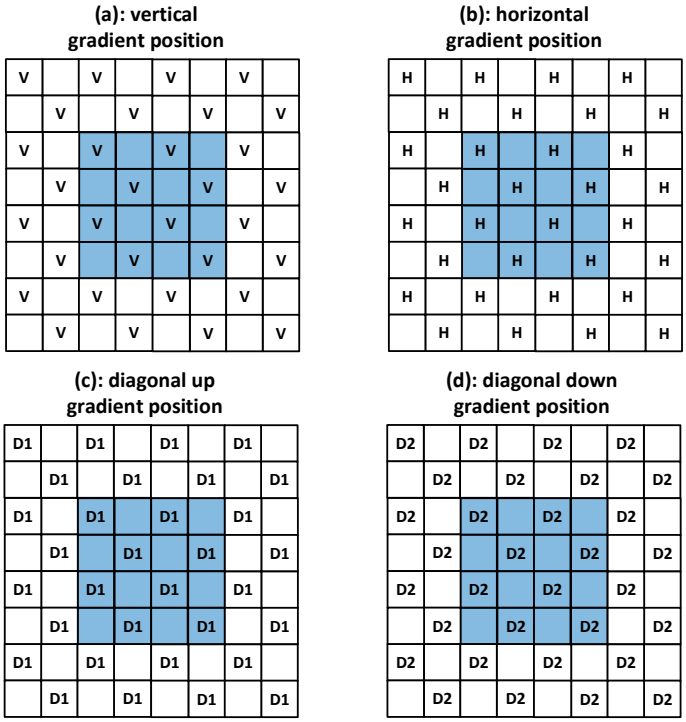


Fig. 3: Sub-sampled gradients in vertical, horizontal and two diagonal directions. Empty samples are not used for gradients computation. The samples of current 4×4 macro block are highlighted in blue color.

The quantized activity value $\widetilde{A}$ is calculated by Equation (7):

$$\widetilde{A} = \left\lfloor \sum_{k=i-2}^{i+5} \sum_{l=j-minY}^{j+maxY} (V_{k,l} + H_{k,l}) \right\rceil_0^4, \tag{7}$$

where i and j are the positions within the image relative to the top-left corner , $\lfloor A \rceil_0^4$ stands for a quantized value of A to the nearest integer and saturated in the interval $[0, 4]$. Finally, based on the values of D and $\widetilde{A}$, the corresponding class value C is derived by Equation (1). The class value is used to determine the filter coefficients to filter the Luma component.

Before applying the filter, ALF performs geometric transformations to the filter kernel. This is equivalent to applying these transformations to the samples in the filter support region. The idea is to make different filtered macro blocks more similar by aligning their directionality.

2) Geometric transformations: Three geometric transformations are introduced including rotation, vertical flip and diagonal flip expressed by Equations (8), (9) and (10), respectively.

$$\begin{aligned} f_r(k,l) &= f(N-l-1,k), \\ c_r(k,l) &= c(N-l-1,k), \end{aligned} \tag{8}$$

$$\begin{aligned} f_v(k,l) &= f(k,N-l-1), \\ c_v(k,l) &= c(k,N-l-1), \end{aligned} \tag{9}$$

$$\begin{aligned} f_d(k,l) &= f(l,k), \\ c_d(k,l) &= c(l,k), \end{aligned} \tag{10}$$

Algorithm 1 Pseudo-code to compute D

Input: g_{hv}^{max} maximum between horizontal and vertical gradients
g_{hv}^{min} minimum between horizontal and vertical gradients
g_{d}^{max} maximum between first and second diagonal gradients
g_{d}^{min} minimum between first and second diagonal gradients

```
if g_hv^max ≤ t1 g_hv^min and g_d^max ≤ t2 g_d^min then
  D ← 0
else
  if g_hv^max/g_hv^min > g_d^max/g_d^min then
    if g_hv^max > t2 g_hv^min then
      D ← 2
    else
      D ← 1
    end if
  else
    if g_d^max > t2 g_d^min then
      D ← 4
    else
      D ← 3
    end if
  end if
end if

return D
```

TABLE I: Mapping between the gradient values and the geometric transformations

Gradient value comparison	Transformation type
$g_{d2} < g_{d1}$ and $g_h < g_v$	No transformation
$g_{d2} < g_{d1}$ and $g_v < g_h$	Diagonal flip
$g_{d1} < g_{d2}$ and $g_h < g_v$	Vertical flip
$g_{d1} < g_{d2}$ and $g_v < g_h$	Rotation

where N is the size of the filter and $0 \leq k, l \leq N-1$ are coefficients coordinates, such that coordinates $(0,0)$ refer to the upper left corner and coordinates $(N-1, N-1)$ to the lower right corner. The transformations are applied to the filter coefficients $f(k,l)$ and to the clipping values $c(k,l)$ depending on gradient values calculated for the current macro block. The relation between the transformation and the four gradients of the four directions are summarized in Table I.

3) Luma 7×7 diamond shape filter: The ALF uses a 7×7 2D diamond shape filter for the Luma component as shown in Fig. 4:(a). The DSF is a point-symmetrical filter which allows factoring the coefficients two by two and thus reducing the number of multiplications by half. Since multipliers need much larger chip area than an adder or a subtractor, this factorization reduces significantly the processing part area by eliminating half of the multipliers for the ALF.

Using the symmetry, the total number of coefficients for the Luma filter is 12. These coefficients are of two types: 1) the first is the fixed filter coefficients, 2) the second is the customized filter coefficients for Luma signaled in APS.

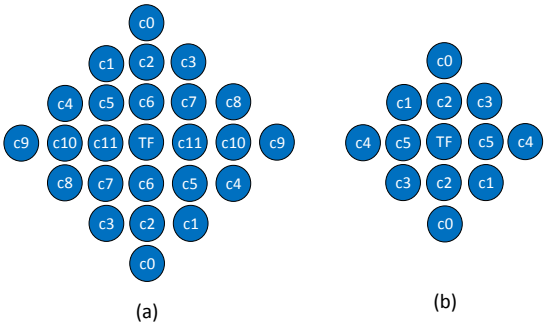


Fig. 4: Diamond Shape filters for Luma and Chroma, (a): Luma 2D 7×7, (b) Chroma 2D 5×5

VVC defines the fixed filter coefficients using two matrices. The first matrix, named *AlfFixFiltCoeff*, contains the 64 fixed Luma filters defined by VVC. The second matrix, named *AlfClassToFiltMap*, holds 16 sets of 25 indices that can be used in the *AlfFixFiltCoeff* matrix of size 16×25 to determine the Luma filter. The first index of the *AlfClassToFiltMap* is signaled in the slice header and can be in the range 0 to 15. The second index is determined based on the classification value C of the current $4\times$ 4 macro block, where C is an integer bounded in the interval $[0, 24]$. Based on these indices, the third index can be accessed from the *AlfClassToFiltMap* thus retrieving the filter coefficients defined in the *AlfFixFiltCoeff*. The second type of luma coefficients is the Luma coefficients in APS. This alternative uses coefficients that are signaled in APS. In one APS, up to 25 sets of Luma filter coefficients and clipping values could be signalled. The choice of these coefficients is determined by the encoder. This process depends on a certain criteria based on the pixel level analysis and rate-distortion optimisation. Each 4×4 macro block uses one filter set which can be selected among 25 filters using the classification value of the macro block.

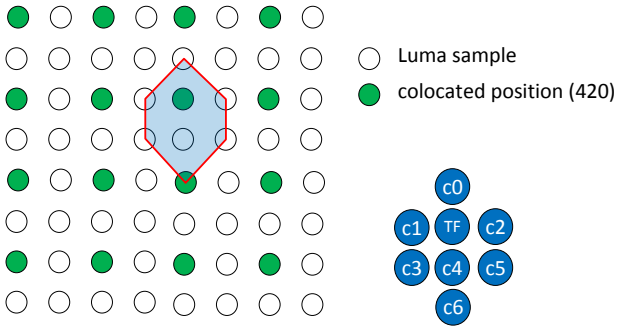


Fig. 5: Cross component 2D Diamond Shape filter, 420 subsampling example

C. Chroma filters

The ALF uses two DSF for the Chroma components, the first one is a 5×5 DSF shown in Fig. 4:(b) and the second one, shown in Fig. 5, is a cross-component filter which uses co-located Luma samples to filter Chroma ones. Like the Luma filter, the 5×5 DSF is point symmetrical where only 6 of

coefficients are signaled in the APS. On the opposite, the CCALF filter does not have the symmetrical propriety and needs a total of 7 coefficients for its filtering process. Unlike the Luma filter, the coefficients of Chroma filters can only be signaled in the APS, therefore there is no classification for Chroma macro blocks nor fixed coefficients. Eight sets of coefficients are transmitted in the APS, while the index of the used set is signaled at the CTB level.

III. Related Work

In this section, a brief description of existing hardware implementations of the ALF is given. Several ALF hardware implementations have been proposed in the literature, since it was previously proposed for the HEVC standard. Du *et al.* [20] proposed a hardware architecture for both deblocking filter and ALF. They implemented the ALF based on macro block size in raster scan order. The ALF needs the eight surrounding 4×4 final-deblocked blocks to process a 4×4 block. The authors concluded that the deblocking and ALF orders require large local buffers to enable data reuse for deblocking. Ruhan *et al.* [21] proposed a shared hardware architecture for the 5×5, 7×7 and 9×9 diamond shaped ALF filters using the fact that large filter sizes have small ones nested in them. The paper took advantage of the ALF symmetrical characteristics to reduce the total number of multiplications from 13 to 7 for the 5×5 DSF, from 25 to 13 for the 7×7 DSF and from 39 to 20 in the case of 9×9 DSF. This is carried-out by applying addition operations to the samples that have the same corresponding coefficient. The work in [21] was synthesized targeting an FPGA platforms of 40 nm technology. Fabiane *et al.* [22] proposed efficient hardware designs for the ALF cores of sizes 5×5, 7×7 and 9×9. Similar to [21], the symmetrical feature of the ALF is explored to reduce the total number of multipliers. The three architectures were synthesized targeting FPGA platform of 90 nm technology. Results showed that the proposed designs were able to process 110, 102 and 98 frames per second in 1080p resolution for the 5×5, 7×7 and 9×9 filter sizes, respectively. In [23], the same author presented a hardware architecture for the 5×5 ALF core targeting the same family of FPGA platforms.

However, the aforementioned studies do not address the gradient computation nor the memory management of the ALF which are considered as the most critical parts of the design. Moreover, a fixed system latency for all filter sizes is an important feature to ensure efficient interaction of the ALF module with other decoder modules. This paper proposes a complete study of the ALF module including the memory, the gradient and the filtering module. The proposed design leverages all the proprieties of the ALF including the symmetry, the subsampled gradients and the virtual boundary clipping to reach an efficient trade off between memory usage and chip surface. Our design uses 26 shared regular multipliers (RM) for the three filters in a pipe-lined architecture, fulfilling the target of 2 pixels/cycle throughput with a fixed system latency. To our knowledge, this is the first ALF hardware design study, compliant with the VVC standard targeting ASIC platform.

IV. Proposed Design

This paper proposes a novel hardware-architecture for the VVC ALF filter, considering its three filters: 7×7, 5×5 and 3×4 CCALF. This design is based on the Working Draft 10 [24] and the test model VTM10 of the VVC standard. The proposed design sustains a fixed throughput of 2 pixels/cycle with a fixed system latency that reaches processing a 4K video at 30 frames per second in 4:2:2 Chroma subsampling. The proposed design is integrated into a professional video decoder chip supporting multiple video coding standards.

This section is composed of three subsections. Subsection IV-A describes the top level design of the ALF module, then subsection IV-B describes the different memory configurations. The proposed scanning order for the optimal memory usage is presented in subsection IV-B, and finally subsection IV-D describes the unified architecture for the Luma and Chroma filters.

A. ALF top level design

Fig. 6 shows the top level design of the ALF module. The proposed design is decomposed of three main parts: the memory management, the gradients computation and the filtering module. The memory module (MEMORY_ACCESS_MODULE in Fig. 6) is the most critical part of the ALF module, and this design aims to reduce the cost of the memory. To find the lowest memory cost, we investigated multiple scanning orders for Luma and Chroma components. The lowest cost for memory is found when the scanning of Luma and Chroma samples is interlaced. This design uses two memories: the first stores Luma and Chroma macro blocks where the second one stores only Luma macro blocks. The memory handler takes care of memory accesses and selects the appropriate group of macro blocks to send to core module. The memory access of the design is detailed in the next section. The gradient module is applied only when Luma component is selected. It computes the sum of gradients and determines the classification value used to select the appropriate Luma filter coefficients. Finally, the filtering module aims to take advantage of the fact that larger filters have smaller ones nested inside. It allows the reuse of hardware ressources used in small filters to implement larger filters. Thus, it is possible to reduce the total hardware resources cost in comparison with an architecture that implements the three filters independently. For ASIC devices, this reduction in terms of resource consumption is important, since these devices are limited in terms of both size and battery.

B. Block scanning and memory optimisation

The ALF is applied at the CTU level. The CTU within a picture are scanned in raster order from left to right and from top to bottom. VVC supports CTU of sizes up to 128×128. To reduce the memory usage and eliminate the use of line buffer (storing an entire frame line of 4×4/4×2 Luma/Chroma macro blocks), the ALF introduces a set of clipping methods applied for different pixels close to the CTU boundaries. However, this does not eliminate totally the need for a memory, it only reduces the total number of macro blocks to store.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCE.2022.3146272, IEEE Transactions on Consumer Electronics

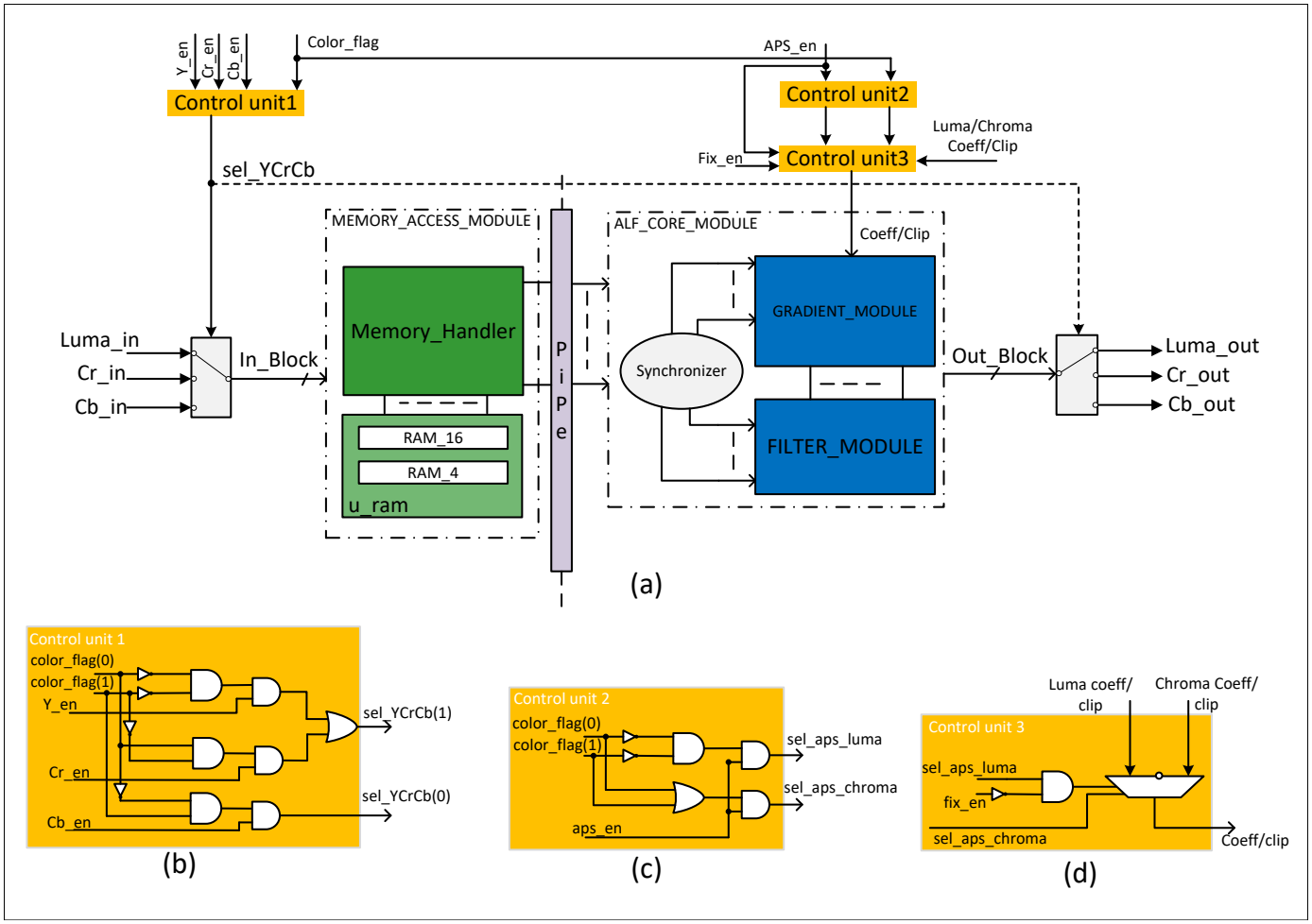


Fig. 6: Proposed hardware ALF architecture, (a): Unified design for all filters, (b): Control unit 1 initiates the start signals depending on the input color component. (c): Control unit 2 initiate the coefficients in APS enable signals depending on the input color component, (d): depending on the APS enable and the Fix_en signals, control unit 3 provides the core module with appropriate coefficients and clip values.

For the Luma component, the 7×7 DSF is applied for each 4×4 macro block using one among 25 filters determined by the classification results. To compute the class and apply the filter, each macro block needs its eight surrounding macro blocks: top left (TL), top (T), top right (TR), left (L), right (R), bottom left (BL), bottom (B) and bottom right (BR), as shown in Fig. 7:(a). Within the Luma coding tree block (CTB), macro blocks are scanned using raster scan order and when BR block is received, the current block, in orange, is processed.

Unlike the Luma, the Chroma components has two filters: the 5×5 DSF and the CCALF DSF, the coefficients of these filters are determined by the encoder and signaled, via the APS, to the decoder. The 5×5 DSF is applied to macro blocks of size of 4×2. Same as for the Luma component, to filter one Chroma 4×2 macro block, its eight surrounding macro blocks: TL, T, TR, L, R, BL, B and BR are needed, as shown in Fig. 7: (b). Following the raster scan order, when BR block is received, the current block, in orange, is filtered. On the other hand, the CCALF filter uses co-located Luma pixels to apply 4×3 CCALF filter, as shown Fig. 7:(b) for 4:2:0 chroma subsampling example. As it can be noticed, a

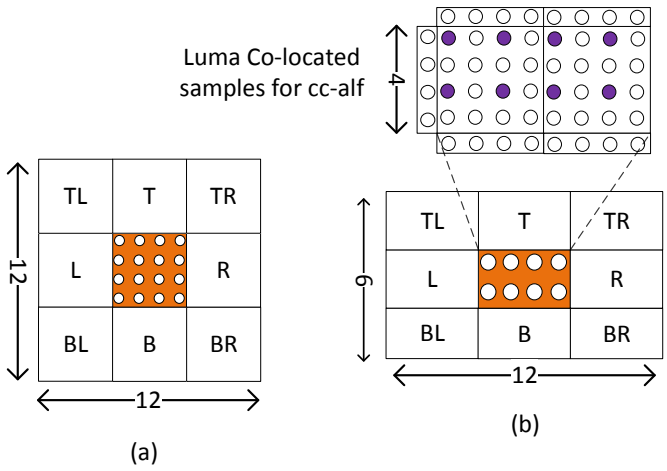


Fig. 7: (a) required macro blocks for Luma component (b) required macro blocks for Chroma component

group of surrounding Luma pixels are required to apply the CCALF filter. Finally, the filtered Chroma block is the sum of the result of the two filters (CCALF and 5 × 5 DSF).

In this paper, we propose a novel scanning order for the Luma and Chroma components that reduces the memory size from CTB wise to just lines of the CTB. It is important to note that to filter a Chroma macro block, Luma macro blocks need to be already stored in memory. This means that all required Luma macro blocks must be stored in memory before applying the CCALF. This fact led us to study multiple scanning order scenarios between Luma and Chroma components with the objective of minimizing the total required memory size. The main studied scenarios are the following:

- (A) Process the entire Luma CTB then start Chroma CTB: raster scan or Z-scan order for macro blocks.
- (B) Process the entire Chroma CTBs then start Luma CTB: raster scan or Z-scan order for macro blocks.
- (C) Alternate processing between Luma and Chroma macro blocks: one 4 × 2 Cr followed by one 4×2 Cb followed by two 4×4 Luma macro blocks.

Table II shows the memory usage for each scanning order scenario. For scenarios (A) and (B), the raster scan and the z-scan are almost similar in terms of memory usage, therefore, Table II reports the results for raster scan. As it can be noticed, the proposed scanning order reduces significantly the memory usage compared to the other two scenarios. For scenario A, the entire Luma CTB is stored so that the CCALF can be applied when starting the Chroma components. Compared to the proposed scanning order, it consumes 75% more memory. For scenario B, the entire Chroma CTBs is processed using the 5×5 DSF, however it is not possible to start the CCALF since the Luma components are still not available. As a result, the processed Chroma CTBs are stored, waiting for the CCALF filter result, which is applied when the Luma component is received. Compared to the proposed scanning order, it consumes 74% more memory. To conclude, the lowest memory can only be achieved when the Luma and Chroma macro blocks are interlaced, which led us to define our own scanning order which is an alternation between color components of the same CTU line. The ALF module alternates between Luma and Chroma macro blocks by reading for each CTU line one 4×2 Chroma Cr macro block followed by one 4×2 Chroma Cb macro block followed by two 4×4 Luma macro blocks. The proposed scanning order is described in details in the next subsection.

TABLE II: Memory usage (Kbits) comparison for different scanning order scenarios.

Scanning scenarios	A	B	C
7×7 DSF	163.84	20.48	20.48
5×5 DSF	25.60	163.84	25.60
3×4 CCALF	0*	0*	1.28
Total size	189.44 (100%)	184.32 (97%)	47.36 (25%)

* Required samples are already in memory.

C. Proposed scanning order

As shown in Fig. 7:(a), each 4×4 Luma macro block needs its eight surroundings. Starting from the TL to the BR macro block. The current macro block starts the processing only when the BR is received. This implies that the top two macro block lines of the Luma CTB need to be stored in memory. These lines of 4×4 macro blocks are measured on the maximum size of a CTB supported by the VVC standard which is 128×128, which results in storing 8×128 pixels. Using these two macro block lines (8 pixel lines) in a rotating memory, the entire Luma CTB can be filtered. Moreover, the last column of a CTB can be filtered only once the next CTB is available. As a result, macro blocks belonging to the last column of the CTB need to be stored in memory along with its TL, L, BL, T and B neighbors. As a result, for Luma components, two lines and two columns of 4×4 macro blocks measured on a CTB of size 128×128 are stored. Thus, the total random-access memory (RAM) size for Luma components is 20.48 Kbits for 10-bits pixels.

Like Luma, each 4×2 Chroma macro block is filtered, with the 5×5 DSF, using its eight neighbors, as shown in Fig. 7:(b). This requires to store a total of two Chroma macro block lines in memory. These macro block lines are measured on 128×64 Chroma CTBs, for 4:2:2 sub-sampling, which means a total of 4×64 pixels are stored. As for Luma, the last column of the Chroma CTB can be filtered only once the first column of the next CTB is available. As a result, for the 5×5 DSF, two lines and two columns of 4×2 macro blocks measured on a CTB of size 128×64 are stored, which corresponds to a total of 4×64 pixels for line memory and 8×128 pixels for column memory. Thus, the total RAM size for the 5×5 DSF for both Chroma components (Cr and Cb) is 25.60 Kbits for 10-bits pixels, supporting 4:2:2 Chroma sub-sampling.

Fig. 8 shows an example of Luma/Chroma scanning order for a CTU of size 32×32 for 4:2:0 chroma sub-sampling. The Luma, Cb and Cr chroma samples are illustrated in Fig. 8 by gray, blue and yellow colors, respectively. Fig. 8:(b) shows the alternation pattern between Luma and Chroma macro blocks. As shown in the figure, one Chroma Cr followed by one Chroma Cb followed by two Luma macro-blocks are processed. This pattern is repeated until the CTB finishes the filtering process. The macro blocks of consecutive numbers and color in the figure belong to the same CTB. For each color component, four CTBs are shown with different color intensities. As it can be noticed, the ALF is actually applied to a shifted version of the CTB with the same size, delimited by the green rectangle in Fig. 8:(a). For the Chroma component, once the macro blocks of position 31 for Cr (or Cb) components is available, the macro block at positions 26, from the previously received line, is processed. For the Luma component, macro blocks of positions 53 and 54 are processed once the macro blocks of positions 62 and 63, respectively, are available. The distance between received macro block and the current macro block is one CTB line, which can be seen in the input/output sequencing in Fig. 8:(b). As it can be noticed from the timeline, the CCALF filter can be applied in parallel with the 5×5 DSF taking advantage of the fact that co-located Luma macro blocks are already retrieved from the memory when applying the 7×7 DSF. As a result, no further

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCE.2022.3146272, IEEE Transactions on Consumer Electronics

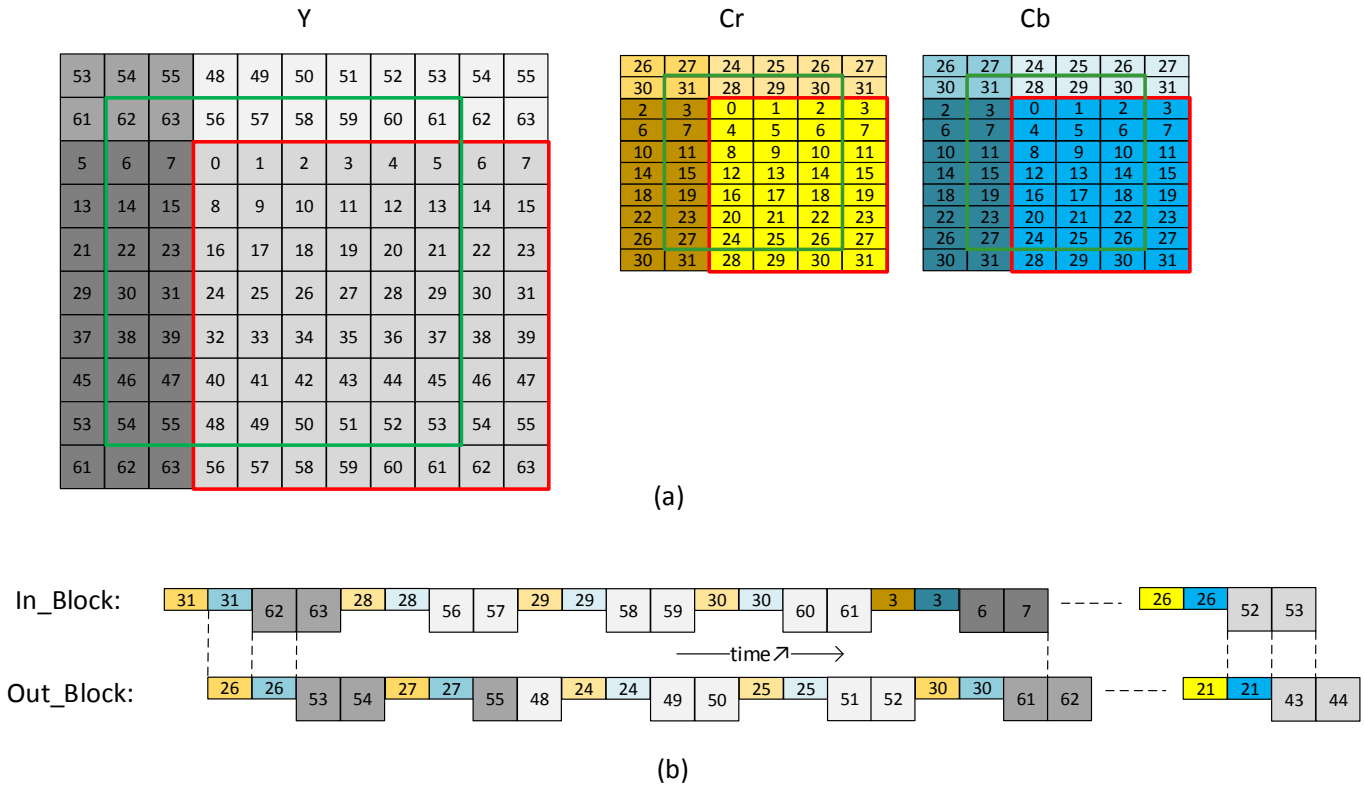


Fig. 8: ALF macro block scanning for 4:2:0 sub-sampling. (a): CTU32 example: each color intensity represents a CTB. The red rectangle is the current CTB. The green rectangle represents the block processed by the ALF (shift version of the current CTB). (b) input/output macro block sequencing: one 4×2 Cr macro block followed by one 4×2 Cb macro block followed by two 4×4 Y macro blocks.

memory accesses or memory storage are required for this filter. However, since the last column of the Chroma CTB is filtered at the reception of the first column of next CTB, a column of 1 pixel wide is stored in memory. Without that, when the the second CTB is available, the CCALF filter would be missing the left co-located samples. Thus, the memory required for CCALF is 1.28 Kbits ($128\times10-bits$). As a result, for the two Chroma filters, a total of 26.88 Kbits (25.60 required for 5×5 DSF + 1.28 for CCALF) for the RAM are required for 10-bits pixels for supporting 4:2:2 and 4:2:0 Chroma sub-sampling.

Finally, the total memory used for both Luma and Chroma components is 47.36 Kbits of RAM (20.48 for Luma 7×7 DSF + 25.60 for Chroma 5×5 DSF + 1.28 for CCALF). Compared to filtering color components in order (Luma CTB followed by Chroma Cr CTB followed by entire Chroma Cb CTB), the proposed scanning order reduces the memory by 75%.

D. Luma/Chroma unified filtering architecture

After selecting the right group of macro blocks from the memory, the next stage of the ALF depicted by the ALF_CORE_MODULE in Fig. 6: (a) is executed. This module is composed of two sub-modules: the gradient module and the unified filtering module. The gradient module, applied only when Luma samples are processed, computes the sum of gradients and determines the classification value used to select the appropriate Luma filter coefficients which are then delivered to the filtering module. The filtering module is a unified design that uses 26 multipliers to apply the three Luma/Chroma filters : 7×7 DSF, 5×5 DSF and 3×4 CCALF.

The gradient module is actually formed by three sub-modules: the gradient sum, the read-only memory (ROM), and the kernel transpose. The first module is the gradient sum module. In this module the sum of gradients is computed following Equations (3), (2), (4) and (5). Using the gradient result, the classification value is determined and the filter coefficients to be used for the Luma component are selected. When one of the Chroma components is chosen, the gradient sum module is bypassed and the it delivers directly the Chroma coefficients to the filtering module. These coefficients (whether Luma or Chroma) are synchronized with the required data and delivered to the filtering module. The second module is the ROM memory which is used to store the fixed Luma coefficients defined by the VVC standard. Luma components have also coefficients that are signaled in the APS. In all cases, from the ROM or from the APS, one Luma filter is selected. But, before transmitting these coefficients to the filtering module a certain transposition is applied to the filter kernel. This transposition is lead by the transpose module, where, depending on the gradient sum results a rotation, vertical flip or horizontal flip is applied to the Luma filter, according to the conditions presented in Table I.

The last sub-module of the ALF core is the filtering module,

TABLE III: Performance (%) in terms of Bjøntegaard delta rate (BD-BR) and run time complexity (%) when ALF, CCALF and SAO tools turned off in VTM10.0 [25]. Evaluations performed under the VVC common test conditions [26].

Disabled tool	All Intra Main 10					Random Access Main 10					Low Delay B Main 10				
	BD-BR (%)			Complexity (%)		BD-BR (%)			Complexity (%)		BD-BR (%)			Complexity (%)	
	Y	U	V	*EncT*	*DecT*	Y	U	V	*EncT*	*DecT*	Y	U	V	*EncT*	*DecT*
ALF	2.20	12.81	12.43	100	92	4.34	19.31	19.06	97	89	4.12	25.06	19.58	99	92
CCALF	−0.14	9.69	8.55	100	97	−0.13	13.88	13.73	100	99	−0.15	18.58	13.79	100	97
SAO	0.01	0.15	0.20	100	91	0.08	0.14	0.31	97	96	0.11	0.43	1.23	97	93

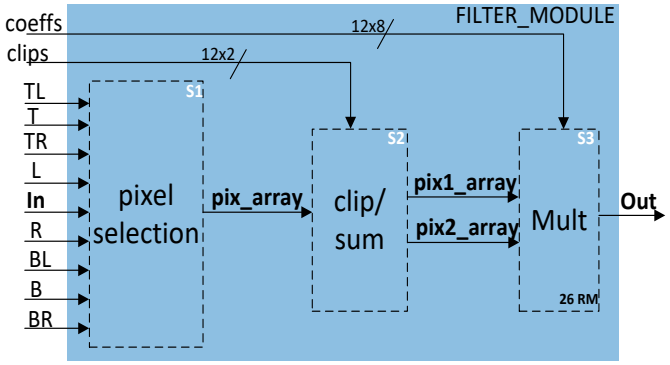


Fig. 9: ALF filter module

shown in Fig. 9. At the input, the module receives nine 4 × 4 macro blocks containing the current macro block (*In*) with its eight surroundings, along with the filter clipping and coefficients values. At the output, it delivers the filtered macro block at a throughput of 2 pixels/cycles. This module is composed of 3 pipeline stages including S1: pixel selection, S2: clip/sum and S3: mult_stage. The S1 stage is used for pixel selection. In fact, for a given pixel position (x, y) within the 4×4 macro-block, it selects the required neighbour pixels to use for the core filter. The selected pixels are then transmitted, via the "pix_array" register to the pre-multiplication stage (S2). As we know, ALF filters are point symmetrical, this allows to add symmetrically positioned pixels before applying the multiplication. Therefore, stage two performs additions and clipping operations. The results of this stage are delivered to the third and final stage via two separate arrays (pix1_array, pix2_array), each representing one pixel. In order to process 2 pixels/cycle, stage three uses a total of 26 RM. Finally, the 4×4 ALF filtered macro block is delivered via the *Out* interface.

V. Experimental Results

A. Experimental Setup

The proposed design was implemented by the VHDL hardware description language. A state-of-the-art logic simulator [27] is used to test the functionality of the ALF memory and core multiplication modules. The test strategy is as follows: a software implementation of the ALF has been developed, based on the specification of the last version of the VVC standard. Using self-check techniques, the bit accurate testbench compares the simulation results with those obtained using the reference software implementation. In the next section we evaluate the coding gain and the software complexity of the ALF.

B. ALF coding gain and software complexity

The coding gain and software complexity of the ALF and CCALF modules under the VVC common Test conditions in three main coding configurations including all intra (AI), random access (RA) and low delay B are analysed. Table III gives the coding losses and the complexity reductions at both encoder and decoder when the ALF and CCALF tools are turned off. It can be noticed that disabling ALF and CCALF tools introduces a high coding losses for Chroma components for all configurations with a maximum of 25.06% and 18.58% BD-BR (U component) for low delay configuration when these two tools are disabled, respectively. Compared to SAO, disabling the ALF tools introduces larger coding loss specifically for the Chroma components. This is due to the fact that the ALF uses two filters for the Chroma components: the 5 × 5 DSF and the CCALF. Fig. 10, shows the complexity of the ALF compared to other VVC tools in terms of decoding time. As it can be noticed, the ALF, embedding the CCALF, is considered as one of the most complex tool of the VVC decoder. Compared to the DBF, ALF takes more than 24% of decoding complexity and it comes second after the motion compensation (MC) for both QP values.

C. Synthesis Results and Analysis for ASIC platform

TABLE IV: Synthesis results for ALF @2 p/c at 600 MHz on ASIC 28-nm platform for typical condition of 85 Celsius with 0.65 input voltage

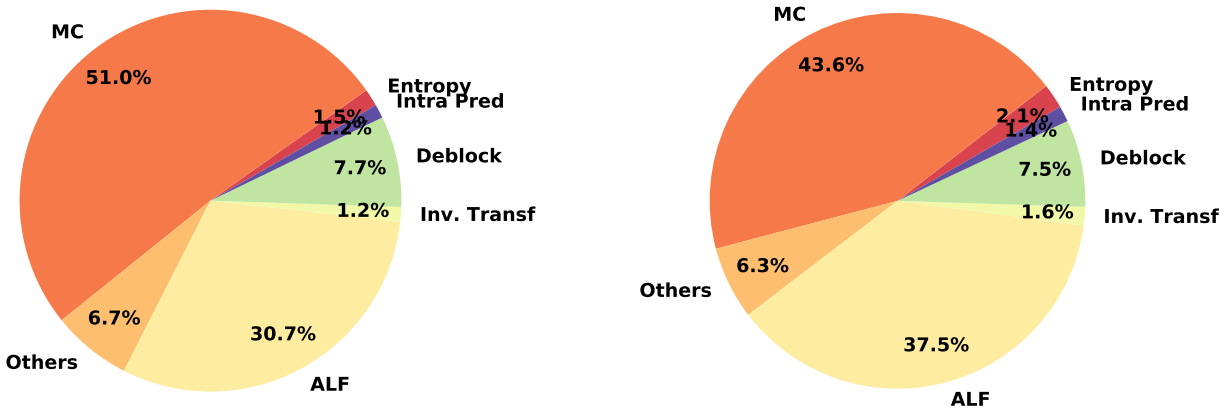
	Memory access module	Gradient module	Filter module
Number of mult.★	0	0	26
Combinational area	9368	10398	12910
Non combinational area	20281	4368	5115
Total area	30758	14767	18025
RAM (bits)	47360	-	-

★ The number of multipliers is included in the total number of gates.

TABLE V: Power results for ALF @2 p/c at 600 MHz on ASIC 28-nm platform for typical condition of 85 Celsius

	0.8 V	0.65 V
Power (mW)	9.5819	6.6921
Total area (gate count)	66997	76521

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCE.2022.3146272, IEEE Transactions on Consumer Electronics



(a) QP 37 (b) QP 32

Fig. 10: Complexity distribution of decoding tools in the VTM software for two different QPs.

TABLE VI: Adaptive Loop Filter vs Inverse transform blocks (MTS + LFNST) @2 p/c

	ALF	iMTS	iLFNST
number of mult.★	26	32	32
Total area	66997	89082	74592
RAM (bits)	47360	-	-

★ The number of multipliers is included in the total number of gates.

Table IV gives the results of ALF module in terms of chip surface and memory consumption targeting ASIC platforms at 600 Mhz. This table also shows how resources are distributed over the sub-modules of the ALF. The ALF design operates at 600 MHz with a total of 66.997 Kgate and a total 47.36 Kbits of RAM used to store two lines of 4×4 macro blocks for each color component and one additional Luma column of 1 sample wide measured for CTUs of size 128×128. The memory access module consumes 48% of design total resources while the gradient and the filter modules consumes only 23% and 29%, respectively. The proposed design was integrated into a real time video decoder enabling to decode a 4k video at 60 frame per second (fps) for 4:2:2 chroma subsampling. The decoder also supports the three recent MPEG video standards including AVC/H.264, HEVC/H.265 and VVC/H.266 standard.

To further evaluate the performance, the ALF design is synthesised for two input voltages at a typical condition of a temperature of 85 Celsius, the result can be seen in Table V. As it can be noticed from the table, for 0.8V the power consumption is about 9 mW which is almost 30% larger than the one with the 0.65V which is about 6 mW. Reducing the power consumption comes at the expense of larger chip area, the one targeting 0.65 V is 12% larger than the first one. In Table VI, the result of this study are compared to our previous study [12] done for the VVC inverse transform block. Both studies were done in the same environment and got integrated in a real time 4Kp60 multi-standard decoder, which allows an accurate comparison. Both the ALF and transform block can be reduced to a matrix multiplication problem. Taking advantage of the butterfly decomposition and the zeroing for large block sizes, the inverse MTS can be computed using only 32 multipliers for two pixels/cycle. The inverse LFNST on the other hand, is applied to the top left 8×8 corner of the input transform block, this allows it to use only 32 multipliers for the same throughput. The largest filter for the ALF is the 7×7 DSF and by taking advantage of the symmetrical feature of the filter, the ALF is able to process 2 pixels/cycle using only 26 multipliers. Compared to the transform blocks, the ALF uses less multipliers which allows it to consume 11% less area than the LFNST and 24% less area than the MTS. Since the ALF requires neighboring samples to apply the filters, it uses about 47Kbits of memory, on the other hand, the 1-D MTS and the LFNST do not require any memory where the blocks are independently processed.

D. Comparison with state-of-the-art

A fair comparison with the state of the art solutions is quite difficult since most of works focus on earlier versions of the ALF and do not support the memory management and the gradient computation. Table VII give the key performance of state-of-the-art FPGA and ASIC-based works. The only work that discusses the memory is found in [20], however it focuses on the chaining between the DBF and ALF and does not provide the results of memory usage. Compared to previous works, the net memory results related to the ALF and CCALF are provided. Also, a study of a shared architecture for all ALF filters based the last version of the VVC standard [24] is provided.

The solutions proposed in [21–23] discuss only the calculation part of ALF filters and do not address the memory. Unlike these solution, our solution provides a complete

TABLE VII: Comparison of different ALF hardware designs on FPGA and ASIC platforms.

Solutions	Du *et al.* [20]	Conceicao *et al.* [21]	Redies *et al.* [22]	Redies *et al.* [23]	Proposed design
Platform	FPGA 65 nm	FPGA 90nm	FPGA 90nm	FPGA 40 nm	ASIC 28 nm
9×9 DSF	✓	✓	✓	✗	✗
7×7 DSF	✓	✓	✓	✗	✓
5×5 DSF	✓	✓	✓	✓	✓
Gradient sum	✗	✗	✗	✗	✓
CCALF	✗	✗	✗	✗	✓
Chroma sub-sampling	$4:2:0$	$4:2:0$	$4:2:0$	$4:2:0$	$4:2:2/4:2:0$
LUTs	15561	1660	2071	113	✗
Gates	✗	✗	✗	✗	66997
Frequency (MHz)	211	115	229	41	600
DSPs/RMs	41	39	20	13	26★
Throughput (fps)	1920 × 1080 30 fps	3840 × 2160 33 fps	2560×1600 49 fps	1920 ×1080 19 fps	3840×2160 30 fps
Memory	✗	✗	✗	✗	47.36 Kb

★ The number of multipliers is provided for information since it is already included in the total number of gates.

study for the ALF hardware implementation and discusses the most challenging part which is the memory. Like the work of [21, 22], a shared architecture is proposed for multiple filters taking advantage of the fact that smaller filters are nested within larger ones by using a number of regular multipliers. Compared to [21], our solution consumes less multipliers while supporting the same throughput. This is because [21] supports a previous version of the ALF that uses 9×9 DSF which was not included in the VVC specification. Compared to [22], our solution supports larger throughput and more Chroma sub-sampling, this comes at the use of more multipliers. Solution in [23] provides a study only for the 5×5 DSF which is considered to be little. To the best of our knowledge, our solution is the first to address a complete study for ALF based on the last version of the VVC standard [24].

The proposed design is integrated into a video decoder that supports three different video standards including AVC/H.264, HEVC/H.265 and the last MPEG VVC/H.266. The proposed architecture operates at 600Mhz with 66.997 Kgate and with 47.36 Kbits of RAM spread over two RAMs, one of size 46.08 Kbits for the 7×7 and 5×5 DSF and the second of size 1.36 Kbits for the CCALF filter. The design is able to decode in real-time 4k video at 30fps for 4:2:0 and 4:2:2 Chroma sub sampling formats.

VI. Conclusion

In this paper a hardware implementation of the adaptive loop filter (ALF) has been investigated targeting a real time 4Kp30 VVC decoder on ASIC platforms. The ALF is part of the in-loop carried-out at the end of the coding loop. It aims at reducing visible artifacts from the reconstructed image by performing adaptive filtering to minimize the mean squared error (MSE) between original and reconstructed samples based on Wiener filtering. The proposed hardware implementation relies on regular multipliers and sustains a constant system latency with a fixed throughput of 2 pixels per cycle. It uses a total of 26 multipliers in a pipe-lined architecture for all ALF filters including the 7×7, 5×5 and CCALF diamond shape filters. It is the first hardware design that addresses the memory architecture of the ALF. By creating an alternation between Luma and Chroma components, the memory size is reduced at most, from CTB wise to just lines of CTB. The proposed design is able to reach real time decoding of 4K video 4:2:2 at 30 frames par second. This design has been successfully integrated in a hardware ASIC decoder supporting recent MPEG standards including AVC, HEVC and VVC.

References

[1] B. Bross, Y.-K. Wang, Y. Ye, S. Liu, J. Chen, G. J. Sullivan, and J.-R. Ohm, "Overview of the versatile video coding (vvc) standard and its applications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 10, pp. 3736–3764, 2021.

[2] W. Hamidouche, T. Biatek, M. Abdoli, E. François, F. Pescador, M. Radosavljević, D. Menard, and M. Raulet, "Versatile video coding standard: A review from coding tools to consumers deployment," *arXiv preprint arXiv:2106.14245*, 2021.

[3] F. Bossen, X. Li, A. Norkin, and K. Sühring, "Jvet-o0003," in *Meeting Report of the 16th Meeting of the Joint Video Experts Team (JVET), Geneva*. JVET AHG report: Test model software development (AHG3), July. 2019.

[4] N. Sidaty, W. Hamidouche, P. Philippe, J. Fournier, and O. Déforges, "Compression Performance of the Versatile Video Coding: HD and UHD Visual Quality Monitoring." Picture Coding Symposium (PCS), November 2019.

[5] F. Pakdaman, M. A. Adelimanesh, M. Gabbouj, and M. R. Hashemi, "Complexity Analysis Of Next-Generation VVC Encoding And Decoding," *2020 IEEE International Conference on Image Processing (ICIP)*, Oct 2020.

[6] M. Saldanha, G. Sanchez, C. Marcon, and L. Agostini, "Complexity analysis of vvc intra coding," in *2020 IEEE International Conference on Image Processing (ICIP)*, 2020, pp. 3119–3123.

[7] M. J. Garrido, F. Pescador, M. Chavarrías, P. J. Lobo, and C. Sanz, "A High Performance FPGA-Based Architecture for the Future Video Coding Adaptive Multiple Core Transform," *IEEE Transactions on Consumer Electronics*, vol. 64, no. 1, pp. 53–60, 2018.

[8] M. J. Garrido, F. Pescador, M. ChavarrÃas, P. J. Lobo, and C. Sanz, "A 2-d multiple transform processor for the versatile video coding standard," *IEEE Transactions on Consumer Electronics*, vol. 65, no. 3, pp. 274–283, Aug 2019.

[9] A. Kammoun, W. Hamidouche, F. Belghith, J. Nezan, and N. Masmoudi, "Hardware Design and Implementation of Adaptive Multiple Transforms for the Versatile Video Coding Standard," *IEEE Transactions on Consumer Electronics*, vol. 64, no. 4, October 2018.

[10] A. Mert, E. Kalali, and I. Hamzaoglu, "High Performance 2D Transform Hardware for Future Video Coding," *IEEE Transactions on Consumer Electronics*, vol. 62, no. 2, May 2017.

[11] Y. Fan, Y. Zeng, H. Sun, J. Katto, and X. Zeng, "A pipelined 2d transform architecture supporting mixed block sizes for the vvc standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 9, pp. 3289–3295, 2020.

[12] I. Farhat, W. Hamidouche, A. Grill, D. Menard, and O. Déforges, "Lightweight hardware implementation of vvc transform block for asic decoder," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 1663–1667.

[13] I. Farhat, W. Hamidouche, A. Grill, D. Ménard, and O. Déforges, "Lightweight hardware transform design for the versatile video coding 4k asic decoders," *IEEE Transactions on Consumer Electronics*, vol. 67, no. 4, pp. 329–340, 2021.

[14] M. Karczewicz, N. Hu, J. Taquet, C.-Y. Chen, K. Misra, K. Andersson, P. Yin, T. Lu, E. François, and J. Chen, "Vvc in-loop filters," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 10, pp. 3907–3925, 2021.

[15] C. Tsai, C. Chen, T. Yamakage, I. S. Chong, Y. Huang, C. Fu, T. Itoh, T. Watanabe, T. Chujoh, M. Karczewicz, and S. Lei, "Adaptive loop filtering for video coding," *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, no. 6, pp. 934–945, 2013.

[16] C. Chen and al, "The adaptive loop filtering techniques in the hevc standard," *Proceedings of SPIE Applications of Digital Image Processing*, vol. 7, no. 35, p. 849913, 2012.

[17] B. Bross, J. Chen, S. Liu, and Y.-K. Wang, "Algorithm description for versatile video coding and test model 8 (vtm 8)." oint Video Experts Team (JVET) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, April 2020.

[18] A. Norkin, G. Bjontegaard, A. Fuldseth, M. Narroschke, M. Ikeda, K. Andersson, M. Zhou, and G. Van der Auwera, "Hevc deblocking filter," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1746–1754, 2012.

[19] C. Fu, E. Alshina, A. Alshin, Y. Huang, C. Chen, C. Tsai, C. Hsu, S. Lei, J. Park, and W. Han, "Sample adaptive offset in the hevc standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1755–1764, 2012.

[20] J. Du and L. Yu, "A parallel and area-efficient architecture for deblocking filter and adaptive loop filter," in *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*, 2011, pp. 945–948.

[21] R. Conceição, F. Rediess, B. Zatt, M. Porto, and L. Agostini, "Configurable hardware design for the hevc-based adaptive loop filter," in *2014 IEEE 5th Latin American Symposium on Circuits and Systems*, 2014, pp. 1–4.

[22] F. Rediess, L. Agostini, C. Cristani, P. Dall'Oglio, and M. Porto, "High throughput hardware design for the adaptive loop filter of the emerging hevc video coding," in *2012 25th Symposium on Integrated Circuits and Systems Design (SBCCI)*, 2012, pp. 1–5.

[23] F. Rediess, C. Cristani, P. Dall'Oglio, M. Porto, and L. Agostini, "Architectural design for the adaptive loop filter of the emerging high efficiency video coding standard," in *XXVII SIM - South Symposium on Microelectronics*, 2011, pp. 1–5.

[24] J. Chen, Y. Ye, and S. H. Kim, "Algorithm description for versatile video coding and test model 10 (vtm 10)." JVET AHG report: Test model software development (AHG3), October 2020.

[25] W.-J. Chien and J. Boyce, "Jvet ahg report: Tool reporting procedure (ahg13)." JVET document T0013, 19th meeting by teleconference, July 2020.

[26] X. L. K. Sühring, "Jvet common test conditions and software reference configurations." JVET Document JVET-H1010, meeting by teleconference, November 2017.

[27] RivieraPro-Aldec-Functional-Verification-Tool. [Online]:https://www.aldec.com/en/products/ functional_verification/riviera-pro.

Ibrahim Farhat was born in Eljem, Tunisia, in 1993. He received the engineering degree in communication systems and computer science from SUP'COM school of engineering, Tunis, in 2018. In 2019, he joined the Institute of Electronic and Telecommunication of Rennes (IETR), Rennes, and became a member of VITEC company hardware team, France, where he is currently pursuing the Ph.D. degree. His research interests focus on video coding, efficient real time and parallel architectures for the new generation video coding standards, and ASIC/FPGA hardware implementations.

Wassim Hamidouche received Master's and Ph.D. degrees both in Image Processing from the University of Poitiers (France) in 2007 and 2010, respectively. From 2011 to 2013, he was a junior scientist in the video coding team of Canon Research Center in Rennes (France). He was a post-doctoral researcher from Apr. 2013 to Aug. 2015 with VAADER team of IETR where he worked under collaborative project on HEVC video standardisation. Since Sept. 2015 he is an Associate Professor at INSA Rennes and a member of the VAADER team of IETR Lab. He has joined the Advanced Media Content Lab of b<>com IRT Research Institute as an academic member in Sept. 2017. His research interests focus on video coding and multimedia security. He is the author/coauthor of more than one hundred and forty papers at journals and conferences in image processing, two MPEG standards, three patents, several MPEG contributions, public datasets and open source software projects.

Adrien Grill was born in 1988 in Aix-en-Provence, France. After receiving his engineering degree at Supelec in 2010, he specialized in hardware algorithm implementation. He joined Vitec in 2014, and currently works as technical leader on codec implementation projects. His fields of interest include signal, image processing, and video coding.

Daniel Ménard received the Ph.D. and Habilitation degrees in Signal Processing and Telecommunications from the University of Rennes, respectively in 2002 and 2011. Since 2012, he has been Full-Professor at INSA Rennes - department of Electrical and Computer Engineering and member of the IETR lab. He has 20 years of expertise in the design and implementation of image and signal processing systems. His research interests include low power video codecs, approximate computing and energy consumption. He has a long experience of collaborative projects, he has been involved in different national and European projects He is currently member of different Technical Program Committees of international conferences (ICASSP, SiPS and DATE). Since 2018, he has been an elected member of the Technical Committee ASPS of the IEEE Signal Processing society. He has published more than 100 scientific papers in international journal and conferences.

Olivier Déforges received the Ph.D. degree in image processing in 1995. He is a Professor with the National Institute of Applied Sciences (INSA) of Rennes. In 1996, he joined the Department of Electronic Engineering, INSA of Rennes, Scientic and Technical University. He is a member of the Institute of Electronics and Telecommunications of Rennes (IETR), UMR CNRS 6164 and leads the IMAGE Team, IETR Laboratory including 40 researchers. He has authored over 130 technical papers. His principal research interests are image and video lossy and lossless compression, image understanding, fast prototyping, and parallel architectures. He has also been involved in the ISO/MPEG standardization group since 2007.