VLSI architectures of a wiener filter for video coding

electronics

MDPI

Article

VLSI Architectures of a Wiener Filter for Video Coding

Walid Walid *, Giorgio Armanno, Sandro Di Paola, Massimo Ruo Roch and Guido Masera and Maurizio Martina *

Department of Electronics and Telecommunications, Politecnico di Torino, Corso Duca Degli Abruzzi, 24, 10129 Torino, Italy; Giorgio.Armanno@studenti.polito.it (G.A.); Sandro.Paola@studenti.polito.it (S.D.P.); massimo.ruoroch@polito.it (M.R.R.); Guido.Masera@polito.it (G.M.);
* Correspondence: walid.walid@polito.it (W.W.); maurizio.martina@polito.it (M.M.)

Abstract: In the modern age, the use of video has become fundamental in communication and this has led to its use through an increasing number of devices. The higher resolution required for images and videos leads to more memory space and more efficient data compression, obtained by improving video coding techniques. For this reason, the Alliance for Open Media (AOMedia) developed a new open-source and royalty-free codec, named AOMedia Video 1 (AV1). This work focuses on the Wiener filter, a specific loop restoration tool of the AV1 video coding format, which features a significant amount of computational complexity. A new hardware architecture implementing the separable symmetric normalized Wiener filter is presented. Furthermore, the paper details possible optimizations starting from the basic architecture. These optimizations allow the Wiener filter to achieve a 100× reduction in processing time, compared to existing works, and 5× improvement in megasamples per second.

Keywords: video coding; AV1; Wiener filter; VLSI

Citation: Walid, W.; Armanno, G.; Di Paola, S.; Ruo Roch, M.; Masera, G.; Martina, M. VLSI Architectures of a Wiener Filter for Video Coding. *Electronics* **2021**, *10*, 1961. https://doi.org/10.3390/electronics10161961

Academic Editor: Byung-Gyu Kim

Received: 14 July 2021
Accepted: 11 August 2021
Published: 14 August 2021

1. Introduction

In the last years, the need for an open media codec has increased with the growth of internet video contents since the triumph of the internet is founded on the fact that the basic technologies (such as browsers, operating system, etc.) are open and available to be freely implemented. Combining these needs led several big companies to create some alternatives to codecs with complex and expensive royalties. The main goal was to create a new generation of video coding, to share video fast, easy and at low cost. In this panorama, Mozilla, Google and Cisco, with Amazon and Netflix and some hardware vendors like AMD and Intel, founded AOMedia in 2015 that, in 2018, published the first version of AV1 [1,2], a video codec largely based on VP9 [3]. Still, including many significant improvements, primarily the full compatibility with W3C Patent Policy [4]: essentially, it can be fully implemented with royalty-free licensing requirements. The basic idea was to start from the analysis of the entire AV1 codec and then focus on a particular part based on the "*profiling*" results of the AV1 Software model [5] to understand the usage percentage of each one and evaluate which one needed more attention. From this analysis, the attention turned on Wiener filter [6]. The importance of the Wiener filter in image processing is highlighted by the authors in [7,8]. The Wiener filter has many application in video domain [9,10]. It is also used in other application including speech processing, noise reduction, deblurring, etc. [11–14].

As mentioned, the Wiener filter reduces noise and removes blurring [15,16]. Among other signal processing applications, the Wiener filter can be used in de-convolution, noise reduction, signal detection [17]. The Wiener filter is used to reconstruct a degraded frame by means of a non-causal filter. Each frame pixel is taken with a $w \times w$ window around it. w is an odd number such that $w = 2r + 1$, where r is an integer number representing the radius of involved window [5,10]. Thus the filtering block, instead of operating on

$w \times w$ or w^2 input taps, operates on a processed version of the input taps. These taps are contained in the matrices H and M. In particular, H is given by

$$H = E[XX^T] \tag{1}$$

that is the autocovariance of X, the column-vectorized version of the w^2 input taps, where $E[\cdot]$ corresponds to the expectation operation. M is given by

$$M = E[YX^T] \tag{2}$$

that is the cross correlation between X and the source pixel Y. This approach requires transmitting w^2 values for each filtered pixel, and this will increment both bit rate cost and decoding complexity. For this reason, some constrains are imposed [5,10]:

- The resultant filter has to be separable;
- Each horizontal and vertical filter has to be symmetric;
- Horizontal and vertical filter coefficients cannot take any possible value. Their sum must be exactly S for both filters, where S is a constant value that, for the AV1 implementation, is equal to 2^{16}.

These constraints allows to send, for each filter, just r values instead of w. Moreover, since the filter is now symmetric, it operates only to compute the first r elements. Thus, the implementation complexity is reduced considering that both the vertical and horizontal filter, from now on called a and b, respectively, can be reconstructed from the r values. They can be derived as follows:

$$a(i) = a(w-1-i), i = 0, 1, ..r-1 \tag{3}$$

$$a(r) = S - 2\sum_{i=0}^{r-1} a(i) \tag{4}$$

$$b(i) = b(w-1-i), i = 0, 1, ..r-1 \tag{5}$$

$$b(r) = S - 2\sum_{i=0}^{r-1} b(i) \tag{6}$$

The filtering process follows a simple iterative scheme: it starts with an initial value of horizontal and vertical filters. It optimizes one of them (a in this case) while the other is kept fixed (b_{in}). Once the first the *r-taps* version of the filter is obtained, it is reconstructed using Equations (3)–(6). Then this is used as input for the other filter processing. The Wiener filter process is represented in Figure 1.

This work provides hardware implementation of the Wiener Filter for AOMedia AV1 video coding. Also a possible high-speed implementation is provided. It is possible for it to be used for real-time data processing. This is true due to the high frame rates achieved by the hardware implementation. The next Section 2 details the architecture implementation of the filter. Section 3 details the results and discussion. Section 4 presents a real-time evaluation of the filter. Finally the conclusions are given in Section 5.

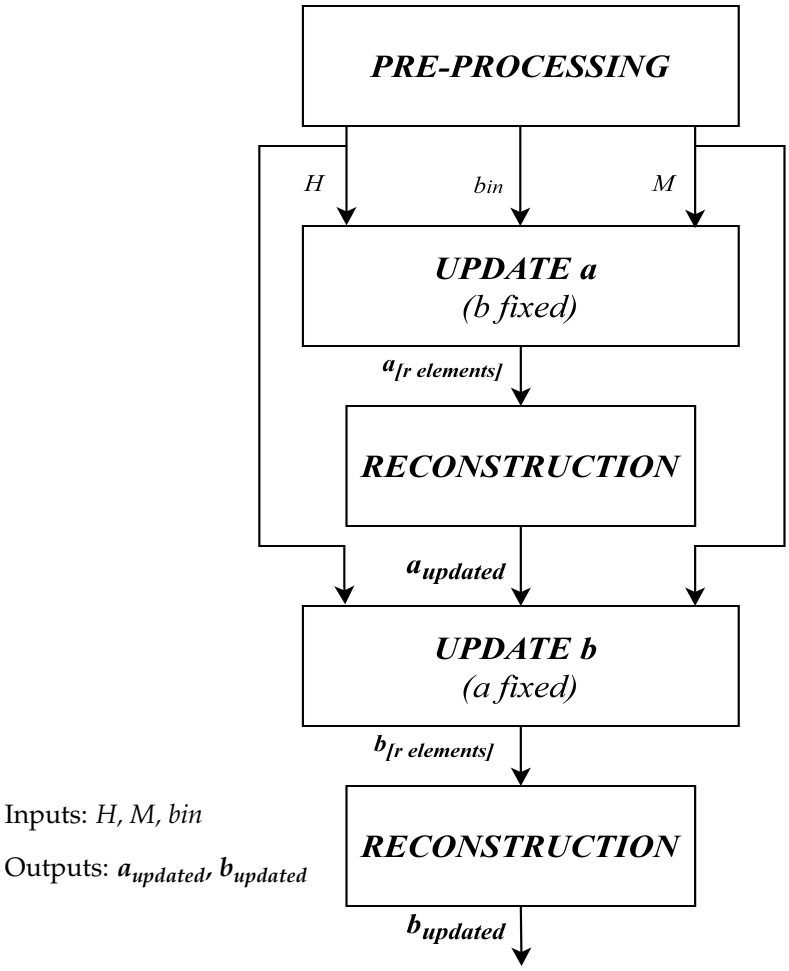


Figure 1. Wiener filter process.

2. Architectural Implementation

From an initial mathematical analysis of the algorithm described in [5] and looking at the implementation into the AV1 codec, it was possible to create an architecture that performs the same function as the codec. Thus, it obtains the same results. The inputs of the architecture are:

- The H_{ij} matrix, a single element of the H matrix, of size 7×7.
- The M matrix of size 7×7.
- The starting guess vector b_{in}, composed of 7 elements.

The outputs are the new couple of horizontal and vertical filters, represented by the vectors a and b, each comprising 7 elements. The whole architecture can be divided into two main blocks, referred to as *update a* and *update b*. Further, both blocks can be divided into different steps and eventually into different sub-blocks. As shown in Figure 1, the *update a* block receives the initial b vector and computes a 4×4 B matrix and a 4-element A vector using the following equations:

$$B = \sum_{i=0}^{w-1} \sum_{j=0}^{w-1} H_{ij} b(i) b(j) \tag{7}$$

$$A = \sum_{i=0}^{w-1} M_i b(i) \tag{8}$$

The output of the filter is obtained by solving a linear system of equations in which:

- Matrix B is the coefficients matrix
- Vector A is the vector of constant terms

The resulting solution of the system of equations represents the output values of the filter a. As the software model of AV1 gives $r = 3$, thus it is necessary to process these structures by an *Enforcement* block, which reduces the dimensions. For matrix B, the dimension is reduced to 3×3. The vector A is reduced to 3 elements. Thus, a proper dimensioned linear system of equations is obtained. To solve this system, the Gaussian elimination method is exploited. The Gaussian method consists of *Partial Pivoting*, *Forward Elimination* and *Back-Substitution* steps which are implemented by using blocks of *Partial Pivoting*, *Forward Elimination* and *Back-Substitution*, respectively. The result is the output vector X consisting of 3 elements. Finally, by applying the symmetry constraints, the updated a vector is reconstructed to the dimension w. Similarly, for the *update b* block, starting from the new a vector, b vector is obtained by following the same steps. The only difference is, instead of using a feedback approach in the computation method of martx B and vector A, a matrix storing mechanism is utilized. We can summarize the operations performed in the following equations:

$$B = \sum_{i=0}^{w-1} \sum_{j=0}^{w-1} H_{ij} a(i) a(j) \tag{9}$$

$$A = \sum_{i=0}^{w-1} M_i a(i) \tag{10}$$

Finally, applying the same constraints as for for a vector, the updated b vector is reconstructed. A more detailed presentation of the block that performs the mentioned operations is reported below:

- The *Enforcement* block compresses the inputs adapting them to the 3-dimensional linear system of equations. By using every component of A, the enforced output vector is computed as represented in Figure 2. The same approach has been used to process the B matrix, exploiting the same flow for every 16 components, reducing them to 9, i.e., 3×3. To be coherent with the C model, from now on, B matrix will be called A and vector A will be called b.

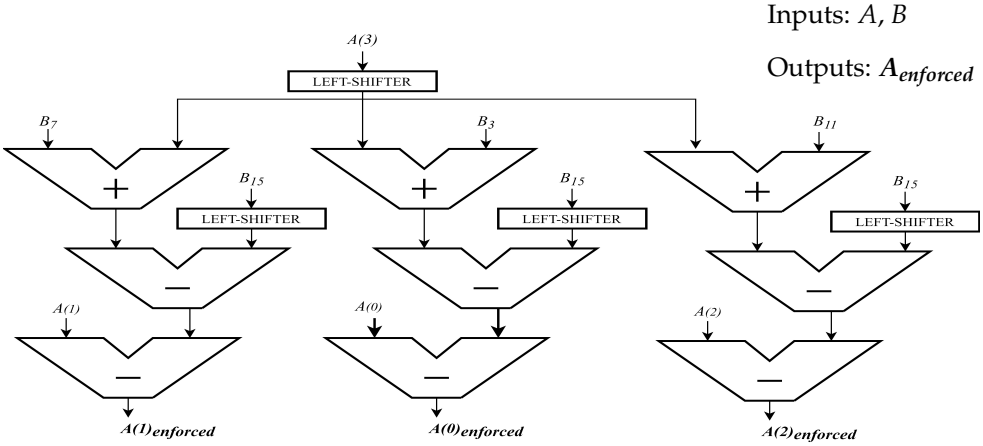


Figure 2. Enforcement architecture.

- The *Partial Pivoting* operation is the simplest block inside the whole architecture as it only involves interchanging rows of the matrix. Figure 3 represents its hardware architecture, where $k = 0$ indicates the first stage of *Partial Pivoting* , while $k = 1$ represents the second one. In particular, in the first stage, the absolute values of A_0, A_4 and A_8 are compared two by two, to find the largest one. Then by using the *Swap Rows* block, changing the position of b elements based on the outcome

of comparators. Similarly, in the second stage, the absolute values of A_5, A_9 are compared and eventually swapped to adapt the matrix to be solved with the Gaussian Elimination Method.

- *Forward Elimination* is the mathematical step of linear system resolution: it performs multiplication, division and subtraction to combine properly two rows and transforms the matrix as close as possible to an upper triangular form. Figure 4 reports the hardware implementation of the Forward Elimination operation for b vector.

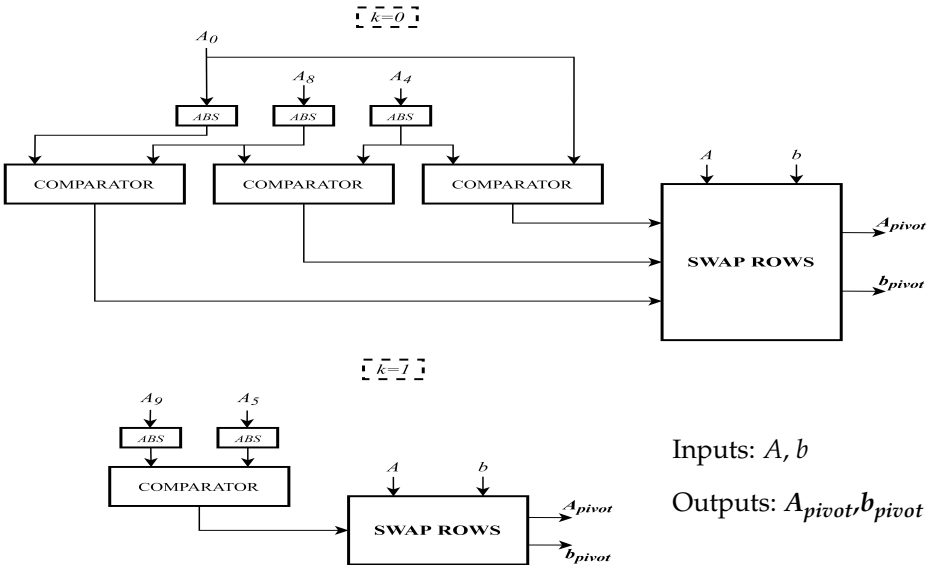
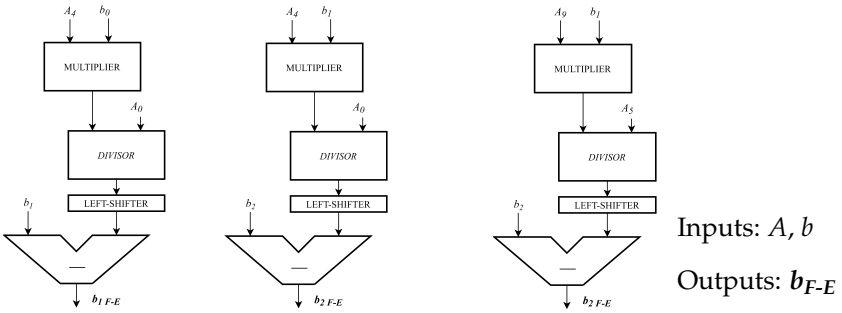


Figure 3. Partial Pivoting architecture.



(**a**) *First stage* (**b**) *Second stage*

Figure 4. Forward Elimination architecture.

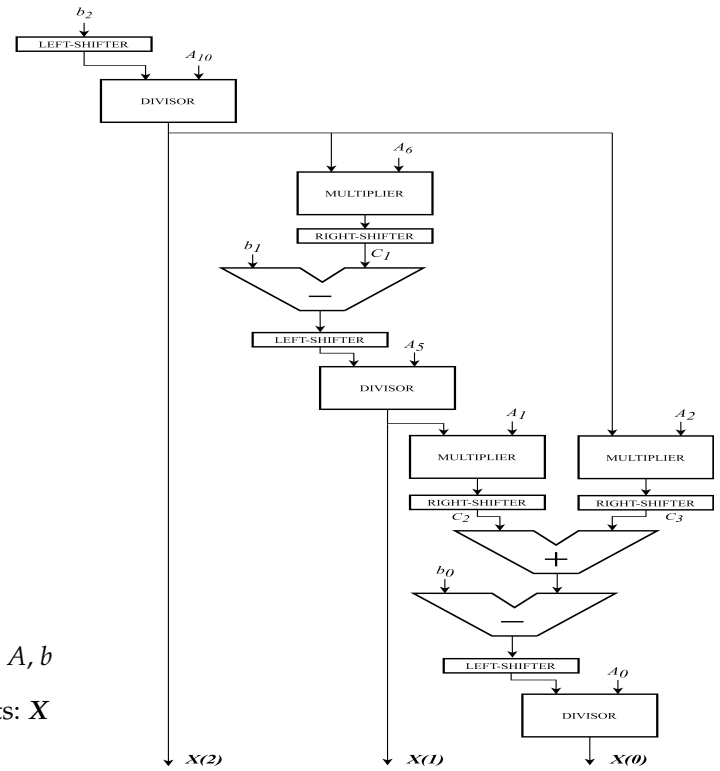
- What remains is to solve a linear system by using the *Back-Substitution and storing* block:

$$\begin{cases} A_0 \cdot a_0 + A_1 \cdot a_1 + A_2 \cdot a_2 = b_0 \\ A_5 \cdot a_1 + A_6 \cdot a_2 = b_1 \\ A_{10} \cdot a_2 = b_2 \end{cases}$$

This implemented architecture is shown in Figure 5. From a computational perspective, this block is complex because it involves several expensive operators like dividers and multipliers. Figure 6 shows the update a data path. This contains all the previous blocks combined inside. The critical path is displayed with an arrows going from Counter i to the adder on top left. This is because $H_{ij}xb_ixb_j$ involves cascaded multipliers.

- The *dividers* in the first basic architecture has been implemented in a purely combinatorial way. In particular, the one used here performs an n-bit division exploiting $2n$ consecutive operations of addition and subtraction.

The key idea of the presented work is to use the basic implementation as a starting point and optimize it. The architecture that will be presented in Section 2.1 is based on the same data path implementation, but each component block is designed differently depending on the kind of optimization to reach. Finally, each architecture contains a specific FSM.



Inputs: A, b

Outputs: $\boldsymbol{X}$

Figure 5. Back-Substitution and Storing architecture.

2.1. High Speed Architecture

One of the main goals of modern architectures is to process data in the shortest possible time, which means working at high frequency. Additionally, along with this the throughput is an important parameter when matrix processing is involved. Thus, the architecture needs to be accelerated. For this acceleration, the following steps have been performed. Starting from the original architecture, different timing reports have been generated to identify the critical issues of the starting architecture. These reports were analyzed to identify the critical blocks limiting the speedup. Then possible improvements were found to resolve the limitations, thus reducing the clock time and increasing the maximum operating frequency and throughput eventually. This analysis pointed out the two critical points of the structure:

- The length of the combinational paths;
- The combinational dividers.

The first point is due to many operators present along different combinational paths, which means that the time needed to elaborate a single piece of data is very long. This slows down the clock period. The second point is due to the structure of the initial dividers, which performs a division between two 64-bit operands. This means, that each division consists of 128 operators of adders and subtractors along the same combinational path. This also slows down the clock period as well as effecting the throughput. Therefore, the first improvement is to insert pipeline registers to reduce the length of the combinational paths following the typical *Restoring division algorithm* [18,19], as reported in Figure 7.

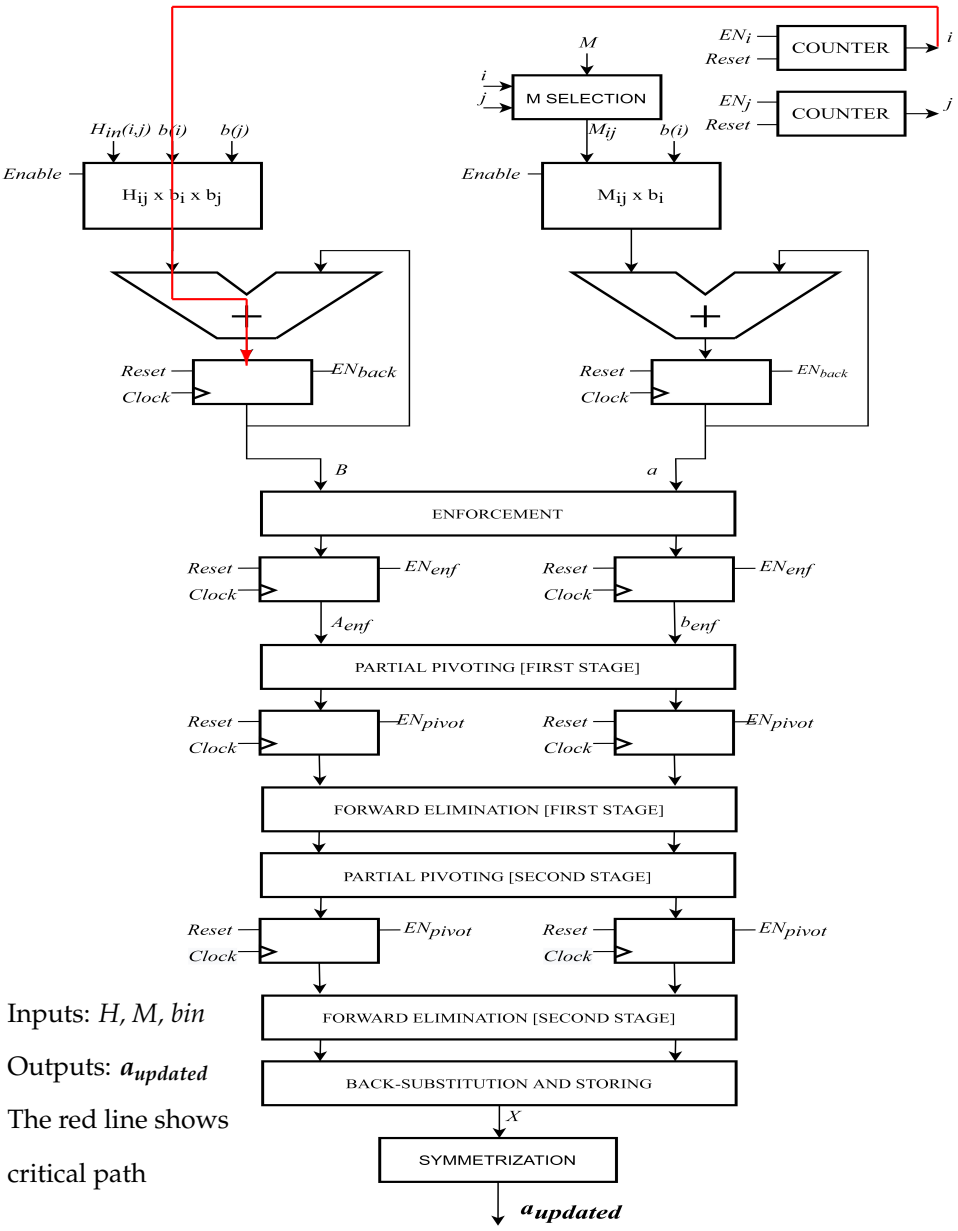


Figure 6. Updating a data path.

This helps to improve the clock period, in other words, it allows to operate at higher frequency. Pipelined registers are inserted in the *Back-Substitution*, *Forward Elimination* basic block, *update a* and *update b* top-level architectures. In this way, the length of the combinational paths has been reduced to a single operator block. The second improvement, instead, consists of replacing all the old combinational dividers with optimized restoring dividers [18,19]. The restoring dividers take the dividend and divisor, and store them in respective registers shown in Figure 7. They are shifted to the left and subtracted. The MSB of the result is complemented and shifted in the quotient register. The counter is decremented. When the counter reaches zero, the result is ready in the quotient register. The main feature of this new divider is that the maximum length of its internal combinational paths is drastically shorter than the old one. In particular, there is a single adder along the divider's critical path. Thus, by implementing these improvements, we obtain a final structure able to work at a higher frequency and providing a good throughput.

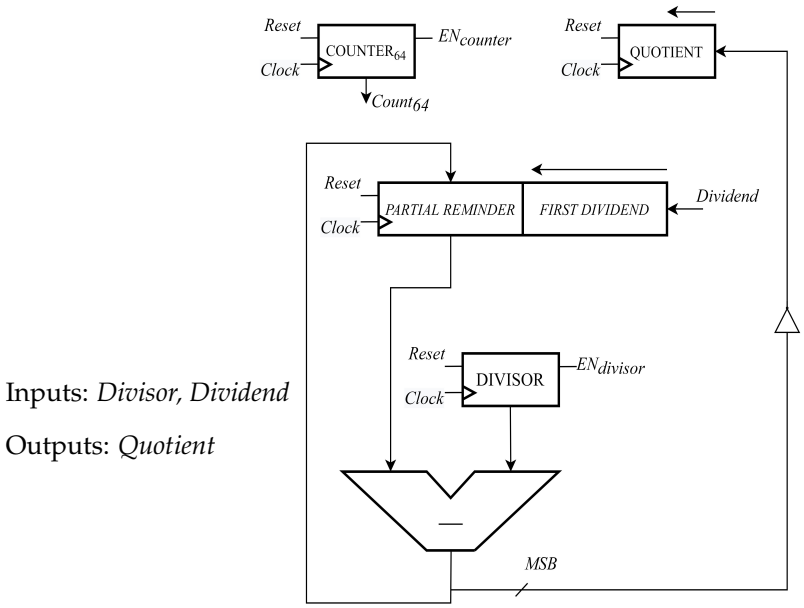


Figure 7. Optimized division data path.

3. Implementation Results

The behavioral simulation of the Wiener Filter is performed in modelsim6.2. The design implementation and synthesis are achieved using Synopsis Design Compiler and Innovus 20.11 with UMC NAND Gate 45nm technology. The solution is also implemented using VIVADO 16.1 for obtaining area results in terms of LUT and DSPs on an FPGA. Zync Zed Board (xc7z020clg484-1) is used as the target device. The video data are streamed to the ASIC implementation of the Wiener filter. The FPGA implementation is just provided to have a fair comparison of resources against state-of-the-art. The video is streamed to the ASIC by using a script to convert the video to pixel values, which is then passed as matrices. The video is streamed with the help of dual port static RAMs, which acts like a buffer. The video can be streamed to the FPGA by using VIVADO video library, this helps treat video as a sequence of frames. Each frame is considered as a matrix and is an input to the Wiener filter.

High Speed Architecture

The architecture has been validated against the AV1 software model. The PSNR values are reported by the authors in [5]. The results showed in the timing, area and power reports are reported here.

Table 1 displays the timing and area results for the dividers. Combinational divider in the basic architecture has a low frequency and higher area, while the Restoring Divider(RD) performs much better in both aspects. In caparison to the dividers in [19,20] our timing results are better by a factor of 100, while we suffer in terms of area. This is because our design is for 64-bit integer divider. The divider in [19] is a 12 $\times$ 12 array divider and [20] is a 16-bit divisor.

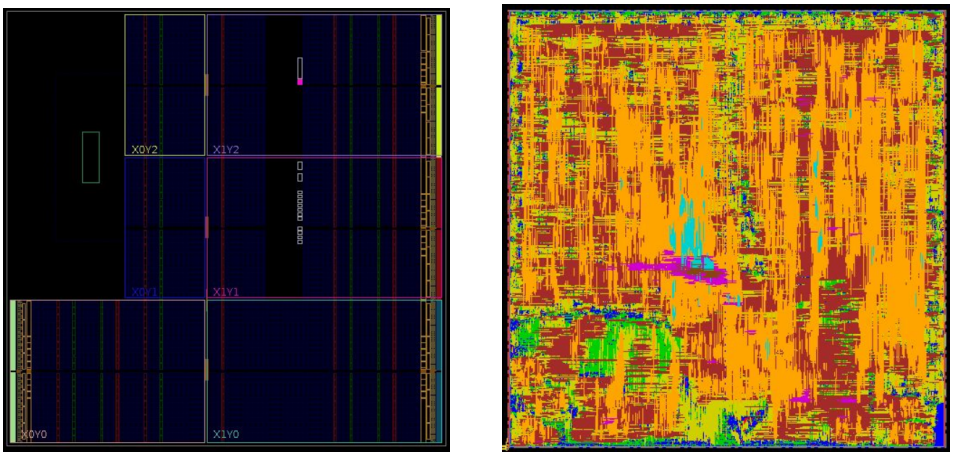
Table 1. Timing and Area results for the restoring divider.

Divider	Frequency [MHz]	Latency [us]	Area [um^2]
Combinational	15.15	29.91	87,258.107
Restoring	153.6	1.59	3426.346
RD1 [19]	-	577	740.44
RD2 [20]	-	-	2799

Table 2 displays the timing, power and area results for the Wiener filter (WF), our solution is called HSF (High Speed Final). The result is shown for post synthesis and post place-and-route. The die aspect ratio is set 1.0 × 0.6 with 5 um die margins. For the clock the fixCap and fixTran are kept true with 10ns provided as the period to satisfy by the tool. From the results shown, the clock is improved after post P-and-R optimization with a maximum frequency of 100 MHz. Power of the solution is 1011 mW. The area is much larger because of unrolling and parallel execution to achieve high performance. The layouts from the ASIC and FPGA implementation are displayed in Figure 8. The post P-and-R timing for technology corners, i.e., fast and slow are 9.95 ns and 24.128 ns, respectively. The fast corner consumes 1519 mW power while the slow one consumes 1196 mW. The fast one consumes more power working at a higher frequency.

Table 2. Area, Timing and Power results for the Wiener filter ASIC implementation.

HSF	Clock [ns]	Freq [MHz]	Area [um^2]	Power [mW]
Synthesis	13.29	75.24	1,988,927.227	12.816
P-and-R	10.03	99.7	1,966,505.5	1011



(**a**) VIVADO Layout (**b**) INNOVUS Layout

Figure 8. Layouts of the Wiener filter.

Table 3 shows the timing and area results of the HSF for the FPGA implementation. For latency, clock cycles information is extracted from the simulation. The latency is obtained by a product of clock cycles with the clock frequency information. The Wiener filter performs better by a factor of 1000 in terms of timing performance compared to state-of-the-art CPU- and FPGA-based solutions. The price is paid in terms of area. It consumes more DSPS, LUT and FFs than other FPGA based solutions. HSF is better in comparison to both software and hardware solutions for latency at the cost of 10 times higher area consumption. The LUT and DSP area results were obtained with the help of Xilinx VIVADO tool. This is only performed to have a reasonable comparison for the HSF architecture. This solution occupies 10 times more area then other solutions but consumes 1000 times less time. A good parameter for comparison is Area-Latency product (A-L), which is the obtained in terms of FPGA cells consumed. Since one FPGA cell contains two LUTs and two FF, so for this calculation half of the maximum from LUT and FF is considered as the number of cells. This is because each cell has two FF and two LUTs In terms of A-L our solution is two orders of magnitude better then the other solutions. Therefore, the area penalty is rectified here. Moreover, this solution is for a 3-times-higher resolution, as shown in Table 4 and 4 times more bit precision. Thus, high resource consumption is justifiable. The frequency determines the latency, a low frequency means high area-latency product, resulting in a poor solution. Whereas, a higher frequency decreases overall latency, thus decreasing the area-latency product. Hence, a better solution.

Table 3. Timing and Area results for the Wiener filter.

Arch	Device	Latency [ms]	LUT	FF	DSP	A-L [Cells/s]
HSF	FPGA	0.00159	565,181	27,423	166	0.449
WF [21]	CPU	406.7	-	-	-	-
WF [22]	FPGA	10	3912	4109	14	20.545
WF [23]	FPGA	4	-	-	-	-
WF [24]	FPGA	25	6721	6186	16	84.013
WF [25]	FPGA	-	8360	2385	13	-

Table 4. Video sequences fps.

Architecture	Resolution	*fps*	Msamples/Sec [1]
High-Speed Final	720 × 480	254.452	87.94
	720 × 576	211.864	87.86
	1280 × 720	95.328	87.85
	1920 × 1080	42.372	87.86
WF [22] (FPGA)	512 × 640	-	3.2
WF [23] (FPGA)	256 × 256	250	16.4
WF [25] (FPGA) [2]	32 × 32	4541	4.6

[1] The Msamples/sec reported are calculated by taking the product of fps and resolution. [2] The fps reported is extracted from the latency information and sample size given in the article.

4. Elaboration for a Real-Time Video Sequence

In order to better analyze the provided results, the effect of the explained improvements has been measured for a target real-time application. By evaluating throughput, intended as the number of samples processed per second, it is possible to define how many frames can be elaborated in real time for a specific target application. This analysis was conducted for high-speed architecture.

For a very precise idea, the best-known video formats were analyzed: SD and HD. For each of them, different resolutions were analyzed for approximated *fps* (frames per second) to obtain a good measure of the implementation speed. A good parameter for comparison is megasamples per second (Msamples/s), which is the product of frame size (height × width) and the reciprocal of latency. Thus, it also takes into account the resolution of the frames. Along with the fps, Msamples/s is also reported in Table 4. The fps of [25] is much better than our solution but at a very small resolution. In terms of Msamples/s, our solution outperforms all of the solutions in literature by a factor of 5. Results shown in Table 4 show that the proposed architecture can sustain very high frame rates both for SD and HD video resolution.

5. Conclusions

The presented work provides an algorithm-to-architecture mapping of the Wiener Filter for AOMedia AV1 video coding. To make it compatible with different sets of application, a possible high-speed implementation aimed at the speed increment is explained. Thus, it is possible to exploit a high-speed architecture in a very efficient way to improve the working frequency. In terms of throughput, the solution is much better then the state of the art. The design choice reported in this paper aims to create a special-purpose application coherent in terms of data, parallelism and operations with the C implementation of the Wiener filter [26]. Future works include the overall power and accuracy analysis of the implemented filter relative to the literature.

Author Contributions: Conceptualization, M.R.R., G.M. and M.M.; methodology, W.W. and M.M.; software, W.W., G.A. and S.D.P.; validation, W.W., G.A. and S.D.P.; formal analysis, W.W., G.A. and S.D.P.; investigation, W.W., G.A. and S.D.P.; resources, M.R.R., G.M. and M.M.; data curation, W.W., G.A. and S.D.P.; Writing—original draft preparation, W.W., G.A. and S.D.P.; Writing—review and editing, all authors; visualization, all authors; supervision, M.R.R., G.M. and M.M.; project administration, M.R.R., G.M. and M.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: The authors would like to thank Politecnico di Torino for providing the tools used for experimentation.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Mozilla Research. Available online: https://research.mozilla.org/av1-media-codecs/ (accessed on 18 December 2020)
2. Chen, Y.; Murherjee, D.; Han, J.; Grange, A.; Xu, Y.; Liu, Z.; Parker, S.; Chen, C.; Su, H.; Joshi, U. An Overview of Core Coding Tools in the AV1 Video Codec. In Proceedings of the Picture Coding Symposium (PCS), San Francisco, CA, USA, 24–27 June 2018; pp. 41–45.
3. Saldanha, M.; Corrêa, M.; Corrêa, G.; Palomino, D.; Porto, M.; Zatt, B.; Agostini, L. An Overview of Dedicated Hardware Designs for State-of-the-Art AV1 and H. 266/VVC Video Codecs. In Proceedings of the 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Scotland, UK, 23–25 November 2020; pp. 1–4.
4. W3C Patent Policy. Available online: https://www.w3.org/Consortium/Patent-Policy-20040205/ (accessed on 11 January 2021)
5. Mukherjee, D.; Li, S.; Chen, Y.; Anis, A.; Parker, S.; Bankoski, J. A switchable loop-restoration with side-information framework for the emerging AV1 video codec. In Proceedings of the IEEE International Conference on Image Processing (ICIP), Beijing, China, 17–20 September 2017; pp. 265–269.
6. Goldstein, J.; Reed, I.; Scharf, L.; Tague, J. A low-complexity implementation of adaptive Wiener filters. In Proceedings of the Conference Record of the Thirty-First Asilomar Conference on Signals, Systems and Computers (Cat. No. 97CB36136), Pacific Grove, CA, USA, 2–5 November 1997; pp. 770–774.
7. Goldstein, J.; Reed, I.; Scharf, L. A multistage representation of the Wiener filter based on orthogonal projections. *IEEE Trans. Inf. Theory* **1998**, *44*, 2943–2959. [CrossRef]
8. De Campos, M.; Werner, S.; Apolinario, J. On an efficient implementation of the multistage Wiener filter through Householder reflections for DS-CDMA interference suppression. GLOBECOM'03. In Proceedings of the IEEE Global Telecommunications Conference (IEEE Cat. No. 03CH37489), San Francisco, CA, USA, 1–5 December 2003; pp. 2350–2354.
9. Dong, J.; Ling, N. An Iterative Method for Frame-Level Adaptive Wiener Interpolation Filters in Video Coding. In Proceedings of the IEEE Workshop on Signal Processing Systems Design and Implementation, Banff, AB, Canada, 2–4 October 2006; pp. 113–117.
10. Siekmann, M.; Bosse, S.; Schwarz, H.; Wiegand, T. Separable Wiener filter based adaptive in-loop filter for video coding. In Proceedings of the 28th Picture Coding Symposium, Nagoya, Japan, 7–10 December 2010; pp. 70–73.
11. Elnady, A.; Noureldin, A.; Liu, Y. Implementation of the Wiener Filter for Extracting Power Quality Disturbances. In Proceedings of the IEEE Power Electronics Specialists Conference, Orlando, FL, USA, 17–21 June 2007; pp. 1116–1120.
12. Pardede, H.; Ramli, K.; Suryanto, Y.; Hayati, N.; Presekal, A. Speech enhancement for secure communication using coupled spectral subtraction and Wiener filter. *Electronics* **2019**, *8*, 897. [CrossRef]
13. Wu, F.; Yang, W.; Xiao, L.; Zhu, J. Adaptive Wiener Filter and Natural Noise to Eliminate Adversarial Perturbation. *Electronics* **2020**, *9*, 1634. [CrossRef]
14. Musznicki, P.; Schanen, J.; Granjon, P.; Chrzan, P. The Wiener filter applied to EMI decomposition. *IEEE Trans. Power Electron.* **2008**, *23*, 3088–3093. [CrossRef]
15. Rajeswari, K.; Krishna, K.; Naveen, V.; Vamsidhar, A. Performance comparison of wiener filter and cls filter on 2d signals. In Proceedings of the Sixth International Conference on Information Technology: New Generations, Las Vegas, NV, USA, 27–29 April 2009; pp. 1244–1249.
16. Trambadia, S.; Dholakia, P. Design and analysis of an image restoration using wiener filter with a quality based hybrid algorithms. In Proceedings of the 2nd International Conference on Electronics and Communication Systems (ICECS), Coimbatore, India, 26–27 February 2015; pp. 1318–1323.
17. Chandra, GVP.; Yadav, S.; Krishna, A.; Kamaraju, M. TPerformance of wiener filter and adaptive filter for noise cancellation in real-time environment. *Int. J. Comput. Appl.* **2014**, *97*, 16–23.
18. TSutter, G.; Deschamps, J.; Bioul, G.; Boemo, E. Power aware dividers in FPGA. In Proceedings of the International Workshop on Power and Timing Modeling, Optimization and Simulation, Santorini, Greece, 15–17 September 2004; pp. 574–584.
19. Kim, S.; Swartzlander, E. Restoring divider design for quantum-dot cellular automata. In Proceedings of the 11th IEEE International Conference on Nanotechnology, Portland, OR, USA, 15–19 August 2011; pp. 1295–1300.

20. Venkatachalam, S.; Adams, E.; Ko, S. Design of approximate restoring dividers. In Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), Hokkaido, Japan, 26–29 May 2019; pp. 1–5.
21. Petkova, L.; Draganov, I. Noise Adaptive Wiener Filtering of Images. In Proceedings of the 55th International Scientific Conference on Information, Communication and Energy Systems and Technologies (ICEST), Niš, Serbia, 10–12 September 2020; pp. 177–180.
22. Malarvizhi, S.; Kayalvizhi, R.; Kumar, A.; Topkar, A. Raw Data Processing Using Modern Hardware for Inspection of Objects In X-Ray Baggage Inspection Systems. *IEEE Trans. Nucl. Sci.* **2021**, *68*, 1296–1303. [CrossRef]
23. Doner, T.; Gokcen, D. FPGA-based infrared image deblurring using angular position of IR detector. *Vis. Comput.* **2020**, *37*, 2039–2050. [CrossRef]
24. Neetu, A. Implementation of Wiener Filter on FPGA using Xilinx System Generator for Speech Enhancement. *Int. J. Sci. Innov. Res. Stud.* **2018**, *6*, 1–12.
25. Yasodai, A.; Ramprasad, A. Noise degradation system using Wiener filter and CORDIC based FFT/IFFT processor. *J. Cent. South Univ.* **2015**, *22*, 3849–3859. [CrossRef]
26. Alliance for Open Media Google Git. Available online: https://aomedia.googlesource.com/aom/ (accessed on 12 July 2020).