Efficient Real-Time Architectures and FPGA Implementations of Histogram-Based Median Filters for High Definition Videos

Anish Goel

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science (Electrical and Computer Engineering) at

Concordia University

Montreal, Quebec, Canada

October 2019

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: Anish Goel

Entitled: Efficient Real-Time Architectures and FPGA Implementations of Histogram-Based Median Filters for High Definition Videos

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Electrical and Computer Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

______________________ Dr. A. J. Al-Khalili	Chair and Examiner
______________________ Dr. C-Y. Su (MIAE)	Examiner, External to the Program
______________________ Dr. M. O. Ahmad	Thesis Supervisor
______________________ Dr. M. N. S. Swamy	Thesis Supervisor

Approved by ______________________________

Dr. Y. R. Shayan Chair of the Department

____

Dr. Amir Asif, Dean

Faculty of Engineering and Computer Science

______________ 2019

Abstract

Efficient Real-Time Architectures and FPGA Implementations of Histogram-Based Median Filters for High Definition Videos

Anish Goel

Digital filtering plays an important role in many signal processing applications. Filtering is performed to recover the original signal from its corrupted version. Median filter is a non-linear digital filter that replaces a sample in a given window by the median value of the samples in the window. For images corrupted with impulse noise, median filter provides a very high quality of filtered images. Several modifications of median filters have been proposed and implemented to achieve high image quality compared to that provided by conventional median filters. When these filters are implemented on hardware platforms such as FPGAs, the performance parameters, namely, the area, power and operating frequency should be taken into consideration in addition to the quality of the filtered image. Therefore, efficient implementation of median filters on FPGAs for image and video processing algorithms has been a topic of much interest.

The existing hardware-based median filters for high definition video formats do not always satisfy the real-time throughput requirements or are inefficient with respect to hardware performance parameters, such as the area and frequency. This is due to the fact that most of the existing techniques use sorting-based median calculation, which results in a low hardware performance. In this thesis, architectures that use histogram-based median computation, which is a non-sorting-based operation, are designed with a view of efficient hardware implementation. This is carried out in two parts. We design and implement efficient architectures that satisfy the real-time throughput requirements of full high definition (FHD) videos in the first part and that of ultra high definition (UHD) videos in the second part.

In the first part, an efficient real-time histogram-based median filter that uses the concept of bit-plane-slicing and adaptive switching median filter (ASMF) is designed and implemented on FPGAs. We term this architecture as hybrid architecture for median filtering (HAMF). The proposed HAMF computes an approximate median, since it uses only the most significant B-bits of the pixel values for median calculation. As a result, the algorithmic level implementation of the proposed HAMF results in a slight degradation in the filtered image quality compared to that provided by ASMF. The proposed HAMF provides a significant improvement over ASMF in terms of the area and operating frequency, when implemented on different generation FPGAs. Analysis of the different parameters, such as the number of bit-planes used in the computation of the median and the number of pipelining stages, is carried out to study the trade-off between the quality of the filtered image and hardware performance.

Although the FPGA implementation of the proposed HAMF provides a very high operating frequency, the quality of the images filtered by its algorithmic level implementation decreases with increasing window size and noise density. This filter may be suitable for applications that require FHD filtering with cost constraints, but not for applications where the output image quality is as important as the hardware performance. Hence, in the second part, we design an efficient and real-time architecture of the hierarchical histogram-based median filter (HHMF). The proposed architecture is designed using a full synchronous pipeline, a synchronous accumulate-and-compare unit, and is scalable. The FPGA implementation of the proposed architecture of HHMF can perform real-time filtering of 4K and 8K UHD videos. The quality of the image filtered by HHMF is not compromised as in the case of HAMF, since HHMF uses all the bit-planes and computes the actual median. Although the FPGA implementation of HHMF results in more area utilization, the proposed implementation is more economical than a GPU-based HHMF implementation and provides a better throughput.

Acknowledgements

First, I would like to thank my supervisors Dr. M. O. Ahmad and Dr. M. N. S. Swamy for providing me with constant guidance and support during my studies. It was a pleasure to know them and their enthusiasm towards the widespread domains of electrical and computer engineering. It would not have been possible to purse my master's degree without the financial support provided by them for which I am very thankful to them.

I am also thankful to professors under whom I completed the courses required towards my degree. Specifically, Dr. W. P. Zhu for courses on signal and image processing, and Dr. A. J. Al-Khalili & Dr. O. A. Mohamed for courses on digital design and hardware verification. My thesis is a combination of "Digital Design" and "Image Processing" which would have not been possible without the in-depth knowledge of these courses. A special thanks to Mr. Tadeusz Obuchowicz for providing access to the VLSI CAD tools on university server.

I am thankful to my lab-mates Abdallah, Alireza, Omid, Abdel Rehman and Mohamed Reza who always maintained a friendly and peaceful environment in the lab. I would like to thank the administration of the university for providing such a wonderful and well-equipped laboratory for the purpose of study and research.

Finally, I am deeply indebted to my son *Samar* and my wife *Priyanka* who sacrificed their family time and allowed me to work upon my thesis. Not only this thesis, but every successful step of my life was possible due to the blessings of my Father and my Late Mother.

To my loving

Late Mother

Table of Contents

List of Figures

List of Tables

List of Acronyms

1D	One Dimensional
2D	Two Dimensional
AC	Accumulate-and-compare
ASIC	Application Specific Integrated Circuit
ASMF	Adaptive Switching Median Filter
BRAM	Block Random Access Memory
CAD	Computer-Aided Design
CB	Combinational Block
CIPMF	Contextual Image Processing Median Filter
CMF	Conventional Median Filter
CMOS	Complementary Metal Oxide Semiconductor
CPU	Central Processing Unit
CS	Compare-and-swap
FF	Flip-Flop
FHD	Full High Definition
FPGA	Field Programmable Gate Array
fps	frames per second
GPU	Graphical Processing Unit
HAMF	Hybrid Architecture for Median Filtering
HHMF	Hierarchical Histogram-based Median Filter
HMH	Histogram of Median High
HML	Histogram of Median Low
ICs	Integrated Circuits
LSB	Least Significant Bit

LUT	Look-Up Table
MPSoC	Multi-Processor System on Chip
MSB	Most Significant Bit
MSE	Mean Square Error
ppc	Pixels per clock
PSNR	Peak Signal to Noise Ratio
RMSE	Root Mean Square Error
ROM	Read-only Memory
RTL	Register Transfer Level
SoC	System on Chip
SSIM	Structural Similarity Index
UHD	Ultra High Definition
VHDL	Very high speed integrate circuit Hardware Description Language
VLSI	Very Large-Scale Integration

List of Symbols

a	Minimum intensity of a pixel
b	Maximum intensity of a pixel
B	Number of the most significant bits
B_{MH}	Count of bin MH
C_i	Constant to avoid instability
C_l	Compare value for ML
C_h	Compare value for MH
δ	Delay of a combinational block
f_{max}	Maximum operating frequency
$f(x_{ij})$	Intensity of pixel x_{ij}
F_s	Frame size
I	Original Image
I'	Filtered Image
K	Number of values equal to MH
L	Pipeline latency
m	Number of Rows
n	Number of columns
M	Median value
ML	Lower nibble of median
MH	Higher nibble of median
v	Number of cores
N	Number of elements in a window
N_p	Number of padded pixels
p	Noise density

p_a	Probability of *a*
p_b	Probability of *b*
P	Position of an adder
P_n	Number of pixels to be processed
Q	Number of cascaded blocks
r	Position of a bit
R	Dynamic range of pixel values
σ_i	Standard deviation
T	Throughput
μ_i	Mean intensity
W	Number of bits
W_s	Window size
$x[i, j]$	Set of surrounding pixels
y_{ij}	Intensity of pixel at location *ij* in original image
$y[i,j]$	Median of pixel at location at (*i,j*)
x_{ij}	Pixel at location *i,j* of a noisy image

Chapter 1

Introduction

1.1 Median Filtering

Filtering is one of the most important topics in signal processing applications and is used in recovering the original signal from a corrupted signal. Median filter is a non-linear filter used for smoothing 1D and 2D signals and is based on the median operation. Median filter replaces a noisy value in a set of given values by the median of all the values within this set. In digital images, noise is a random variation of the brightness or color information. Median filter is used to remove such random variations and it is very effective in removing impulse noise in images. When a median filter is applied to a window of a noisy image, it replaces the *center pixel* of the window by the median of the pixels within. For filtering images affected by salt and pepper noise, a type of impulsive noise, median filter has proved to be very effective in providing high quality filtered images, even for a high density of salt and pepper noise. Since median filter is based on a windowing operation, its window size W_s is an important parameter, which dictates the number of input values to be used, and hence, the number of operations required for the computation of a median value. In image processing, the window size W_s of a median filter is always chosen as odd so that the pixel to be filtered is surrounded by equal number of pixels on all sides. As a result, each window is a square window and its total size in terms of the number of pixels is W_s^2. Median filter with higher window sizes may provide a higher quality filtered image. However, the median operation is costly, and its complexity increases with increasing window size.

The most common method of filtering an image using a median filter is to consider successive overlapping windows of the image and apply the median operation on each of the windows to generate the output image. As a result, each pixel of the input image is replaced by the median of the pixel values of the elements of the window corresponding to that pixel. Such a type of median filter is called conventional median filter (CMF). Although CMF is effective in removing noise from an image, it also filters the edges in the image, resulting in a smooth output image. In a case where all pixels of the input image are not affected by noise, such a smoothing effect results in a low quality output image. Hence, for images affected by salt and pepper noise, an adaptive switching median filter (ASMF) is used. This type of filter replaces the *center pixel* of the input window by the median of the window, only if an impulse noise is detected in the *center pixel*. If an impulse noise is not detected, the input pixel value is passed on to the output. As a result, the non-noisy pixels remain unaltered and the edges in the input image are not smoothened.

Median filters have been implemented on various platforms to provide high quality filtered images and high performance with respect to the speed. However, due to increase in the demand of real-time image and video processing, many hardware-based architectures, such as those in [1] and [2], have recently been proposed and implemented for achieving high processing speeds. Recent trends in increasing frame resolutions from full high definition (FHD) to ultra high definition (UHD) require high speed architectures for image and video processing algorithms such as for filtering, edge detection, segmentation and morphological operations. Among these, filtering is not only an important image pre-processing step, but a stand-alone operation for images corrupted by noise [3].

An image may be affected by different types of noises such as Gaussian, Poisson, speckle noise or impulsive noise. The possible sources of impulsive noise in images are defects in the sensing or capturing device, memory corruptions and shot noise [4]. These sources of noise affect an image in such a way that some of the pixel values are set to a minimum value whereas some others to maximum value, a phenomenon that characterizes salt and pepper noise [5]. To recover the original image from an image corrupted by salt and pepper noise, spatial-domain filtering is typically used, in which the best match for each corrupted pixel is calculated. Finding a pixel from the neighborhood or a close approximation based on the neighborhood pixels to replace the corrupted pixel is a typical approach in spatial-domain image filtering. In this approach a 2D window surrounding the noisy pixel is used for processing. Although algorithms and techniques have tried to solve the problem of retrieving the original pixels from the corrupted pixels, it may be interesting to note that the actual value of the corrupted pixels may never be known. This is supported by the fact that none of the existing algorithms of median filtering provides infinite peak signal-to-noise ratio (PSNR) between the original image and the image filtered from its noisy version, even in the presence of low density noise.

Median filter provides better results compared to that provided by many other filters for images corrupted with impulsive noise [6]. Median filter is based on the median operation, where a value M is selected from a set of N values such that there are $(N\text{-}1)/2$ values greater than M and $(N\text{-}1)/2$ values less than M, assuming that N is odd. A simple way to calculate M is to sort the N input values in ascending order and select the element at position $[(N\text{-}1)/2] + 1$ as the median, which is the basic sorting-based median filtering algorithm [7]. A sorting-based median filter is typically implemented using the classical sorting network [8], which has multiple compare-and-swap (CS) units, as the ones used in [9]. This type of implementation using sorting network is very costly in terms of the hardware resources, since the number of cascaded stages of CS unit increase with increasing window size [10]. These cascaded stages also reduce the speed of the filtering operation, resulting in a low throughput.

Fig. 1.1 shows the typical steps of median filtering of an image of size $m \times n$. As mentioned earlier, the most important parameter of a median filter is its window size W_s. In Fig.1.1 W_s is considered to be 5. To start the filtering operation, the input image is padded on all sides with border pixels. The width of the border N_p resulting from the pixels padded to the input image is given by

$$N_p = (W_s - 1)/2 \tag{1.1}$$

assuming that W_s is odd.

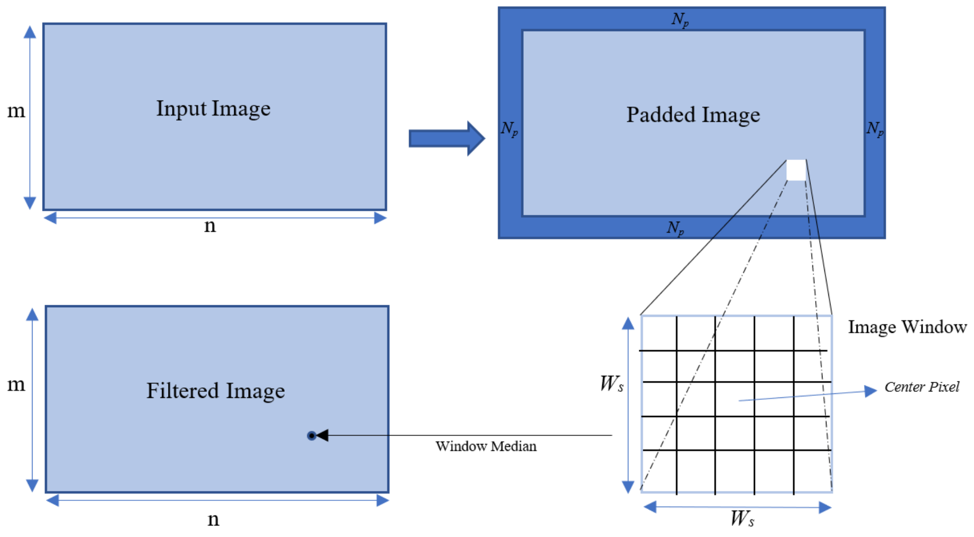


Figure 1.1. Median filtering steps.

Starting from the top left corner of the padded image, overlapping windows are successively processed in a raster pattern. Processing involves calculating the median *y[i,j]* of the pixel values in the window considered using

$$y[i,j] = median\{x[i,j] \in Ws\} \tag{1.2}$$

where *x [i, j]* is the set of pixels surrounding the pixel under consideration. The output image is generated by replacing each image pixel by the median calculated using Eq. (1.2). Since the window size W_s is odd, a pixel to be processed is surrounded by equal number of pixels on each side. Hence, the minimum value of W_s is 3, which forms a 3 × 3 window with the *center pixel* surrounded by a single pixel in each of the horizontal, vertical and diagonal directions.

1.2Literature Review

Several algorithms and implementations have been proposed and implemented on different platforms such as CPUs, GPUs, ASIC and FPGAs. In this section, we discuss many relevant algorithms, techniques and implementations by categorizing them into one of the five following categories. These categories are stated listed along with their main attributes.

1. Computationally complex schemes: Require a special platform such as a GPU or a parallel processing architecture for implementation.
2. ASIC implementations: Implemented as a dedicated IC using VLSI ASIC design flow.
3. Technique with approximate median computation: Implemented to provide approximate median values for optimizing hardware performance.
4. Sorting-based techniques: Use the conventional sorting-based median calculation.
5. Histogram-based techniques: Use histogram-based median calculation.

1.2.1 Computationally Complex Schemes

Computationally intensive median filtering techniques such, as in [11] and [12], have been proposed and implemented to provide high performance with respect to the quality of the filtered images. It is to be noted that a computationally complex technique imposes processing time restrictions on the platform on which the algorithm is implemented, and hence, the throughput of such filters in terms of the number of filtered **f**rames **p**er **s**econd (**fps**) is low. As a result, these algorithms are implemented on GPUs to improve the performance. For achieving high performance of the complex algorithms, hardware platforms such as FPGAs, or parallel processing architectures [13] - [15], are used for their implementations. Such implementations yield performance equivalent to that provided by the GPU implementations.

1.2.2 ASIC Implementations

Although architectural improvements focus on the performance of filters with respect to area, delay and power, the lowest level of abstraction of a digital system lies in the underlying VLSI technology. An ASIC implementation of an area efficient 1D median filter is presented in [16]. Another ASIC implementation, which is energy efficient and yields a high-throughput is implemented using 90-nm technology in [17] and can operate in GHz range. A modular design of the filter in [17] is presented in [18], which provides the highest ever reported operating frequency of over 2 GHz for median filters. These techniques use a basic building block of partial median computing unit to implement 1D median filter, which is based on the concept of bit-plane-slicing and is responsible for area and delay optimization. Although the ASIC implementations provide the best hardware performance, they are still limited with regard to issues like time-to-market, cost, flexibility and yield.

1.2.3 Technique with Approximate Median Computation

When the subjective quality of an image is of concern, there is no significant loss of visual information even if some of the pixels are not retrieved to their near-original values. As a result, a technique based on approximate arithmetic in [19], is implemented to optimize hardware performance parameters such as the area and power at minimal cost of image quality. Approximate arithmetic is typically used at the circuit level to calculate an approximate output [20]. Depending on the application, a drop in the image quality is acceptable in return to the hardware performance parameters.

1.2.4 Sorting-Based Techniques

Many modifications of the classical sorting network have been proposed and implemented to improve the hardware performance of median filters. Techniques presented in [21], [22] and

[23] use row-wise, column-wise and diagonal-wise sorting, whereas the technique presented in [24] uses a batcher's bitonic sort [8] to speed up the process. The latency of a median filtering architecture directly depends on the number of sorting stages that are cascaded in the architecture to calculate the median values. Although most of these architectures use pipelining, however, all the stages in a sorting network have the same latencies, and hence, a longer pipeline in the architecture will not increase the throughput [25].

1.2.5 Histogram-Based Techniques

To overcome the disadvantages of sorting-based algorithms, non-sorting histogram-based algorithms, such as [26], [27] and [28], have been developed for faster computation of median values. However, these algorithms are suitable for CPU or GPU implementations. As for an implementation on an FPGA, the work in [29] provides the first hardware implementation of a non-sorting-based median filtering using histogram-based operation. Another implementation of the architecture of [29], is proposed in [30] for larger window sizes. However, these implementations use read-only-memory (ROM) and process all bit-planes of an image, resulting in a low operating frequency.

Algorithms implemented on GPU for high performance, such as the ones in [31] and [32] are based on histogram calculation. Thus, histogram-based techniques are capable of providing architectures for real-time filtering of FHD and UHD videos. Hardware implementations of histogram-based median filter provides a low throughput as in [29], in view of the memory bins used for storing histogram values. On the other hand, hardware implementation of the conventional histogram-based technique is not feasible without the use of memory bins, since for a high window size, the resource requirement will exceed that available in FPGAs.

An alternate to the histogram-based median filtering is the hierarchical histogram median filter (HHMF) [33], which processes the upper half most-significant bits (MSB) of the input data first followed by processing the lower half least-significant bits (LSB), to obtain the complete median value in two steps. The hierarchical histogram-based median filtering algorithm presented in [33] is designed for GPUs and provides a performance in terms of speed which is superior to that of several other algorithms on the same platform. The hierarchical histogram-based median filtering is based on the same memory bin approach that is used in histogram-based median filter, the difference being that the number of bins is reduced in view of the reduction in the range of input values. This is a consequence of splitting the pixel values into two parts.

1.3 Real-Time Implementation Requirements of FHD and UHD Videos

The two most important parameters of a video are its frame resolution and the frame rate. The former specifies the size of each frame, whereas the latter is related to the smoothness of the video playback. In general, a higher resolution and a higher frame rate provide a superior quality video. A high definition video is a high-quality video with frame resolutions higher than that of its predecessor, the standard definition video. Any video having more than 480 vertical lines in a frame is considered to be a high definition video**.** At present, FHD and UHD are the most commonly used standardized video formats. The typical frame rates of FHD videos as well as that of UHD videos are 30 fps and 60 fps. The typical frame resolutions of FHD and UHD videos are as follows:

a. FHD, 2K = 2048 × 1080
b. UHD, 4K = 3840 × 2160
c. UHD, 8K = 7680 × 4320

The pixels in a frame are placed in the same way as the elements are placed in a rectangular matrix. The frame resolution is the number of pixels present horizontally in the frame multiplied by the number of pixels present vertically in the frame, which gives the total size of the frame in terms of the number of pixels. A higher number of pixels results in more resolution, providing a better quality image. The frame resolutions of FHD and UHD videos are also referred by numbers 2K, 4K and 8K, since these numbers approximately equal to the number of pixels in the horizontal direction of a frame. (Note: 1K binary is equivalent to 1024 in decimal)

In order to satisfy the real-time processing requirements of the FHD and UHD videos, a system shall process P_N number of pixels each second. Table 1.1 shows the values of P_N for the two high definition formats considered. It is seen from this table that around 2 billion pixels need to be processed each second for the real-time processing of the highest resolution of 8K at the rate of 60 fps, making it a challenging task to design a real-time architecture to accomplish it. As a result, only ASIC implementations such as the ones provided in [18] can satisfy the real-time requirements of such a system. Although there exists an implementation of contextual image processing architecture of median filter (CIPMF) presented in [34] that can satisfy the real-time requirements of 4K UHD video at rate of 60 fps, it is limited to window size $W_s = 3$ and only 4K resolution. As the CIPMF architecture uses sorting-based method, its throughput will decrease with increasing window size, making it unsuitable for UHD applications.

Table 1.1. Number of pixels to be processed for real-time processing of FHD and UHD videos

Standard	**Resolution**	***P_N* @ 30 fps**	***P_N* @ 60 fps**
FHD, 2K	1920 × 1080	62,208,000	124,416,000
UHD, 4K	3840 × 2160	248,832,000	497,664,000
UHD, 8K	7680 × 4320	995,328,000	1,990,656,000

1.4 Motivation and Objectives of the Thesis

Most of the existing techniques for the design of hardware architecture for median filtering focus on enhancing the hardware performance by modifying an existing algorithm or by adding another layer of complexity to optimize some of the hardware performance parameters. This optimization is achieved at the expense of some of the other performance parameters. Our objective in this thesis is to investigate the problem focusing primarily on bit-plane-slicing and architecture design and implementation of the histogram-based median filtering operations.

Since the sorting-based techniques do not always satisfy requirements of the real-time filtering of high definition video formats or are inefficient, we consider the histogram-based median filtering techniques as the basis for achieving real-time performance for the high definition video formats considered. While providing high throughput using histogram-based techniques, we also analyze the effect of the implementations of the proposed hardware architectures on other performance parameters.

We present two different architectures for high-speed median filtering along with their FPGA implementations. The proposed median filter designs are based on the histogram technique and implemented with a pipelined architecture to increase their maximum operating frequencies. In the first design, the speed of the median filter is enhanced by designing a hardware that processes a limited number of bits rather than all the bits of the pixels for calculating approximate median values. This design provides real-time median filtering of FHD videos at the cost of minimal image quality degradation. Since the proposed technique does not utilize all bit-planes of an image or video frame, the resulting hardware is simplified resulting in a reduced area utilization as an added advantage.

In the second part of this thesis, we design an efficient architecture for the hierarchical histogram-based median filter, which satisfies the real-time pixel clock requirement of the

UHD videos when implemented on Xilinx UltraSclae+ MPSoC. The proposed architecture ensures that there is no degradation in the output image quality by calculating the exact median values. This is the first hardware implementation of the hierarchical histogram-based median filter and provides a performance superior to that provided by two of the NVIDIA GPUs in terms of the throughput.

The main contributions of the work presented in this thesis can be summarized as follows:

1. Design of histogram-based median filters and FPGA implementations with throughput as the primary criteria of optimization.
2. Combination of multiple techniques to formulate a hybrid architecture for median filtering to provide a high throughput with minimal effect to the image quality.
3. Analysis of the effect of pipeline latency on the hardware performance and that of the number of bit-planes on the quality of filtered images.
4. Design of an efficient hardware architecture for the hierarchical histogram-based median filter to provide a high throughput implementation on FPGAs and MPSoCs.
5. Analysis of the implementation of the proposed hierarchical histogram-based median filter architecture on parameters such as area and power with different window sizes.
6. Analysis of throughput of the implementations of the proposed median filter architectures with respect to real-time requirements of FHD and UHD videos and their comparison with existing implementations.

1.5 Organization of the Thesis

In Chapter 2, a brief review of the concepts that characterize the various architectures of sorting and non-sorting based median filters is carried out. The concept of bit-plane-slicing and the decision-based median filtering are also explained in this chapter and relevant examples are given. The performance parameters used in this work to evaluate the results are presented in detail. Chapter 3 presents the design and implementation of the proposed histogram-based hybrid architecture for median filtering (HAMF). The quality of the images filtered with the algorithmic level implementation of the proposed HAMF, using the PSNR and SSIM metrics is analyzed. An analysis to examine the effect of the number of bit-planes on the quality of filtered image and to determine the effect of the number of pipeline stages on the area is also carried out in this chapter. In Chapter 4, a hierarchical histogram-based median filtering (HHMF) algorithm is first introduced, and then a hardware architecture and its FPGA implementations are presented. Results of implementation for different window sizes are presented and compared with that of HHMF implemented on a GPU platform. Finally, some concluding remarks on the investigation carried out in this thesis are made in Chapter 5.

Chapter 2

Background Material

This chapter presents the details of the sorting-based and non-sorting-based median filtering techniques in order to understand their underlying complexities. The concept of impulse noise in images and its removal is presented with some experimental results. An analysis of the effect of noise adaptive switching median filter on the quality of a filtered image is also presented in this chapter. The concept of bit-plane-slicing is discussed with an appropriate example. This chapter also presents the underlying concepts used for improving the performance of a hardware-based architecture using pipelined processing and reducing the combinational delay. The various parameters considered for the performance evaluation of the proposed filter architectures are also discussed.

2.1 Median Filtering Techniques

The median filtering algorithms used in image and signal processing are broadly classified into two categories, namely, sorting-based and non-sorting-based. The sorting-based median filters perform a sorting operation on the input elements in order to find the median value, whereas the non-sorting-based median filters are typically based on histogram operation for median calculation. Each of these techniques is discussed in detail in the following sub-sections.

2.1.1. Sorting-based Median Filter

A sorting based median filter is based on the bubble sort technique [35]. This type of filter compares every two consecutive elements in the input vector and swaps them, if value of the first element is greater than the value of the second. This type of filter consists of 2 phases – the odd phase and the even phase. In the odd phase, every odd indexed element is compared with the next even indexed element. In the even phase, every even indexed element is compared with the next odd indexed element. The filter structure is formed by deploying N cascaded stages of the odd and even phases placed alternately, where N (an odd value) is the number of elements in the input array. After the input vector is sorted, the element from the output vector at position $(N + 1)/2$, is selected as the median.

Fig. 2.1. shows an example of the median calculation for an input vector consisting of nine values, using the sorting-based median filter. The number of stages of compare and swap operation required in this example is 9 and the last stage generates the sorted vector. The value, 6, at position 5 of the output vector is the median of the input vector.

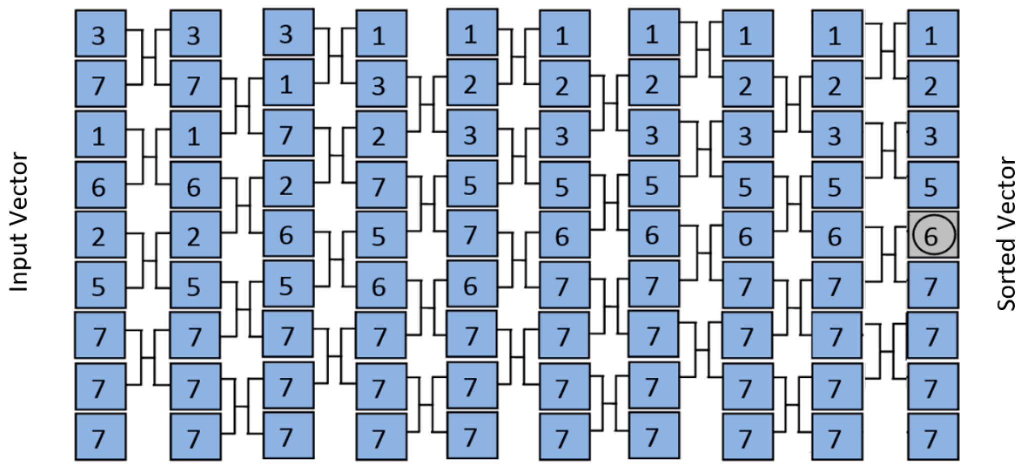
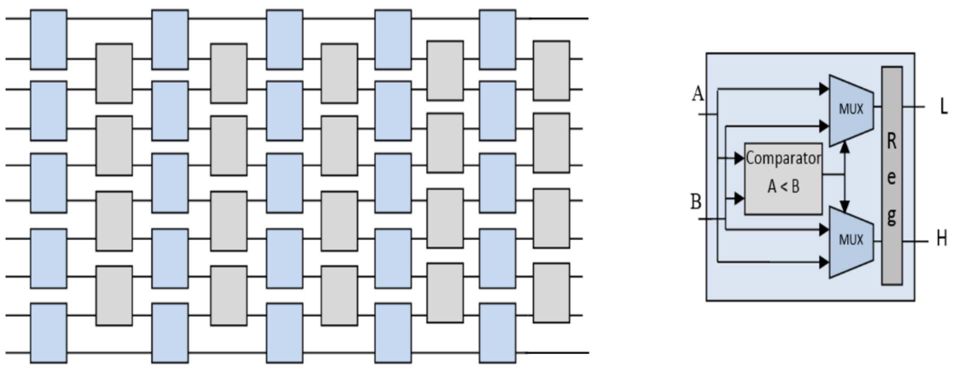


Figure 2.1 An example of sorting-based median calculation.

A hardware architecture of the sorting-based median filter can be designed using the sorting network as in [8], which uses multiple units of a processing element, called as the compare-and-swap (CS) unit. Each CS unit consists of a comparator and a multiplexer. The network sorts the input values and the median value is stored in the appropriate output register. Fig. 2.2 (a) shows an architecture of the sorting network used for sorting ten input elements. Each block in this sorting network is the CS unit, which is similar to the one used in [9]. The architecture of a CS unit is shown in Fig. 2.2 (b). The CS unit compares two input numbers, namely, A and B, and generates the lower value on output L and the higher value on output H.

A register is implemented inside the CS unit that serves as the pipeline register for the sorting network.



(a) (b)

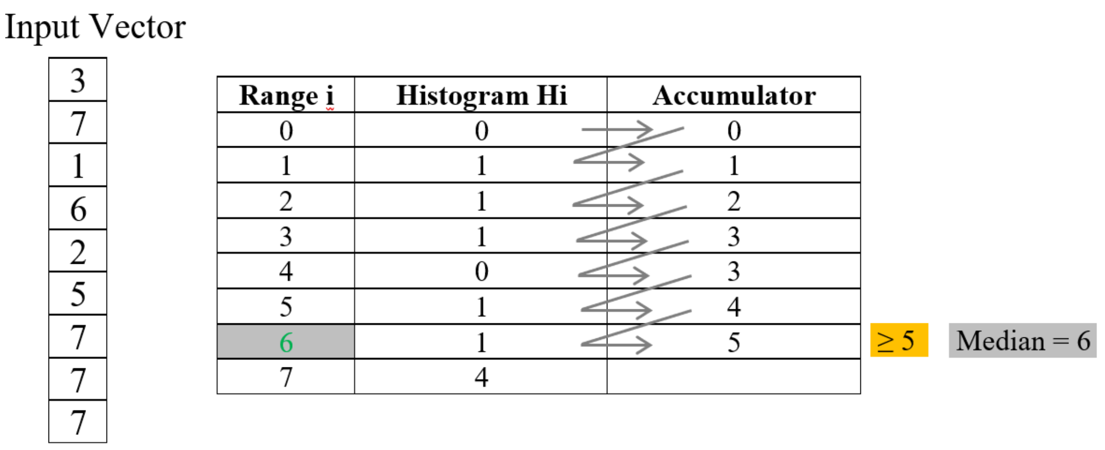
Figure 2.2 (a) Network for sorting 10 input values (b) A compare and swap unit.

Although this is one of the simplest hardware architectures for sorting-based median calculation, the number of stages in a sorting network increases linearly with number of input elements [8]. As a result, the pipeline latency of a sorting network for N input elements is N. The number of CS units required in a network to sort N input elements is $N\,[(N-1)/2]$, which increases rapidly with the value of N. The high latency and the high area contribute to the low efficiency of a sorting-based median filter.

2.1.2. Non-sorting-based Median Filter

To overcome the disadvantages of a sorting-based median filter architecture, non-sorting-based techniques, which are mainly histogram-based, are used for median calculation. The median of the input vector using histogram-based technique is calculated from the histogram of the input data. The histogram values H_0 to H_i of the input data in the range 0 to i is calculated by counting the number of occurrences of each value of the input data in this range. Next, the values H_0 to H_i are successively added until the result of the addition is greater than or equal to $\lceil N/2 \rceil$ (assuming N is odd). The index i of the last added H_i is the median.

Fig. 2.3 shows an example of the median calculation using the histogram-based operation for the same values that were considered in the example of the sorting-based median calculation. The count of occurrences of the values of the input data in the range (0 to 7) is calculated as the histogram values represented by H_i. These values are added successively in an accumulator. On the sixth iteration of the addition operation, the result in the accumulator is $\geq [N/2]$, which in this case is equal to 5. This results in a median value of 6.



Input Vector

3
7
1
6
2
5
7
7
7

Range i	Histogram Hi	Accumulator
0	0	0
1	1	1
2	1	2
3	1	3
4	0	3
5	1	4
6	1	5
7	4	

Figure 2.3 An example of histogram-based median calculation.

The steps to generate H_i and calculate the median are elaborated in Fig. 2.4. It is observed from this figure that the first step is to compare all the input elements with all the possible values that an element can take (0 to 7 in this case).

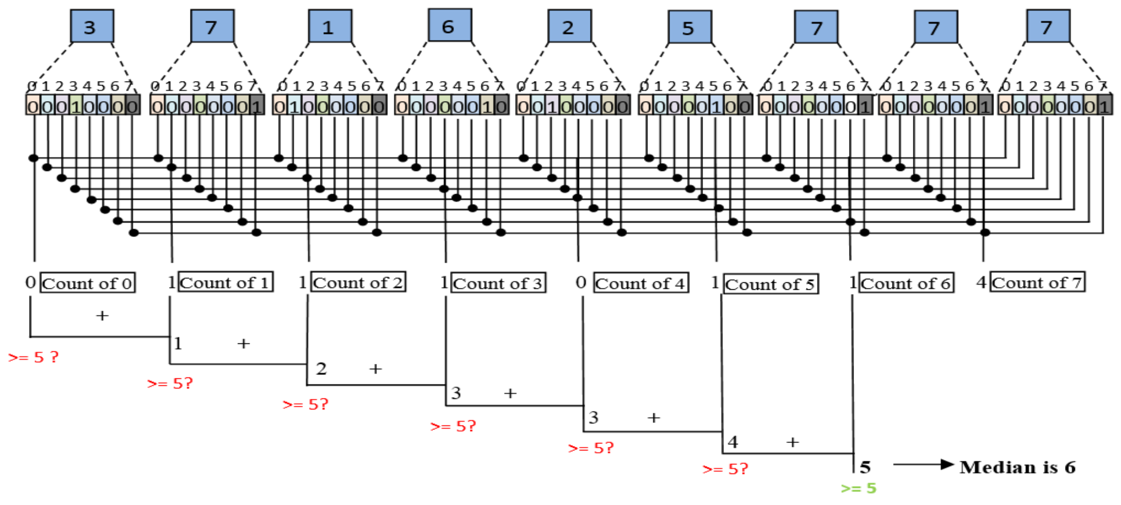


Figure 2.4 Histogram generation and median calculation.

Hence, for nine input values there are seventy-two comparators, since each of the nine input values are compared with the range values 0 to 7. This comparison produces an output of 9 sets of binary values with 8 bits in each set. Next, the respective bits (0 to 7) of each set are added together and the output of the addition generates the histogram of the input data. These histogram values are added successively and after each step, the result is compared with [N/2], which is equal to 5 in this case.

2.2 Impulse Noise Removal

This section presents experimental results of filtering images corrupted by salt and pepper noise. Images with different density of salt-and-pepper noise are filtered using a median filter with different window sizes to analyze the quality of the output image with respect to the noise density and window size. The smoothing effect, that occurs due to the filtering of the edge pixels in the image, is analyzed for different window sizes. Experimental results of images filtered using an adaptive switching median filter are presented to analyses its effectiveness in preserving edges.

2.2.1 Effect of Window Size and Noise Density

The value of a noisy pixel in an image is quite far from the range of values of the other pixels in a given window. Median filter is very effective in removing impulse noise in images [36] and replaces the noisy pixels by the median of the respective window as the best match for the noisy pixels. This *best match* depends on the size of the window and the density of noise. We first examine the effect of filtering an image with low noise density using a median filter. Fig. 2.5 (a) shows an original image and its corresponding noisy version having 5% salt and pepper noise density in Fig. 2.5 (b). Applying a median filter to the noisy image with window sizes 3, 5, 7 and 9 results in the filtered image shown in Figs. 2.5 (c), (d), (e) and (f), respectively.

It is observed from Fig. 2.5 (c - f) that the noisy pixels in Fig. 2.5 (b) are not visible in the filtered images. However, there is a difference between the filtered images in (c), (d), (e) and (f) that can be clearly seen as a result of the smoothing effect, which increases with increasing window size. The smoothing effect is observed due to the pixels of the edges in the image being replaced by the median value of the corresponding window. By subjective evaluation of these figures, we can say that amongst all the filtered image, the image in (c) is the closest approximation to the original image in (a).

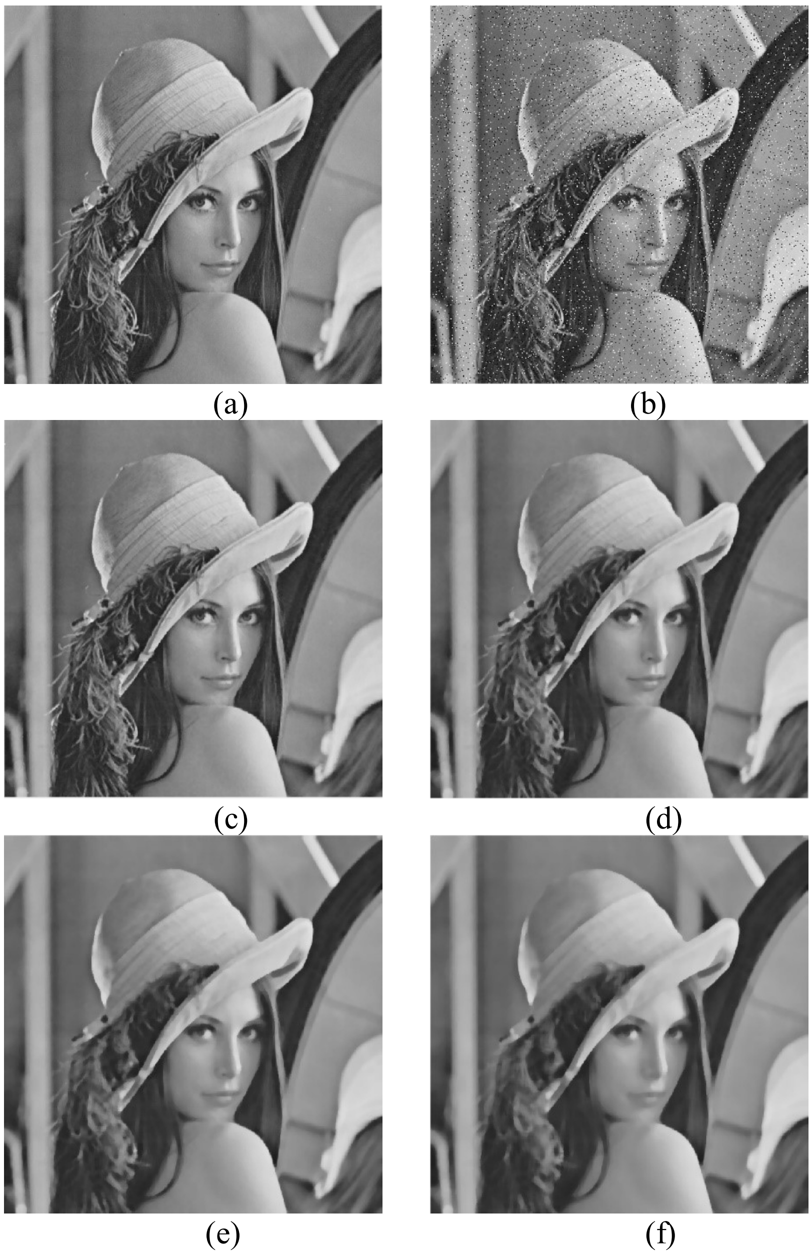


Figure 2.5 (a) Original Image (b) Image with 5% noise density (c-f) Images filtered using CMF with W_s = 3, 5, 7 and 9, repectively.

We now examine the effect of a more significant noise density (50%) on the filtered images with the same image and the same window sizes as in the previous experiment. Fig. 2.6 (a) shows an original image, its corrupted salt and pepper noisy version having 50% noise density in Fig. 2.6 (b) and the filtered images with W_s = 3, 5, 7 and 9 in Figs. (c), (d), (e) and (f), respectively.

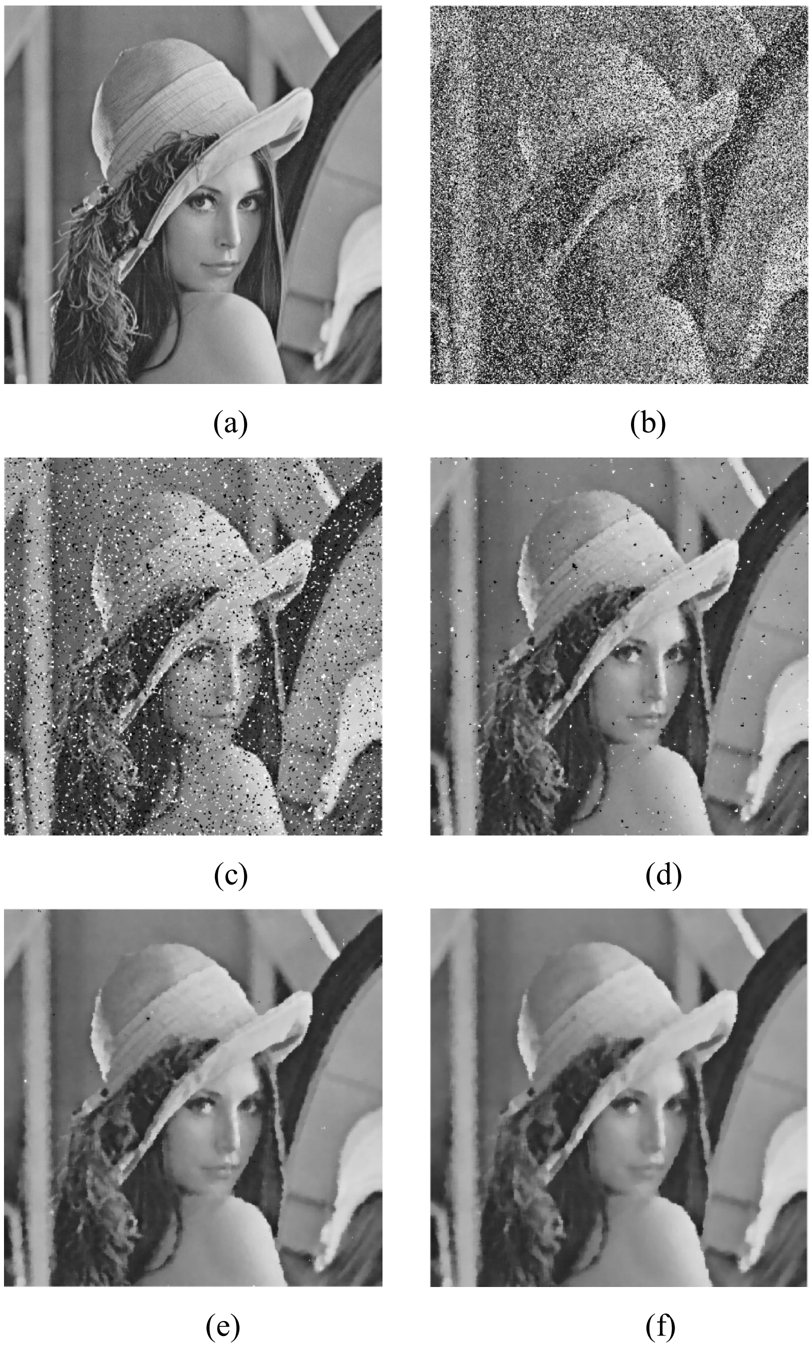


Figure 2.6 (a) Original Image (b) Image with 50% noise density. (c-f) Images filtered using CMF with W_s = 3, 5, 7 and 9, repectively.

It is seen from Fig. 2.6 that the noisy pixels are not completely filtered out using lower window sizes (3, 5 and 7). However, the image in Fig. 2.6 (f) filtered with W_s = 9 does not seem to have any noisy pixel when examined subjectively. As a result, we can say that the image in (f) is the best approximation to the original image in (a) amongst all the filtered images (c), (d), (e) and (f). From these experiments, we can conclude that higher window sizes are better suited images corrupted with high density noise. For an image corrupted with low density noise, a filter with low window size provides a better-quality filtered image. Median filters such as the ones in [37] and [38] are adaptive with respect to window size, which changes depending on the noise density in the corrupted image and provides improved results.

2.2.2 Switching Median Filter

For a noisy image, the intensity of the pixel x_{ij} at the location (i,j) is described by the probability density function $f(x_{ij})$ given by following equation.

$$f(x_{ij}) = \left\{ \begin{matrix} p_a & for\ x_{ij} = a \\ 1-p & for\ x_{ij} = y_{ij} \\ p_b & for\ x_{ij} = b \end{matrix} \right\} \tag{2.1}$$

where a is the minimum intensity, b is the maximum intensity, p_a is probability of generation of intensity a, p_b is the probability of generation of intensity b, noise density $p = p_a + p_b$, and y_{ij} is the intensity of pixel at location (i,j) in the corresponding uncorrupted image. For an efficient removal of the salt and pepper noise, the image pixels are filtered only if they are found to be equal to values a (minimum intensity) or b (maximum intensity) [39]. This type of a filter is commonly referred to as adaptive switching median filter (ASMF) or decision-based median filter. To eliminate the s*moothing effect*, a noise adaptive switching median filter performs filtering only if an impulse noise if detected in the pixel; otherwise, the input pixel is left un-filtered.

We analyze the effect of ASMF on an image corrupted by moderate density (25%) salt-and-pepper noise. Fig. 2.7 (a) shows an original image, its corresponding noisy version with 25% salt and pepper noise density in Fig. 2.7 (b) and images filtered using ASMF with $W_s = 3$, 5, 7 and 9 in Figs. 2.7 (c), (d), (e) and (f), respectively.

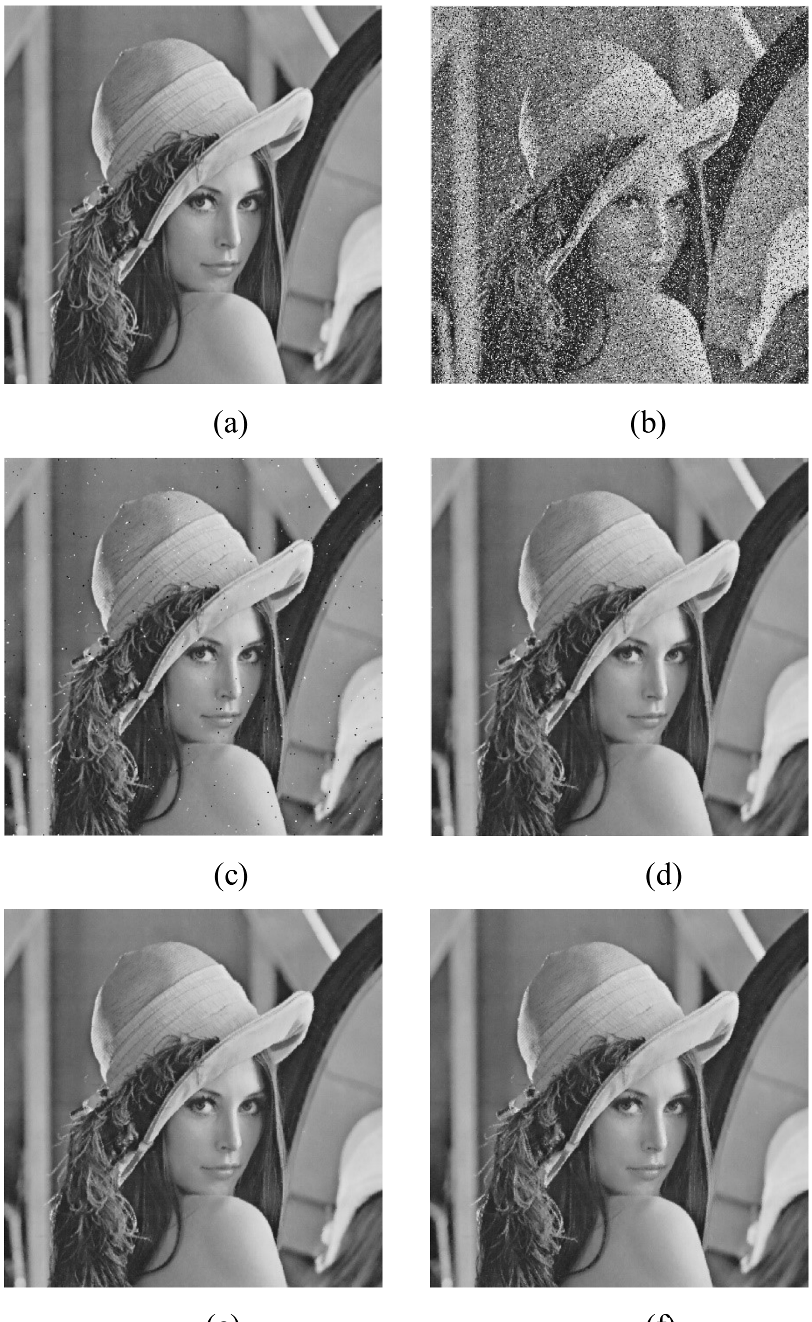


Figure 2.7 (a) Original Image. (b) Image with 25% noise density. (c-f) Images filtered using ASMF with $W_s = 3$, 5, 7 and 9, respectively.

It is observed from this figure that some of the noisy pixels are not filtered with W_s = 3; however, with W_s = 5, 7 and 9, the output images do not seem to have retained any noisy pixel and also retain the sharp edges of the original image. By adaptivly changing the window size and performing decision-based filtering, algorithms such as in [40] provide very high quality filtered images.

2.3 Bit Plane Processing in Images

Pixels in a digital image are represented by numbers in the binary format using a set of bits. For example, in a gray-scale image, pixels have an intensity that ranges from 0 to 255. The numbers in this range can be represented using an 8-bit binary number. Fig. 2.8 shows the 8 bit-planes of an image, each plane consisting of a single bit of all the pixels of the image. Representation of an image with one or more bits of the byte is called as bit-plane-slicing.

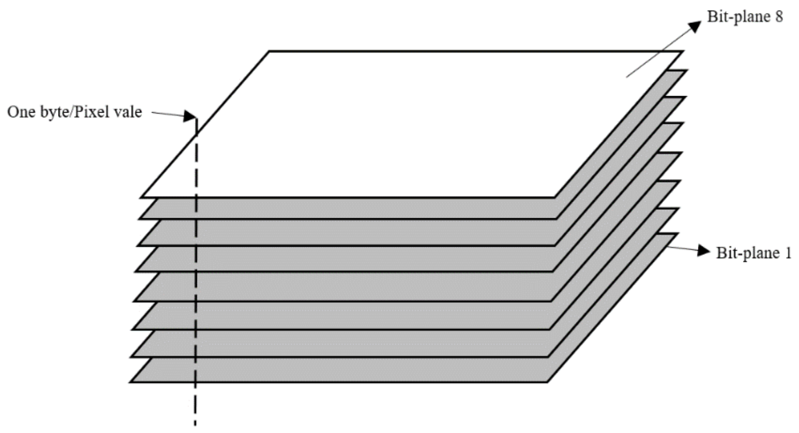


Figure 2.8 Bit-planes in a gray-scale image.

To understand the contribution of each bit plane in the overall visualization of an image, Fig. 2.9 shows a sample image along with 8 separate bit-planes that are shown as separate binary images. It is observed from this figure that the higher bit-planes contribute more to the visual data in an image that the lower bit-planes do. The lower bit-planes add small details to an image, which may not be visible during a subjective examination.

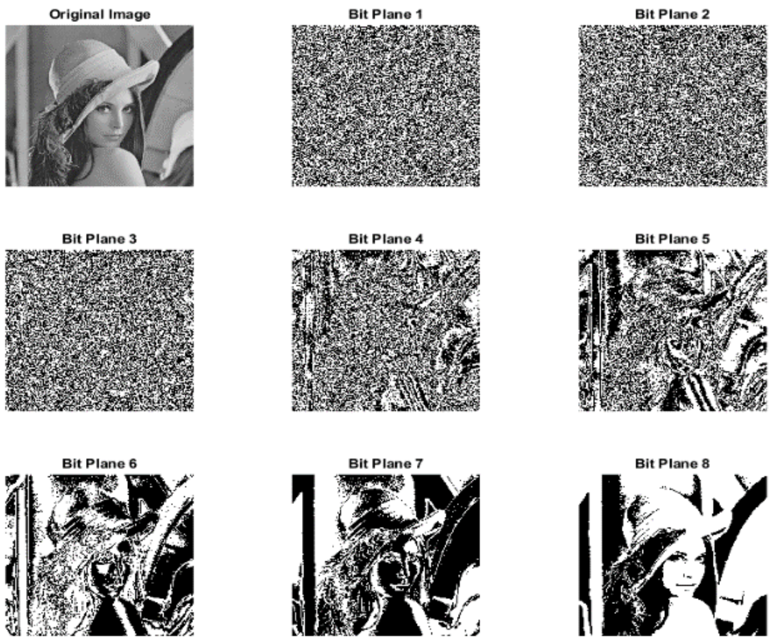
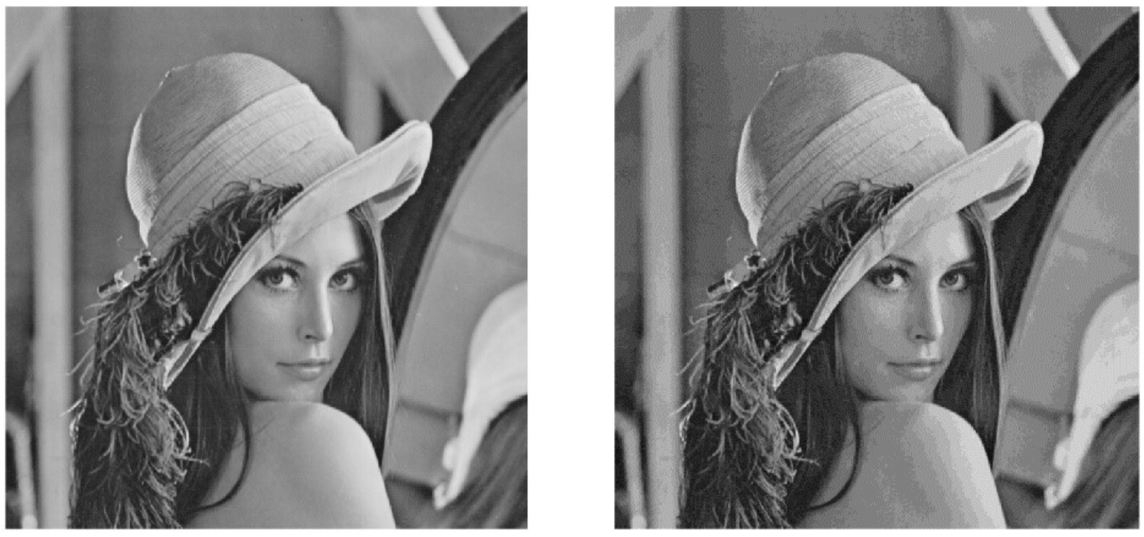


Figure 2.9 Bit planes of a sample gray scale image.

To visualize the effect of bit-plane-slicing, we consider the image shown in Fig. 2.10 (a). Fig. 2.10 (b) shows the image generated using the most significant 4 bit-planes and ignoring the lower 4 bit-planes of the original image shown in Fig. 2.10 (a).



(a) (b)

Figure 2.10 (a) Original image (b) Image generated using the most significant 4 bit-planes.

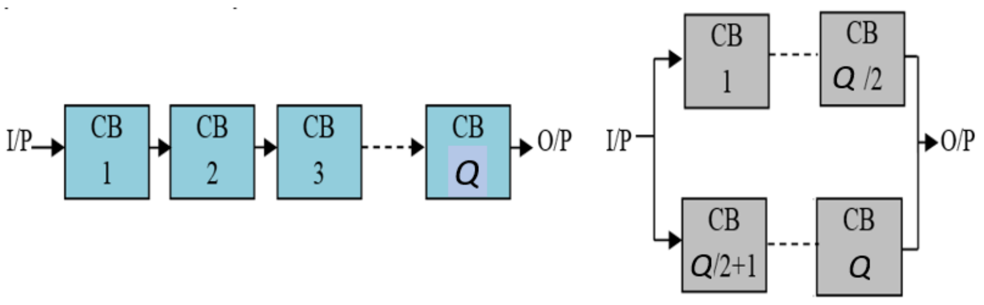
It can be observed from this figure that the image in Fig. 2.10 (b), generated using the most significant 4 bit-planes, conveys all the visual information of the original image. However, the size of the image in (b) in terms of the computer memory is half to that of the image in (a). It is noted that using an image with fewer bit-planes instead of the full image will result in a simplification of the processing system.

2.4 Performance Improvement of a Hardware-based Architecture

The performance of a hardware-based architecture in terms of the delay can be improved by reducing the delay of the underlying circuits that are used in the architecture [41]. This section discusses the basic techniques that are used to reduce the delay in the hardware design for achieving higher operating frequencies.

2.4.1 Reducing Circuit Delay by Parallel Processing

In digital systems, the delay of a circuit is determined by the number of cascaded stages of combinational logic blocks from the input to the output. This delay can either be reduced by reducing the number of stages, or by reducing the delay of each stage. Fig. 2.11 (a) shows Q number of cascaded combinational blocks (CB), each with a delay of δ. As a result, the total latency of this circuit is $Q\cdot\delta$. Fig. 2.11 (b) shows a rearrangement of the Q blocks in a different format, where the number of cascaded blocks from the input to the output is reduced by 50%. As a result, the delay of the circuit is reduced to $(Q\cdot\delta/2)$. Such a parallel arrangement of the blocks is possible only if there is no dependency between the upper and lower blocks in Fig. 2.11 (b). For a digital system having multiple paths from the input to the output, each having a different delay, the delay of the longest path is considered as the maximum combinational delay. This path is known as the critical path delay [42], and the maximum operating frequency of the circuit is calculated using the critical path delay.



(a) (b)

Figure 2.11. (a) Cascaded blocks (b) Cascaded blocks in a parallel arrangement.

2.4.2 Increasing Operating Frequency using Pipelining

The maximum frequency at which a circuit can operate is the inverse of its critical path delay. If the delay of the critical path is reduced, the operating frequency of the circuit will increase. The method used for reducing the critical path delay is shown in Fig. 2.12. Fig. 12 (a) shows two D flip-flops (FF) with a combinational circuit between them. The delay of this combinational circuit is δ. In Fig.2.12 (b), the same combinational circuit is split into two stages (CC I and CC II), each having a delay of $\delta/2$ and a pipeline register is inserted between the two stages. As a result, the operating frequency of the circuit in (b) is twice that of the circuit in (a). This type of approach is possible only if the circuit can be split into two or more stages.

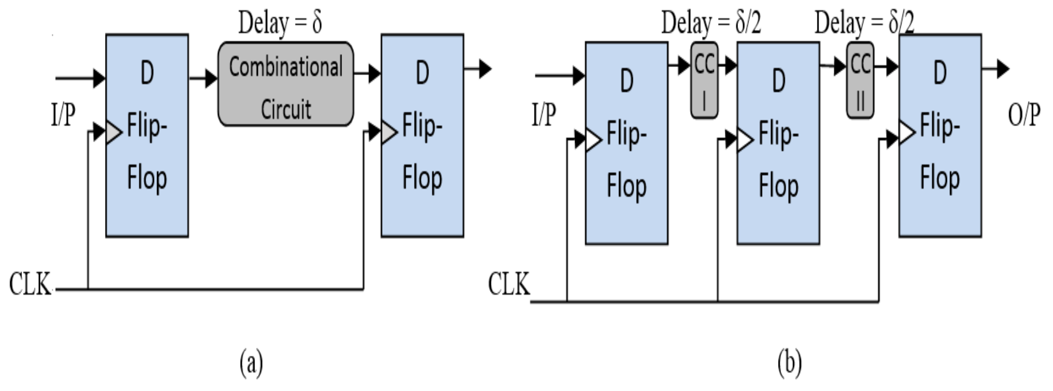


Figure 2.12. (a) Combinational circuit as a single unit (b) Combinational circuit split into two stages.

Super-scaler and super-pipelined architectures are commonly used in designing CPUs and GPUs for increasing their throughput [43]. While the super-scaler architecture aims at replicating the hardware for parallel processing, the super-pipelining aims at making the stages of a processing unit as shallow as possible by splitting them and adding pipelining registers [44]. Architecture such as [45] takes advantage of pipelining as well as parallel processing for designing a median filter, which can provide a very high throughput.

2.5 Performance Parameters used for Evaluation

In this section, we discuss the various performance parameters used for the evaluation of the implementations of the proposed median filter architectures. The proposed hybrid architecture for median filtering provides a high throughput at the cost of the output image quality. To evaluate the quality of the images filtered using the proposed hybrid architecture for median filtering, we calculate the peak-signal-to-noise-ration (PSNR) and the structural similarity index (SSIM) between the original and the filtered images. Since the median filter architectures presented in this thesis are implemented on FPGAs, parameters such as the area (in terms of slice-LUTs and slice-registers), frequency and power are used to evaluate the hardware performance. To analyze if an implementation can satisfy the real-time requirements of the video format considered, we calculate the throughput of the implementation in terms of the number of frames it can filter per second.

2.5.1. Filtered Image Quality

The quality of a filtered image can be judged qualitatively by visualizing the image and quantitatively by calculating the parameters, such as PSNR and SSIM.

a) Peak signal-to-noise ratio

The peak signal-to-noise ratio (PSNR) is the ratio between the maximum possible signal power and the power of the noise distorting the signal [46]. The PSNR is given by

$$PSNR = 20\log\left[\frac{255}{RMSE}\right] \tag{2.2}$$

where RMSE is the root mean square error and is the equal to the square root of the mean square error (MSE), which is calculated using

$$MSE = \frac{1}{m \times n}\sum_{i=1}^{m}\sum_{j=1}^{n}[I'(i,j) - I(i,j)]^2 \tag{2.3}$$

where I is the original image and I' is the filtered image, both of size m × n.

b) Structural Similarity Index

Structural similarity index (SSIM) is used to compare the luminance, contrast and structure of two different images. It can be treated as a similarity measure of two different images. SSIM of two images X and Y is defined as [47]

$$SSIM(X,Y) = \frac{(2\mu_x\mu_y + C1) \times (2\sigma_{xy} + C2)}{(\mu_{x^2} + \mu_{y^2} + C1) \times (\sigma_{x^2} + \sigma_{y^2} + C2)} \tag{2.4}$$

where μ_i ($i = x$ or y) is the mean intensity, σ_i ($i = x$ or y) is the standard deviation, and Ci (i = 1 or 2) is the constant to avoid instability when $\mu_{x^2} + \mu_{y^2}$ is very close to zero and is defined as $Ci = (KiR)^2$, where K_i << 1 and R is the dynamic range of pixel values. For example, R = 255 for 8-bit gray scale image.

2.5.2. Hardware Implementation Parameters

The typical parameters used for the performance evaluation of an FPGA-based hardware implementation are area, delay and power [48]. After implementation of the design on an FPGA, the area is reported by the tool (Xilinx Vivado design suite) in terms of the slices. The tool reports the number of slice-LUTs that are used to map logic, and the number of slice-registers that are used as flip flops for storage. These numbers provide a clear idea of the area of the design on a FPGA device. The implementation tool also reports the total delay (net delay + logic delay) for the implemented design, after inserting the constraints in the timing constraints editor. Using this delay, the maximum operating frequency of the design is calculated. The throughput T of a video processing system in number of frames per second (fps), of an implementation is given by

$$T = v \left[\frac{(f_{max} \times 10^6) + L}{F_s} \right] \tag{2.5}$$

where ν is the number of cores of the architecture, f_{max} is maximum operating frequency in MHz, L is pipeline latency and F_s is size of the frame in number of pixels. Since L is very small compared to the operating frequency, it may be ignored. For all the implementations presented in this thesis, each core processes 1 pixel per clock and the size of each input pixel is considered as 8-bits. For filtering color images having pixel size of 24-bits, an approach presented in [49] can be used to map the three-dimensional color vectors into one-dimensional space (8-bit data). The mapped data can be filtered using the median filters presented in this work, providing the same throughput T, as that given by Eq. (2.5). Another method of filtering a color image is to filter only the luminance plane using CMF and then converting back to the original color space [50].

The Xilinx Xpower analyzer tool reports the static and dynamic powers of the implemented design along with their sum, which is the total power of the design. Although in this work, the primary hardware performance parameters are area and delay, we analyze the power reported by the Xpower analyzer tool by vector-less analysis of the implemented netlist, for various implementations in Chapter 3 and Chapter 4. The Xpower analyzer tool estimates the total on-chip power by assigning the default signal rates and static probabilities to the design nodes [51] [52]. Although the power estimated by this tool gives a rough estimate of the total on-chip power of the design [53], we can analyze the total on-chip power as a function of the pipeline latency, number of the most significant bits and window size. We specify the operating frequency of the design in the Xpower analyzer tool, in order to obtain a better estimate of the total on-chip power.

2.6. Summary

This chapter has presented details of the sorting and non-sorting-based median calculation techniques that are primarily used in the implementation of median filters. The advantages and disadvantages of the sorting and non-sorting-based median calculations have been discussed with respect to hardware implementation. An analysis of the noise density and window size has been provided with respect to the quality of the filtered image. Background techniques used in our work, such as the decision-based median filtering and bit-plane-slicing have been discussed with examples. Parameters used for the performance evaluation with respect to the quality of filtered images and hardware implementation have been presented along with relevant details.

Chapter 3

Design and Implementation of a Hybrid Architecture for Median Filtering

3.1 Introduction

When median filters are implemented on a hardware platform such as FPGAs, the primary aim is to optimize at least one of the three VLSI performance parameters, namely, area, frequency and power. The VLSI technology used for fabrication of FPGAs is based on CMOS circuits [54]. The performance parameters of CMOS circuits are interdependent [55] and hence, it may not be possible to optimize more than one of these parameters at the same time. For instance, increasing the operating frequency increases the power dissipation [56]. Most hardware-based median filters concentrate on optimizing one of the three performance parameters such that the other parameters are not adversely affected. This type of approach is adopted, since the parameter selected for optimization is solely dependent on the target application. For example, in applications that have real-time requirements, the operating frequency of the design is most important and hence the power consumption and area are of secondary concern. This chapter presents the design and FPGA implementation of a median filter architecture for real-time filtering of FHD videos [57]. The implementation of the proposed architecture on different generation FPGAs provides a high frequency of operation and optimizes the area at the same time. The optimization in the proposed median filter architecture is achieved at a minimum cost of filtered image quality.

The design of the proposed architecture for median filtering is based on three different strategies. The first strategy used in the proposed architecture is the histogram-based operation, which reduces the combinational delay of the circuit, resulting in a high frequency of operation. The hardware implementation of the conventional histogram-based median calculation uses memory bins for storing the histogram counts as in the case of [30] and hence, provides a low throughput. Therefore, we design an architecture that does not use memory bins for storage, but instead uses registers. Such an architecture will result in a very large area due to the wide range of pixel values (0 to 255). It will also require a large number of registers (256) to store the histogram counts, resulting in an increased area. In order to reduce the area, we use the concept of bit-plane-slicing and process only B most significant bits (MSBs) for median calculation. This is the second strategy used in the proposed architecture and this results in a low area utilization. The effect of processing only the most significant B-bits of the pixel values results in an approximate median calculation, due to which the quality of the filtered image is compromised. Hence, to improve the quality, we use the strategy employed in the adaptive switching median filter (ASMF) in the proposed architecture. This is the third strategy used in the proposed architecture for median filtering.

The underlying strategies used in the proposed architecture for median filtering were discussed in the background material presented in Chapter 2. Although the underlying strategies used in the proposed architecture have been implemented discretely on FPGAs, their combined effect has never been implemented and analyzed. In view of using the above-mentioned strategies in the design of the proposed architecture, we refer to the resulting architecture as the hybrid architecture for median filtering (HAMF). We implement the proposed HAMF on FPGAs with three different values of B and analyze its effect on the output image quality, operating frequency, area and power.

3.2 Approximate Median Calculation

Many neighboring pixels in an image have values, which are similar or very close to one another [3]. Also, the most significant bit-planes of an image consists of more significant information. As a result, an approximate median of the values in a window of an image can be calculated by processing only the most significant B-bits instead of processing all the bits. The number of hardware components required to implement a median filter, which processes B-bits is significantly less than that required to implement a median filter that processes W-bits, provided $B < W$. Processing the most significant B-bits results in a hardware simplification and is a commonly used technique in image processing algorithms. Median filters that use higher nibble (4-bits) of pixel values have been implemented in [21] and [22], resulting in a lower area and a higher operating frequency. Although it is common to segment a pixel value into upper and lower nibbles, and process only the upper nibble, this concept is derived from the bit-plane-slicing technique. Hence, we utilize and generalize this concept, and process the most significant B-bits to calculate the approximate median value.

Fig. 3.1 shows the weight of each bit of a pixel towards its influence on the value of the pixel. All weights are normalized to 256, which is the maximum value of each pixel of a gray scale image. The weight of each bit is calculated as $2^{r-1}/256$, where r is the index of the bit, 1 being the LSB and 8 being the MSB. It is seen from this figure that the MSB (bit 8) contributes to 50% of the data. If we consider bits 8 and 7, they will together contribute to 75% of the data. Bits 8, 7 and 6 together contribute to 87.5% of the data value and similarly, bits 8 to 5 together contribute 93.75% to the data. We consider $B = 2$, 3 and 4, which will calculate the median using 75%, 87.5% and 93.75% of the pixel data. For $B = 5$, only 3.125% of additional information will be used. However, its hardware requirement will increase significantly and hence, we will consider only 2, 3 and 4 as the suitable values of B in our implementations.

By processing only the most significant B-bits of pixel values, a B-bit median will be obtained. However, the word length W of the median should be the same as that of the word length of the input data elements. Our method to process only the most significant B-bits and calculate a W-bit approximate median is explained by the following example.

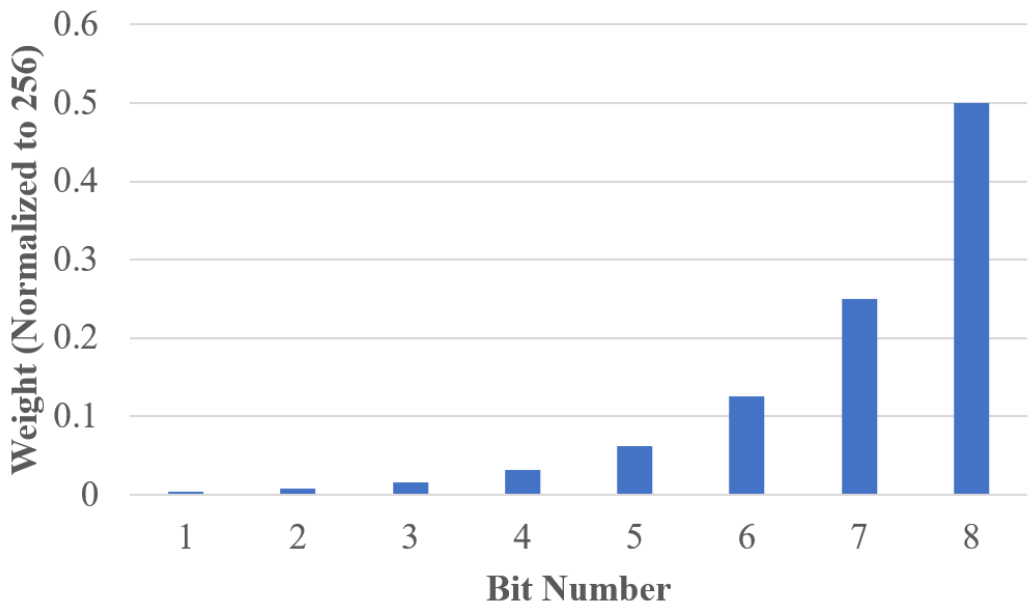


Figure 3.1 Weight of bits in value of a pixel.

Consider a window of size $W_s = 3$ with the values of its elements, as shown in Fig. 3.2 (a). An approximate median of the values with $W = 8$ in this window is to be calculated with $B = 4$. As a first step, the lower nibbles (4-bits) of all the values in the input window are masked and the resulting 3 × 3 window is shown in Fig. 3.2 (b). In the next step, we sort this window using only the higher nibbles and the resulting window is shown in Fig. 3.2 (c). The median as per the sorted data is 0x**4.** Although its lower nibble has a value 0x1, it is not available since it is masked. Hence, we need to find a lower nibble for the value 0x4 in order to find the complete 8-bit median. In our method, we pick the first element from the sorted window, which has its corresponding higher nibble equal to 0x4, which for this example is 0x46. Although the actual median of the values considered in this example is 0x42, our result is 0x46 and hence, we say an approximate median is calculated. The maximum error resulting from this technique is 0xF, which is possible in a case where the 4-bit median is calculated to be 0x4, the actual median is 0x40 and the median value selected is 0x4F.

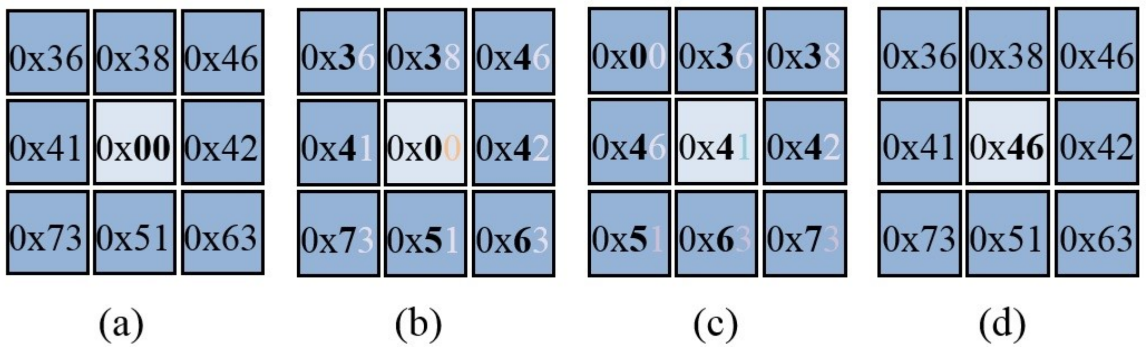
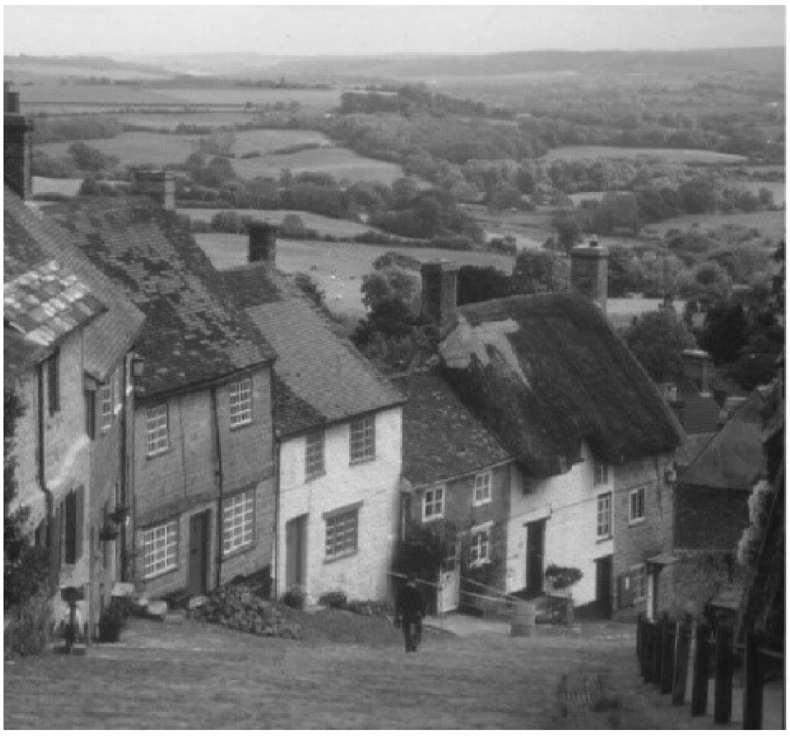


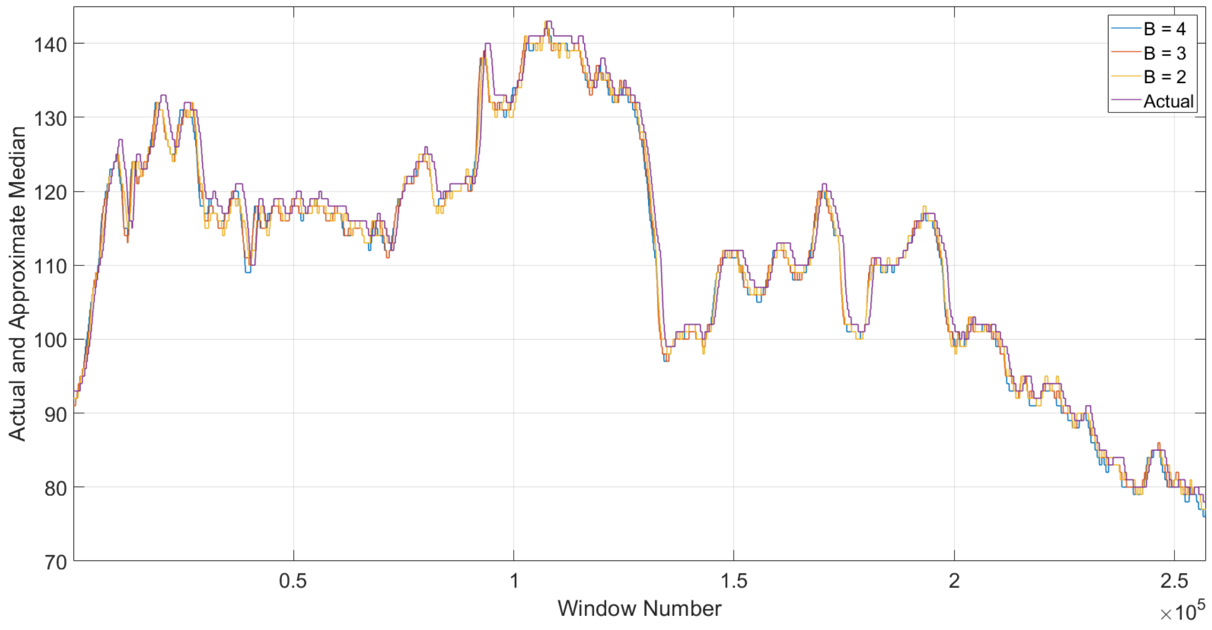
Figure 3.2 Median calculation using only higher nibbles. (a) Input window (b) Window with lower nibbles masked (c) Window sorted using only the higher nibbles (d) Output window.

When this technique is applied for filtering images, it is experimentally observed that there is no significant difference in the quality of the filtered image, irrespective of the value selected as the lower nibble. For the example considered above, the possible values of the lower nibble for 0x4 are 0x6, 0x1, and 0x2. We select the first value (in the sorted matrix), i.e., 0x6 to be the value of the lower nibble of the median, since it does not add any overhead to the hardware. Similarly, for image filtering operation, we choose the first lower nibble in all the windows. Although the lowers nibbles are not used for processing or calculation, when the higher nibbles are processed, the former are always attached to the respective higher nibbles. For many windows in an image, the actual or the exact median is calculated by processing only the most significant B-bits. For instance, in the example considered above, if we interchange the positions of the pixels with values 0x46 and 0x42 in the input window, we observe that the calculated median is 0x42, which is the actual median.

To demonstrate the effect of calculating the approximate median using the most significant B-bits on the output image quality, images from the database [58] and a frame of videos from the database [59] are processed to calculate the actual and approximate medians with B = 2, 3 and 4. Results of the approximate filtering on a sample image are shown in Fig 3.3. In this figure, (a) shows the input image and (b) shows the graphs of actual and approximate medians calculated using the proposed technique with different values of B for various pixels of the input image. It is seen from Fig. 3.3 (b) that the difference between the actual median and the median calculated using the most significant B-bits is negligible.



(a)



(b)

Figure 3.3 (a) *Goldhill* Image from the database given in [58]. (b) Actual median and approximate medians with B = 4, 3 and 2 for various pixels.

Fig. 3.4 shows the filtered images. In this figure, (a) is the image filtered using the actual median and (b), (c) and (d) are images filtered using the proposed technique with $B = 4$, 3 and 2, respectively. Although the images (b), (c) and (d) appear very similar to the image (a), there is a substantial difference between them. This difference is due to the difference between the approximate and actual median values, even though the later difference is not observable in Fig. 3.3 (b). The difference in the image quality is calculated in terms of PSNR of images in Fig. 3.4 (b), (c) and (d) with respect to the image in Fig. 3.3 (a) and is shown in the respective Figures 3.3 (b), (c) and (d).

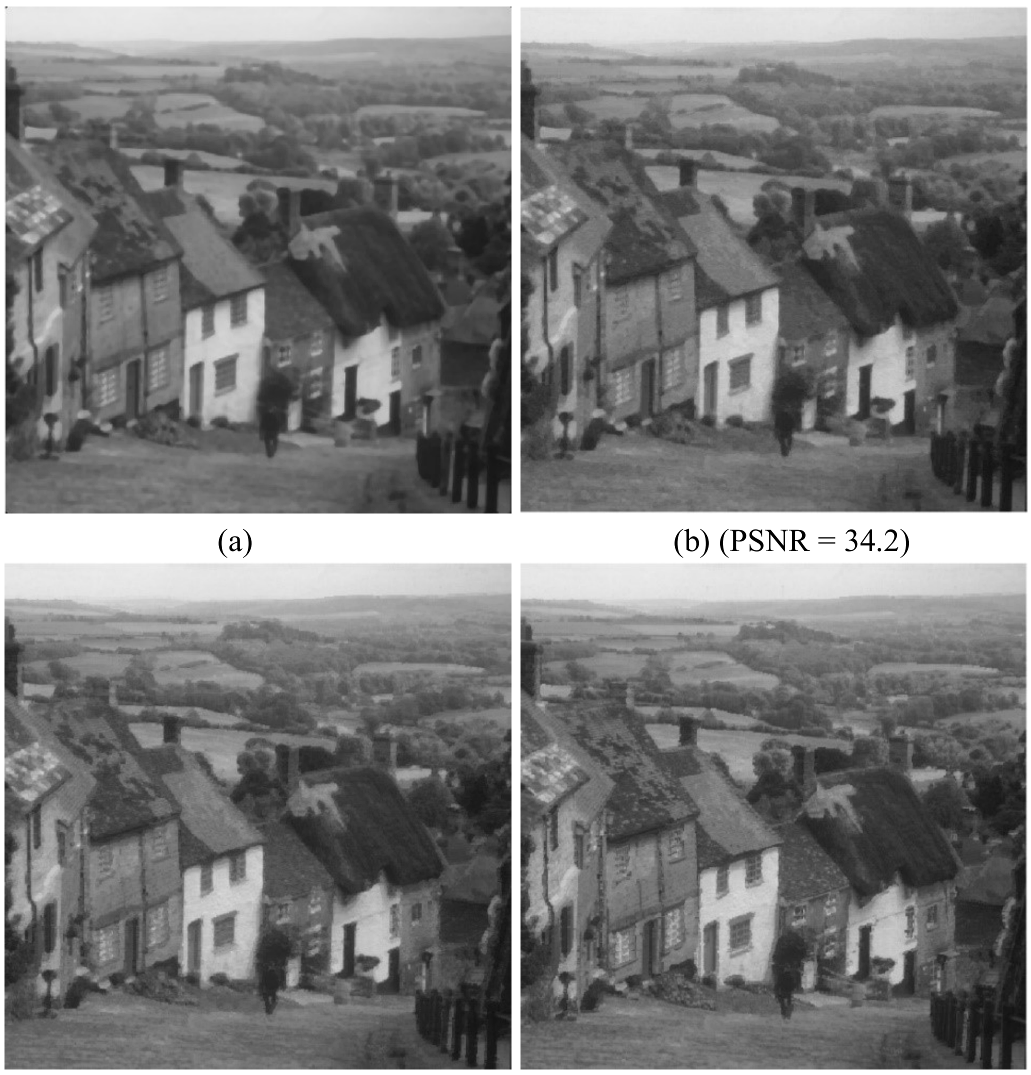
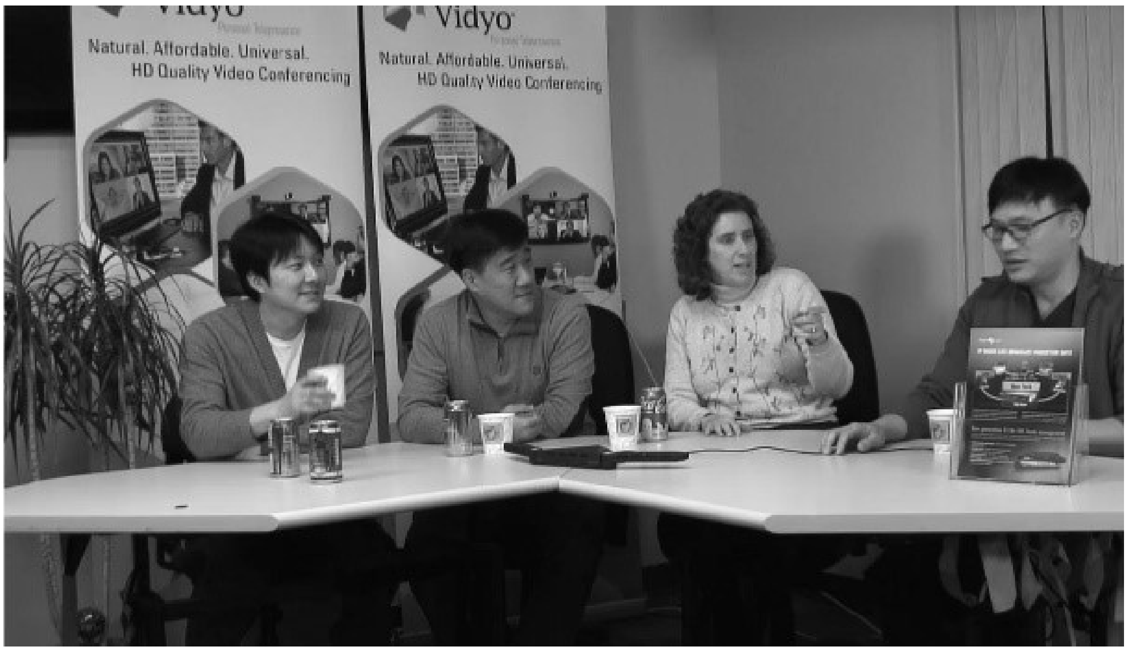


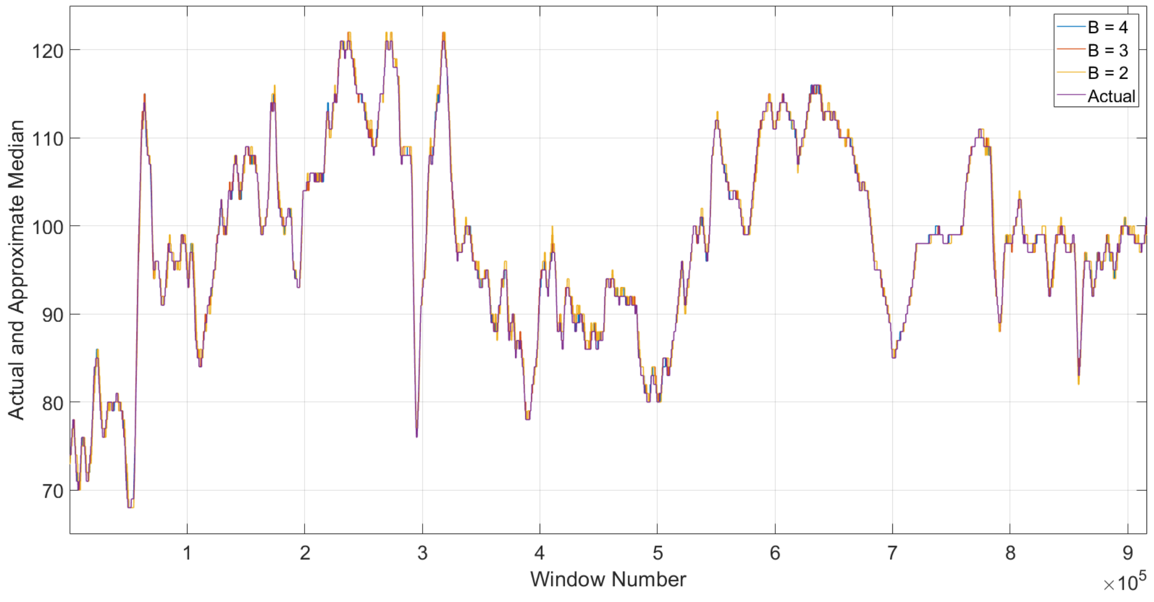
Figure 3.4 *Goldhill* image filtered with (a) actual median, approximate median with (b) $B = 4$, (c) $B = 3$ and (d) $B = 2$.

Fig. 3.5 shows similar results on a sample frame taken from the *four people* video from the database in [59] with the corresponding filtered images, shown in Fig. 3.6. From these figures, the same conclusion can be drawn as in the case of the previous experiment, wherein the image from the database in [58] was considered.



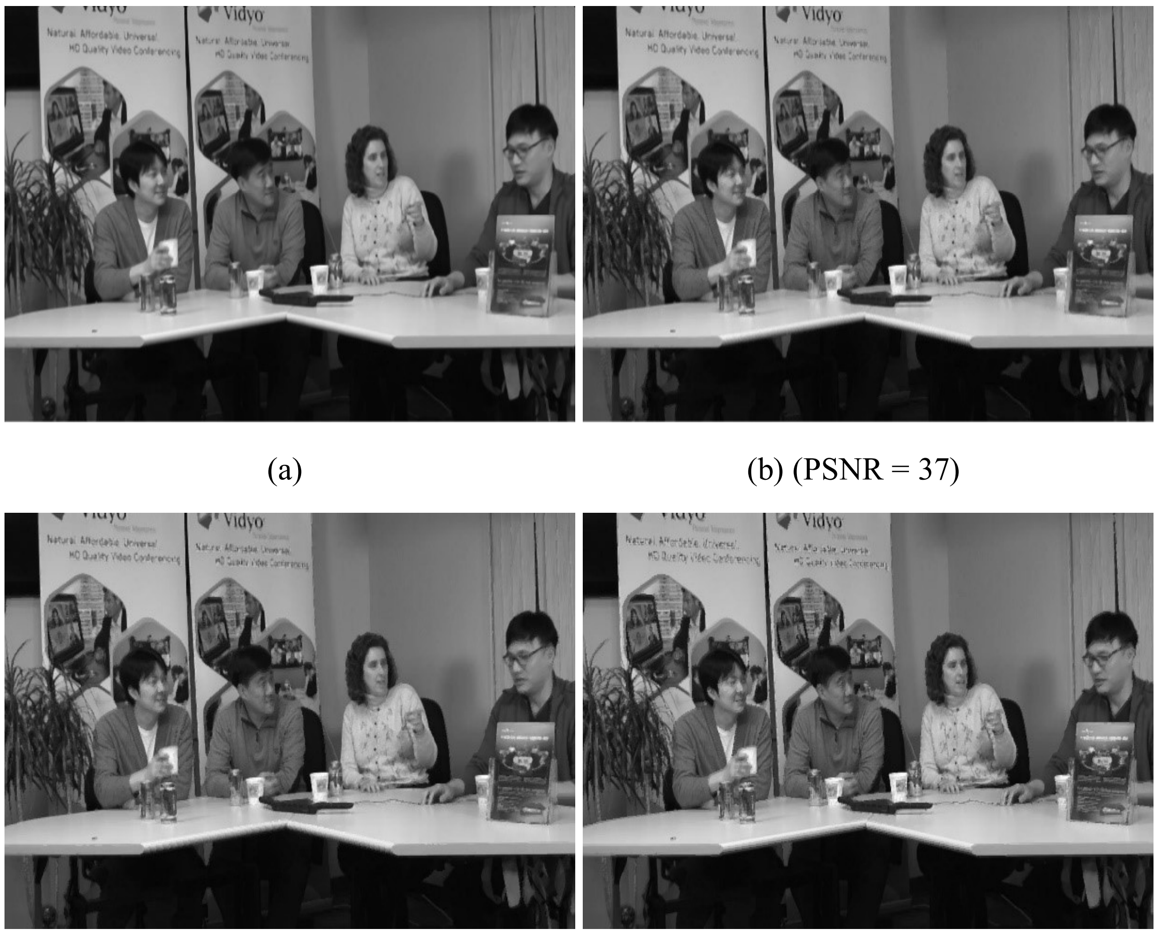
Vidyo
Natural. Affordable. Universal.
HD Quality Video Conferencing
Vidyo
Natural. Affordable. Universal.
HD Quality Video Conferencing

(a)



(b)

Figure 3.5 (a) A frame of *four people* video from the database given in [59]. (b) Actual median and approximate median with B = 4, 3 and 2 for various pixels.



(a) (b) (PSNR = 37)

(c) (PSNR = 32.4) (d) (PSNR = 28.4)

Figure 3.6 A frame of *four people* video filtered with (a) actual median, approximate median with (b) B = 4, (c) B = 3 and (d) B = 2.

From these experiments, it is observed that a better approximation to the actual median is obtained when a higher value of B is employed.

3.3 Block Diagram of the Proposed Hybrid Architecture for Median Filtering

Fig. 3.7 shows the block diagram of the proposed HAMF that can process N W-bit input values. Out of the W-bits, B-bits are processed using the histogram-based technique to calculate a B-bit median value. The calculated B-bit median is used to select a W-bit median from the input window, as explained in the previous section. Based on the decision of the impulse noise detector, the selected W-bit median or the value of the input pixel is passed on to the output. The figure shows an input window consisting of N input values. Each pixel in this window is represented by W-bits. All the pixel values are fed to the proposed HAMF, of which the most significant B-bits of all the pixel values are read by the histogram calculation block. This block calculates the histogram values H_0 to $H_{(2^B-1)}$, of the input data. The histogram values are then added successively, starting with the value of H_0, until the result of the addition exceeds $(N-1)/2$. The index i of the last added H_i is the B-bit median of the data fed to the histogram calculation block.

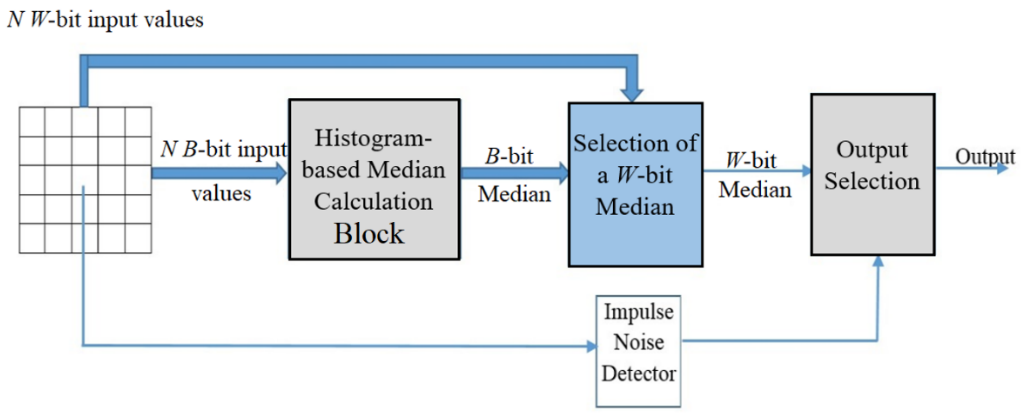


Figure 3.7 Proposed hybrid architecture for median filtering.

3.3.1 Architecture of the Histogram-based Median Calculation Block

Fig. 3.8 shows the architecture of the proposed histogram-based median calculation block for $B = 2$ and $N = 5$, as an example. It is seen from this figure that there are 3 stages in the median calculation block. Stage I is the comparator stage, where all the input values are compared with all the values in the range 0 to 2^B-1. The output of stage I consists of N sets, each set having 2^B binary values. The second stage (Stage II) adds all the binary values corresponding to each value in the range 0 to 2^B-1 to generate the histogram values H_0 to $H_{(2^B-1)}$ of the input data. Stage III adds these values successively and after each addition, compares the result to the value $\lceil N/2 \rceil$ which is equal to 3 for this example. As soon as the comparison becomes true, the index i of the last added H_i is selected as the output. The various stages of the architecture have pipeline registers between them, as shown in this figure.

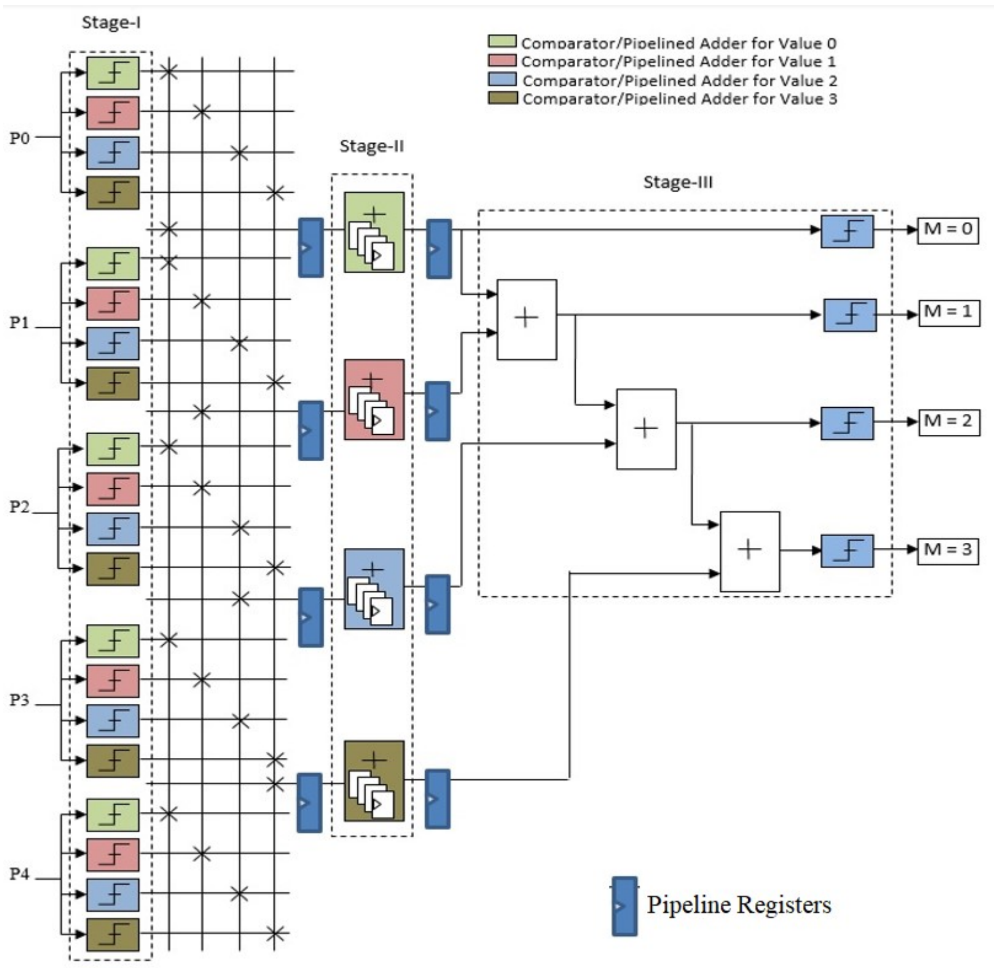

Stage-I
Comparator/Pipelined Adder for Value 0
Comparator/Pipelined Adder for Value 1
Comparator/Pipelined Adder for Value 2
Comparator/Pipelined Adder for Value 3
Stage-II
Stage-III
P0
P1
P2
P3
P4
M = 0
M = 1
M = 2
M = 3
Pipeline Registers

Figure 3.8 Pipelined histogram-based median calculation block.

The structure of the histogram-based median calculation block is the same (3-stage) irrespective of the values of B and N. However, the number of comparators in stage I increases with increasing values of N and B. The number of comparators required in stage I is $N \cdot 2^B$. The number of adders in stage II is dependent on B alone and is equal to 2^B; however, each adder is of N-bits. The number of adders in stage III is also dependent on B alone and is equal to 2^B-1.

The operating frequency of the proposed median calculation block is dependent on the delay of its critical path. By analyzing the proposed architecture, we find that this delay is the delay of stage II. Hence, we apply the concept of super-pipelining technique by introducing registers inside stage II to increase the operating frequency of the design. Although increasing the number of registers will increase the latency of the design, the corresponding throughput will also increase. An analysis of the relationship between the latency, operating frequency and area will be presented in the results section of this chapter. Since a super-pipelined architecture results in a greater utilization of the registers, we implement the super-pipeline with different latencies to analyze the relation between the throughput and the utilization of the registers.

3.3.2 Decision Based Median Filtering

Fig. 3.9 shows the modification carried out in a conventional median filter to convert it to a noise adaptive switching median filter. It is seen from this figure that the output pixel is selected as the median of the input window only if the input pixel is found noisy. Salt and pepper noise in a gray scale image can be easily detected by comparing an input pixel value with a value of 255 for salt and with a value of 0 for the pepper. If one of these two comparisons hold true, the input pixel is detected as noisy and is filtered. If neither of the comparison holds true, the pixel is interpreted as a non-noisy pixel and its corresponding value is passed to the output. Filtering only the noisy pixels results in a better image quality without an undesired smoothing effect.

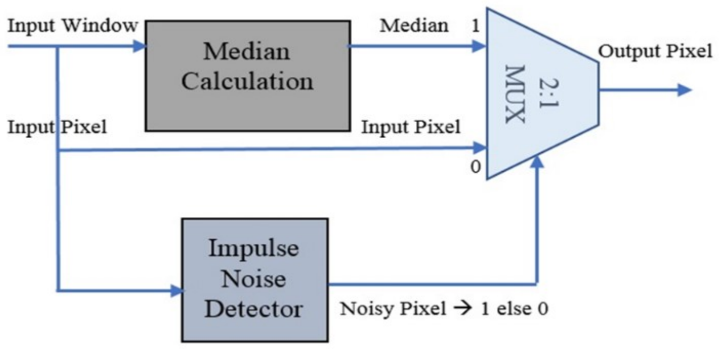


Figure 3.9 Noise adaptive switching median filter.

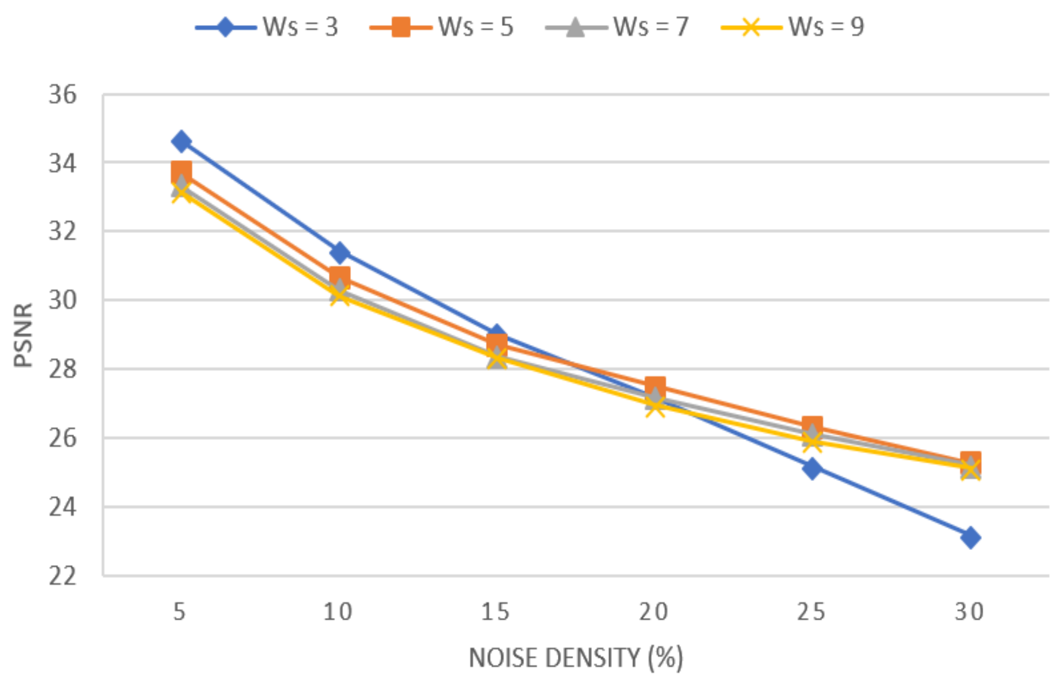
This technique can be applied to any median filter, which does not use noise adaptive filtering to convert it to a noise adaptive switching filter. This type of filtering is also called as the decision based median filtering. In the proposed HAMF block diagram shown in Fig. 3.7, the decision based median filtering is achieved using the impulse noise detector and output selection blocks.

3.4 Results and Analysis

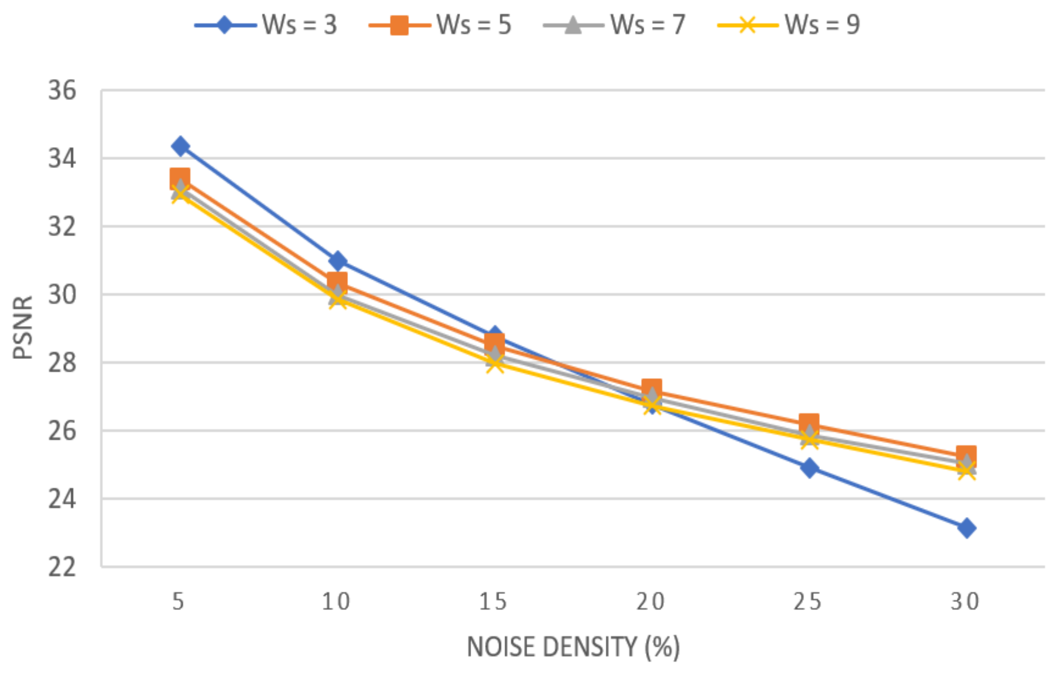
In this section, results on the quality of images filtered by an algorithmic level Matlab implementation of the proposed HAMF, depicted in Fig. 3.7, are analyzed for different values of the noise density, window size and B. Based on this analysis, an appropriate window size that is suitable for hardware implementation is selected. The proposed HAMF is implemented for the selected window size using RTL coding in VHDL. The design is simulated for functional verification and mapped to different generation FPGAs to evaluate the hardware performance. We implement the proposed HAMF on FPGAs with different values of B and pipeline latency L, to study the effect on the operating frequency, power and area. The throughput of the hardware implementation of the proposed HAMF in terms of FHD video frames per second (fps) is calculated to see whether it satisfies the real-time requirements of FHD videos.

3.4.1 Filtered Image Quality

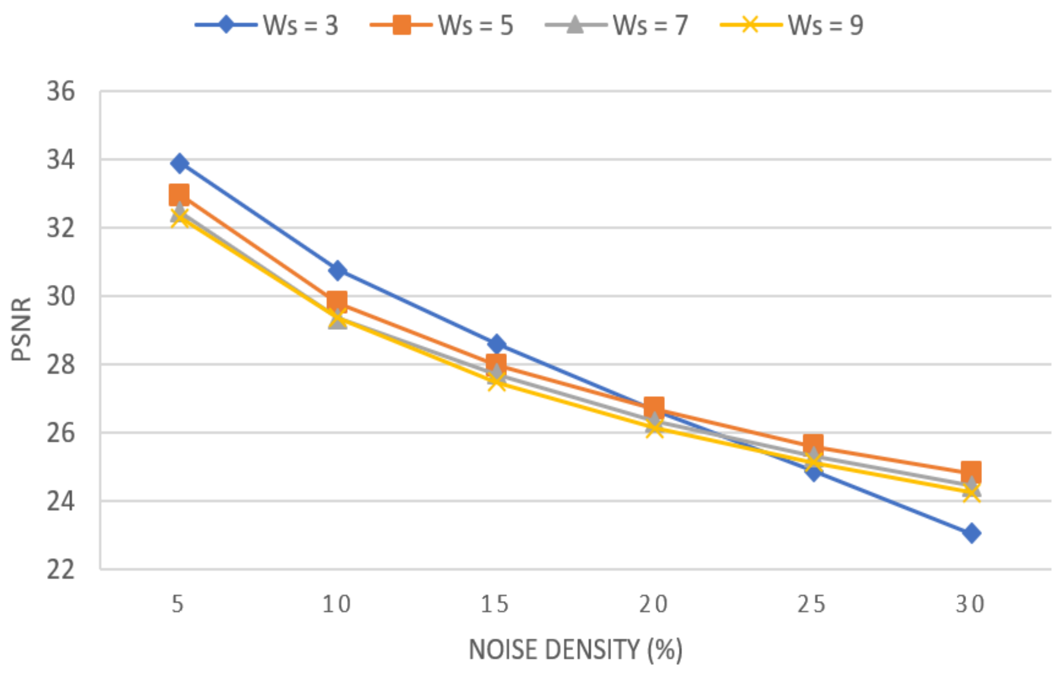
To evaluate the performance in terms of the filtered image quality, images and frames of videos from databases in [58] and [59], respectively, are considered. Salt and pepper noise of different densities are added, and the noisy images and video frames are filtered using the algorithmic level Matlab Implementation of the proposed HAMF with different values of B. For each filtered image, PSNR and SSIM are calculated with respect to the original image. The PSNR and SSIM values of the images filtered using adaptive switching median filter (ASMF) are also calculated. Fig. 3.10 shows the results of PSNR with increasing noise density on the *baboon* image from the database in [58]. Figures 3.10 (b), (c) and (d) show the results for $B = 4$, 3 and 2, respectively, for different window sizes. Fig. 3.10 (a) show the result for different window sizes for the image filtered using ASMF.



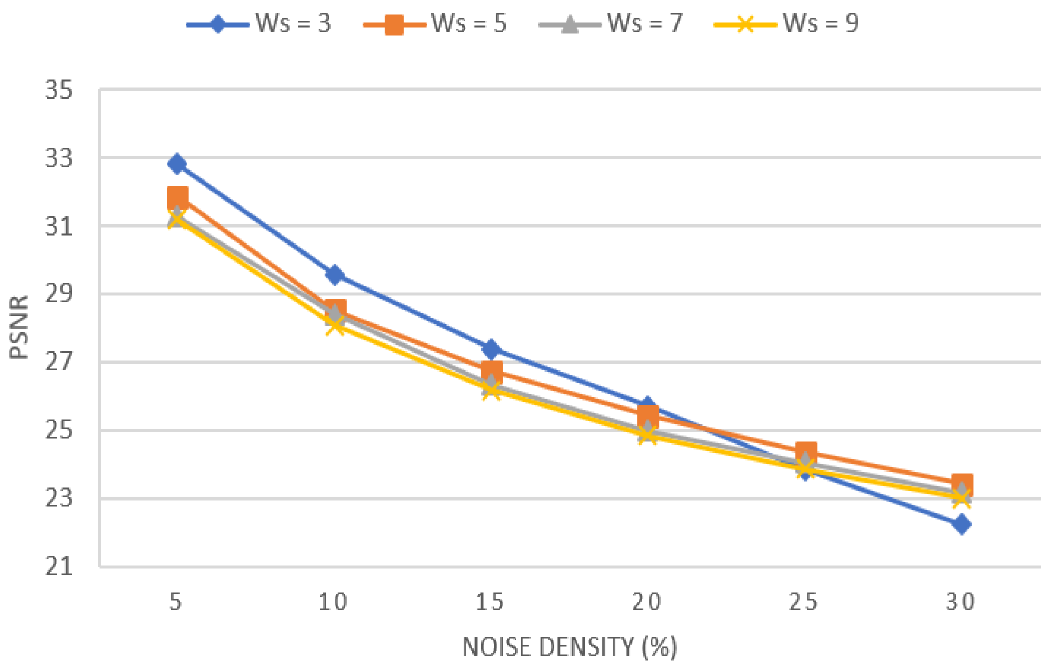
(a)



(b)



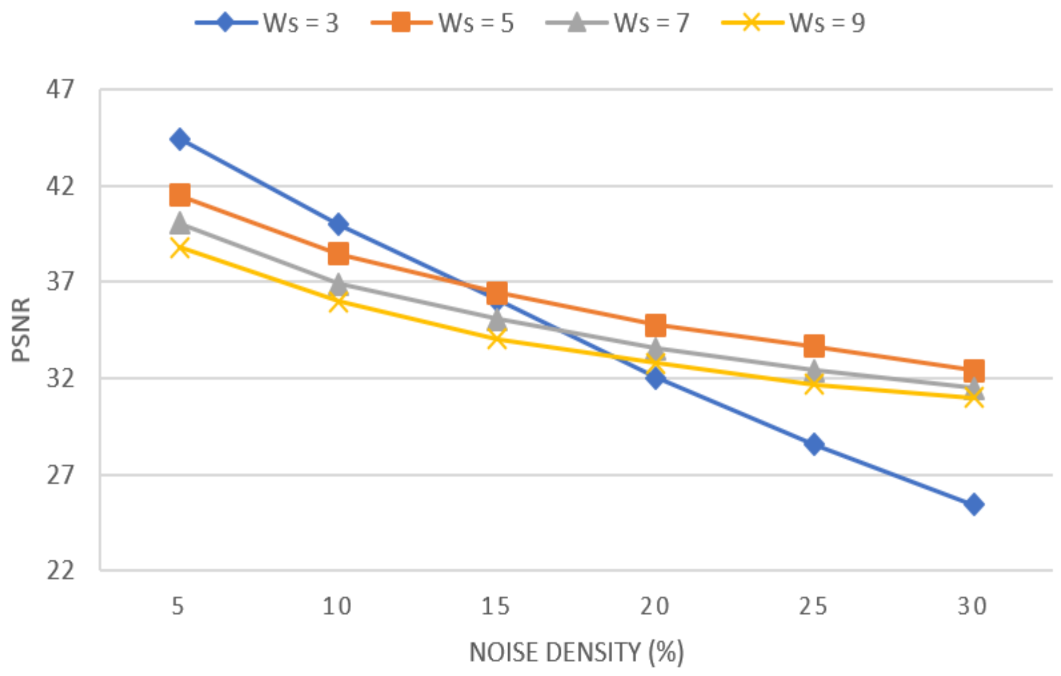
(c)



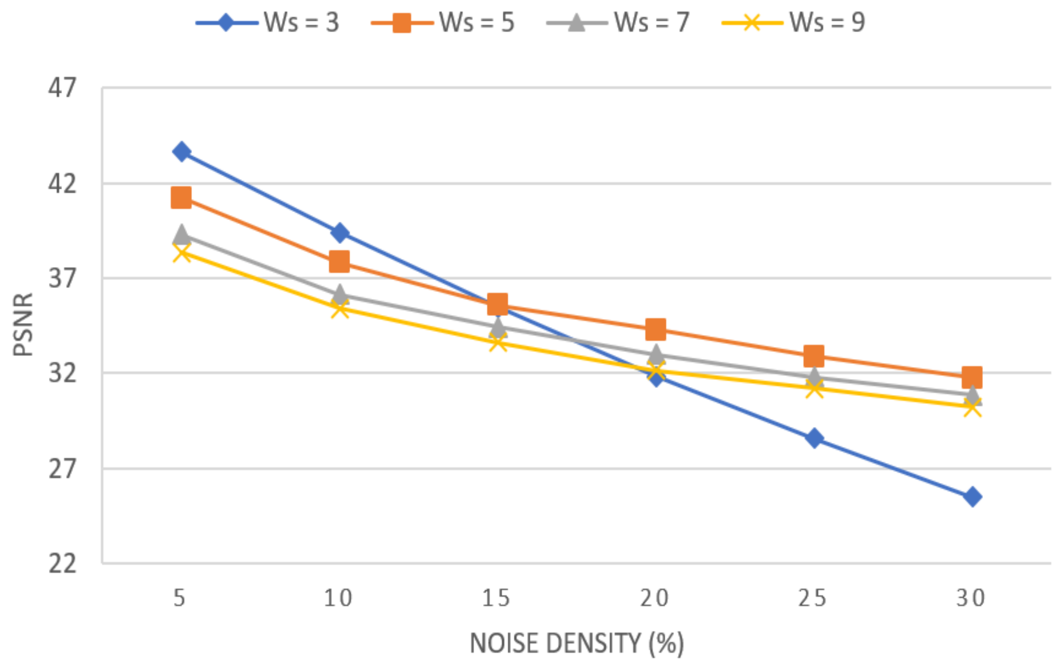
(d)

Figure 3.10 PSNR as a function of noise density for the *baboon* image for different values for W_s, using (a) ASMF and the proposed technique with (b) $B = 4$ and with (c) $B = 3$ and (d) $B = 2$.

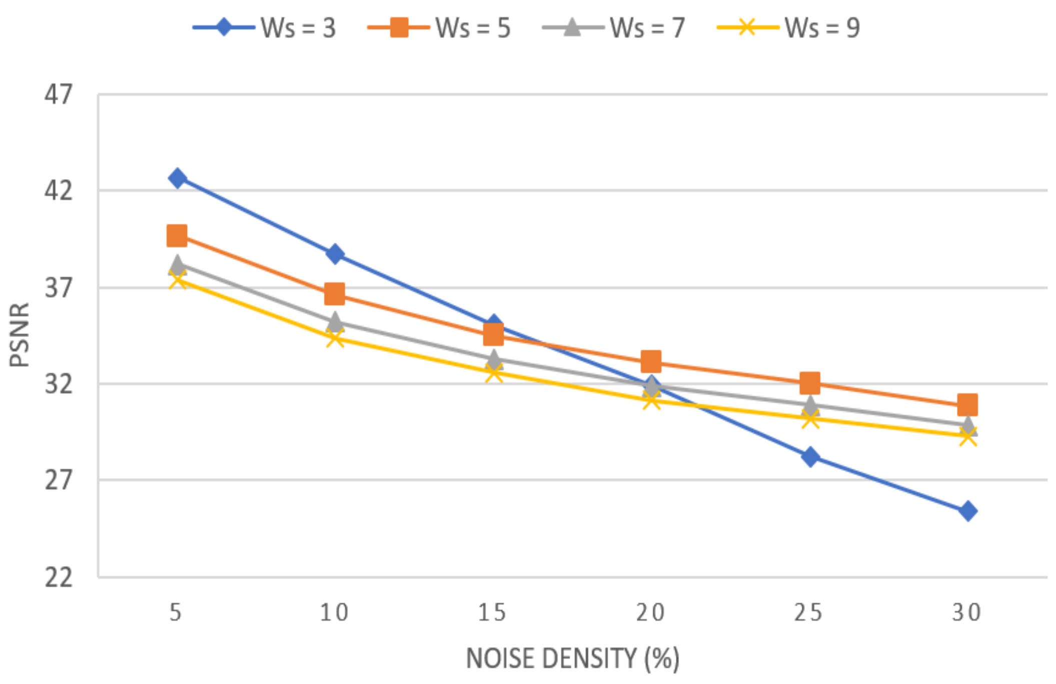
Fig. 3.11 shows similar results on a single frame of the video *four people* from the database in [59].



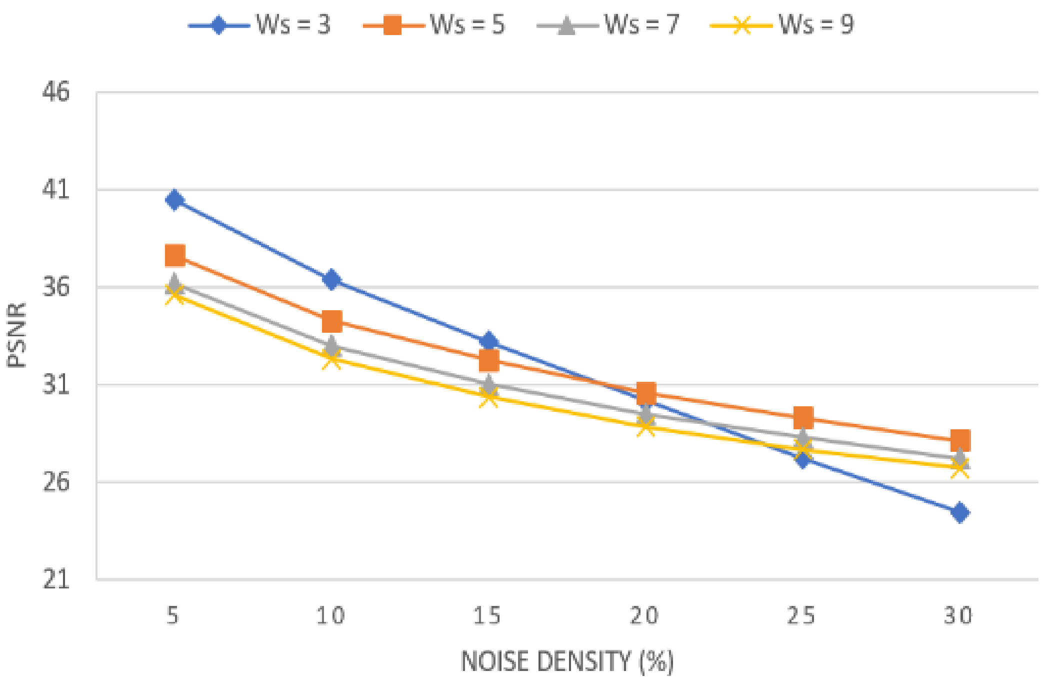
(a)



(b)



(c)



(d)

Figure 3.11 PSNR as a function of noise density for a single frame of *four people* video for different values of W_s, using (a) ASMF and the proposed technique with (b) $B = 4$ and with (c) $B = 3$ and (d) $B = 2$.

It is observed from Fig. 3.10 and Fig. 3.11 that PSNR decreases with increasing noise density. The value of PSNR for $W_s = 3$ is the highest for lower noise densities; however, as the noise density increases, PSNR decreases rapidly and goes below the value corresponding to the window sizes 5, 7 and 9. We choose $W_s = 5$, since PSNR is the highest for this this value of window size in the range of noise density considered. The PSNR is higher for higher values of B and is the highest with ASMF, where the median filtering is performed using all bits of the pixel values.

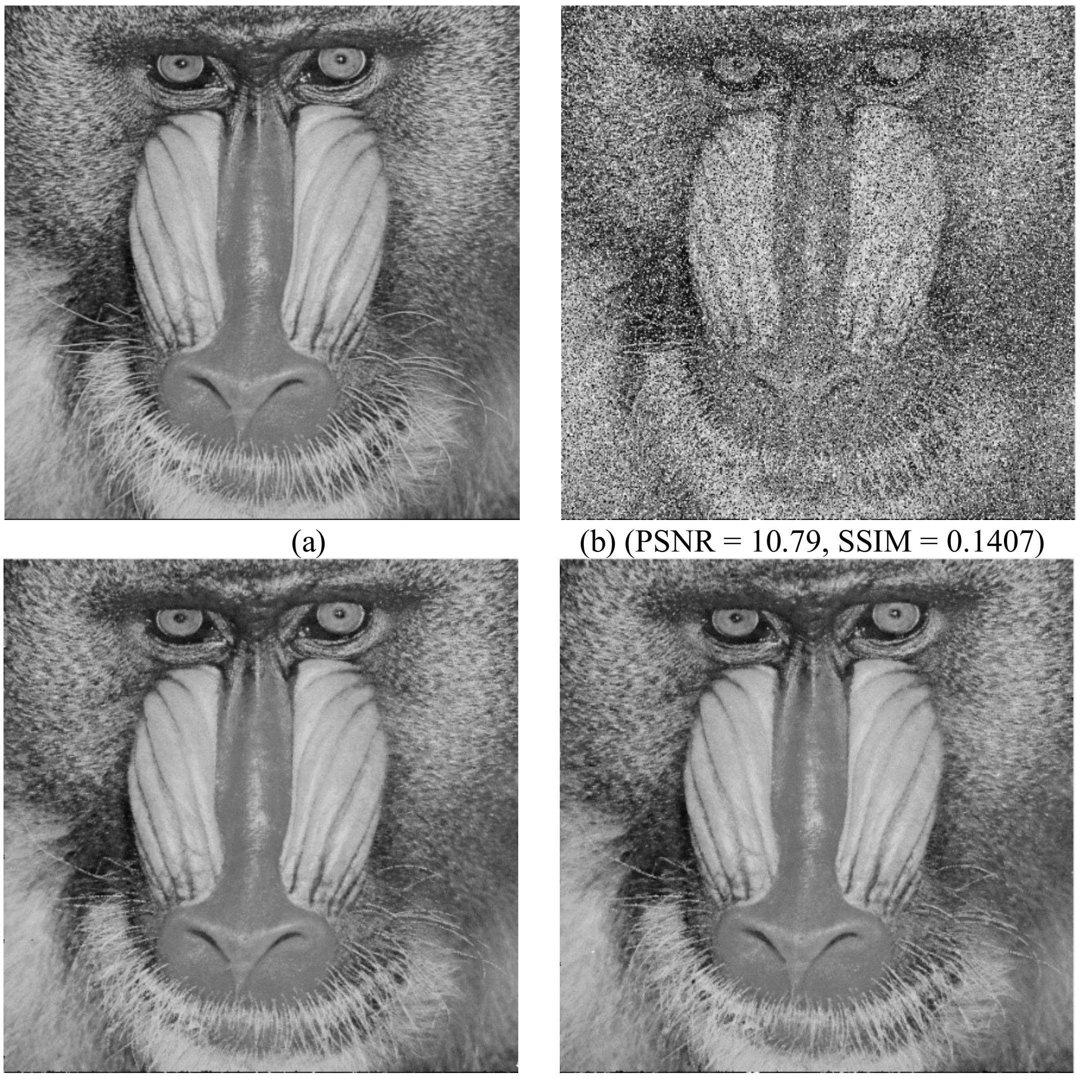
Table 3.1 shows the values of PSNR and SSIM for some of the images in the database [58] and a single frame of each of the six videos selected from the database in [59]. These results are obtained with $W_s = 5$, 30% noise density and different values of B.

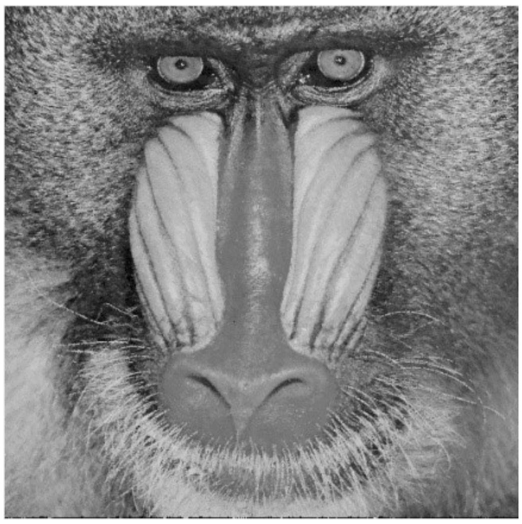
Table 3.1: PSNR and SSIM for images and video frames from the databases in [58] and [59]

Image/Video	**PSNR**				**SSIM**			
	ASMF	**Proposed HAMF**			**ASMF**	**Proposed HAMF**		
		$B = 4$	$B = 3$	$B = 2$		$B = 4$	$B = 3$	$B = 2$
Baboon	25.42	25.17	24.75	23.41	0.8650	0.8457	0.8233	0.7707
Barbara	27.33	27.15	26.58	25.02	0.9066	0.8878	0.8648	0.7825
Gold hill	32.48	31.62	30.36	27.46	0.9205	0.8930	0.8502	0.7526
Lena	33.71	31.88	30.88	28.12	0.9500	0.9268	0.9067	0.8049
Peppers	33.10	32.48	31.28	27.81	0.9453	0.9187	0.8945	0.7879
Four People	32.41	31.88	30.85	28.09	0.9667	0.9412	0.9143	0.8385
Kris and Sara	32.38	32.11	31.19	27.80	0.9765	0.9581	0.9096	0.7584
People	40.79	38.05	35.58	30.33	0.9861	0.9497	0.9081	0.7780
Tennis	26.49	26.29	25.85	24.94	0.8482	0.8349	0.8110	0.7669
Vidyo 1	35.65	34.77	33.26	29.06	0.9759	0.9538	0.9250	0.7929
Vidyo 4	35.50	34.78	33.22	28.70	0.9748	0.9538	0.9154	0.7591

The PSNR and SSIM values for the images and video frames filtered using ASMF are also included in this table for the purpose of comparison. It is seen from this table that the quality of the filtered images and video frames in terms of PSNR and SSIM increases with increasing value of B. The values of PSNR and SSIM for the images filtered using the algorithmic level implementation of the proposed HAMF with $B = 4$ is very close to the corresponding PSNR and SSIM values obtained using ASMF.

For qualitative analysis, we consider the original *baboon* image shown in Fig. 3.12 (a) and its corrupted version with 30% salt-and-pepper noise in Fig. 3.12 (b). Fig. 3.12 (c) shows the image filtered using ASMF with $W_s = 5$, whereas Figures 3.12 (d), (e) and (f) show the filtered images using the proposed technique with $W_s = 5$ and $B = 4$, 3 and 2, respectively. The corresponding values of PSNR and SSIM are also included below the figures.


(a) (b) (PSNR = 10.79, SSIM = 0.1407)
(c) (PSNR = 25.42, SSIM = 0.865) (d) (PSNR = 25.17, SSIM = 0.8457)


(e) (PSNR = 24.75, SSIM = 0.8233)

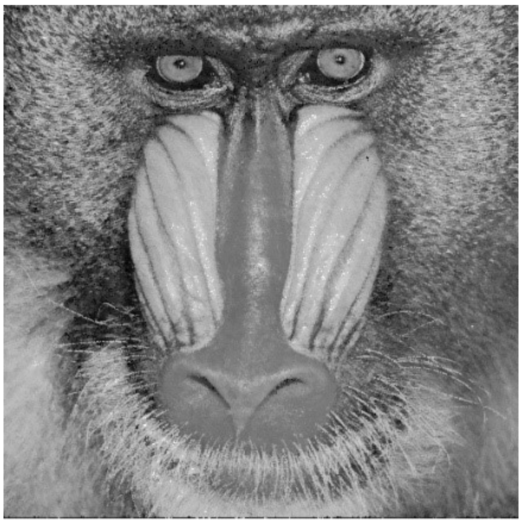

(f) (PSNR = 23.41, SSIM = 0.7707)

Figure 3.12 Results on *baboon* image (a) Original Image (b) corrupted image with 30% salt-and-pepper noise (c) filtered image using ASMF, and filtered images using the proposed technique with (d) $B = 4$, and with (e) $B = 3$ and (f) $B = 2$.

As mentioned before, the optimal window size for median calculation is $W_s = 5$. Hence, we implement the proposed HAMF on hardware with $W_s = 5$ and different values of B. The proposed HAMF can be implemented on hardware for higher values of W_s; however, it will result in a degraded hardware performance due to a large area and low operating frequencies. We synthesize the proposed HAMF on hardware with different values of B in order to find its impact on the area and operating frequency. Using the implementation results, we can see if there exists a trade-off between the output image quality and the hardware performance.

3.4.2 Simulation Results

To verify the functionality of the proposed HAMF that calculates an approximate median, we carry out the behavioral simulation of the proposed HAMF implemented with $B = 2$, 3 and 4 by feeding random numbers as the input pixel values to calculate the output. As mentioned earlier W_s is chosen to be 5. Fig. 3.13 shows the results of simulation for HAMF with $B = 2$ and $W_s = 5$, where the input values **d1-d25** are 0x[72, 7e, 14, 23, 24, 25, 32, 36, 38, 37, 48, 41, 00, 5a, 58, 7a, 72, 7c, 83, 5d, 51, 74, 7a, 7c, a9]. The most significant 2-bits of the input values are considered for median calculation which results in a 2-bit median equal to 0x1. Hence, the first element in the input vector with the value of the upper 2-bits equal to 0x1, which is 0x72 in this case, is selected as the approximate median. This value is available at the output port **dout** as shown in the Fig. 3.13. The result of the median appears at the output after 14 clock cycles, which is the latency of the design implemented with 12 pipeline registers in the histogram calculation module of Fig 3.7. Two additional clock cycles are required by the median selection unit, resulting in a total of 14 clock cycle latency.

Fig. 3.14 shows the simulation results of the proposed HAMF implemented with $B = 3$ and $W_s = 5$. The input vector fed to this design is the same as in the previous simulation. However, as this design processes upper 3-bits, the calculated value of the median is 0x2. As a result, the first matching element from the input vector with the value of the upper 3 bits equal to 0x2, which is 0x48 in this case, is selected as the median.

Fig. 3.15 shows the simulation results for proposed HAMF implemented with $B = 4$ and $W_s = 5$. The input vector fed to this design is the same as in the previous simulations. However, as this design processes upper 4 bits, the value of the calculated median is 0x5. As a result, the first matching element from the input vector with the value of the upper 4 bits equal to 0x5, which is 0x5a in this case, is selected as the median.

The actual value of the median of the input vector considered in these simulations is 0x58. However, the calculated median values using $B = 2$, 3 and 4 are 0x72, 0x48, and 0x5a, respectively. This shows that a closer approximation to the actual median value is obtained using a higher value of B.

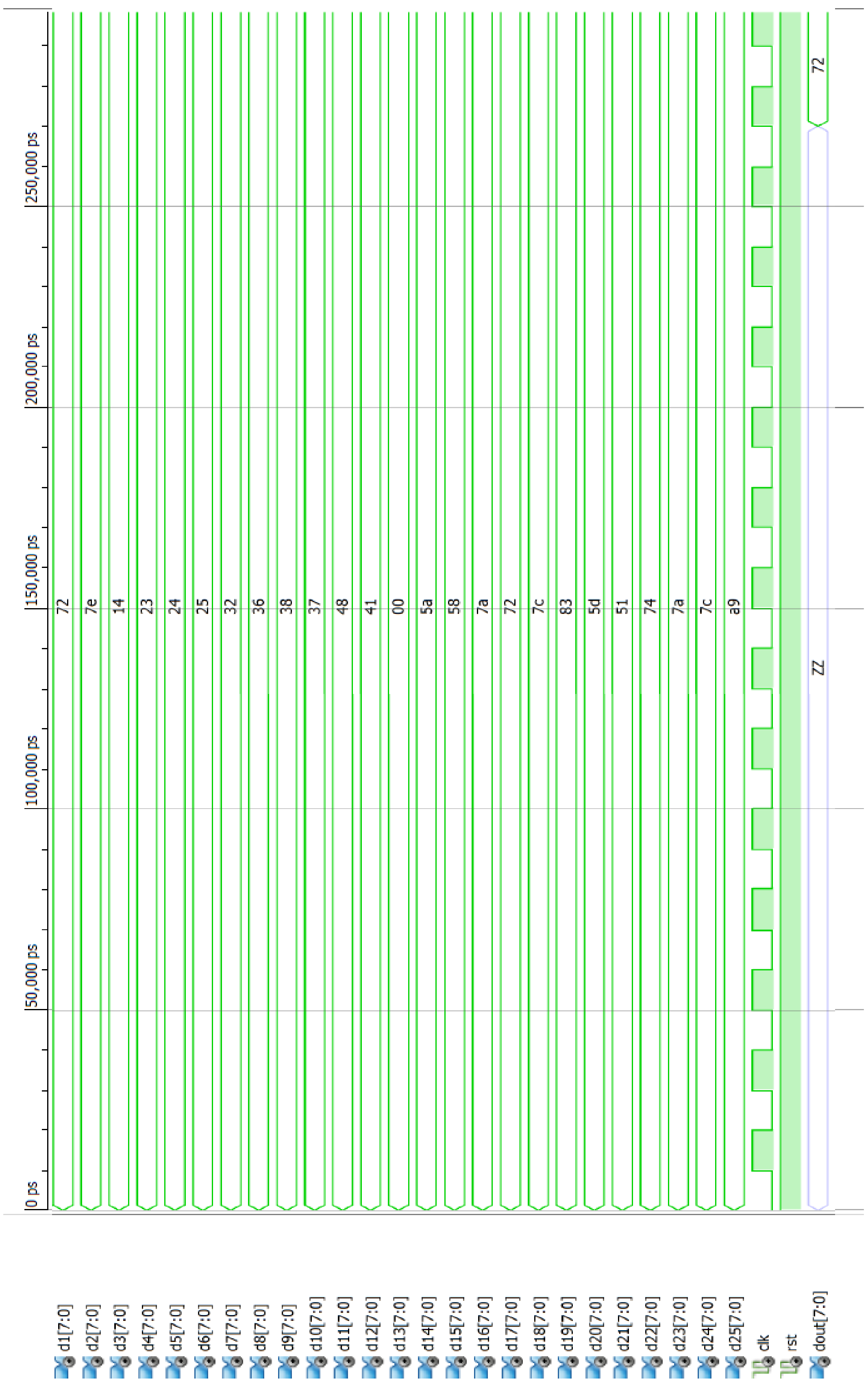


Figure 3.13 Simulation results of the proposed HAMF with $B = 2$, $W_s = 5$.

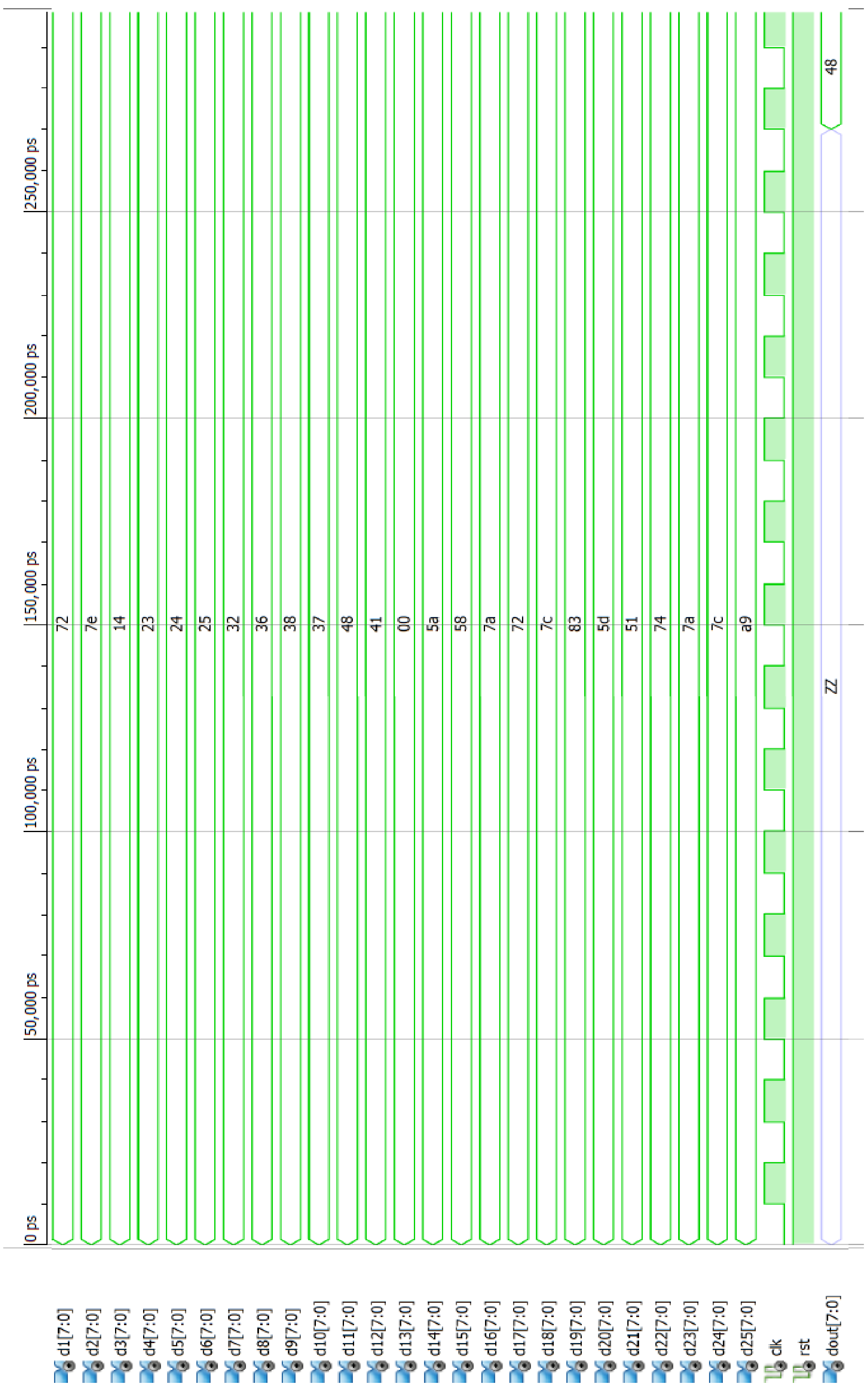


Figure 3.14 Simulation results of proposed HAMF with B = 3, W_s = 5.

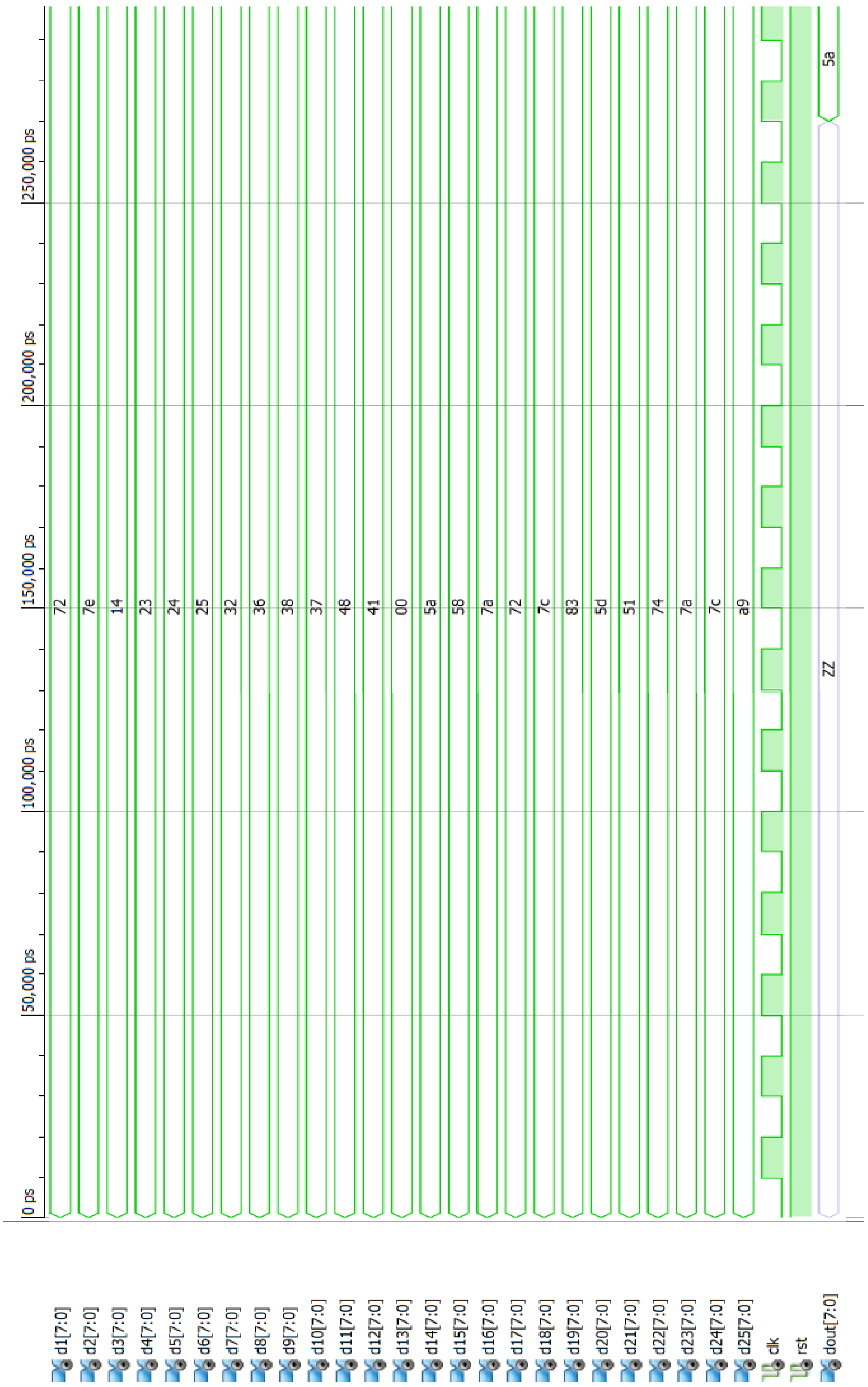


Figure 3.15 Simulation results of proposed HAMF with $B = 4$, $W_s = 5$.

3.4.3 FPGA Implementation of the Proposed HAMF

The proposed HAMF is implemented on 3 different generation of Xilinx FPGAs, namely, Virtex-II, Virtex-6 and Zynq-7, with W_s = 5 and different values of B (2, 3 and 4). Different FPGAs are chosen to analyze the effect of implementation of the proposed architecture on older (Virtex-II and Virtex-6) as well as the newer (Zynq-7) generation FPGAs, in terms of the hardware performance. The results of the implementation of a non-pipelined version of the proposed architecture with different values of B are presented in Table 3.2. In this table, the area is reported in terms of the slice-LUTs and slice-registers. The operating frequency is calculated from the maximum combinational delay as reported by the synthesis tool.

Table 3.2. Hardware implementation results of the proposed HAMF implemented without pipelining.

FPGA Device	**# Slice LUTs**			**# Slice Registers**			**Frequency (MHz)**		
	$B = 2$	$B = 3$	$B = 4$	$B = 2$	$B = 3$	$B = 4$	$B = 2$	$B = 3$	$B = 4$
Virtex-2	301	499	1050	541	881	1838	25.8	20.2	16.9
Virtex-6	271	385	705	-	-	-	98.6	61.4	36.9
Zynq-7	271	385	705	-	-	-	110	68.5	41.5

It is observed from Table 3.2 that the area required for implementation in terms of slice-LUTs increases with increasing value of B for the FPGAs considered. In the case of Virtex-II, slice-registers are required in addition to slice LUTs, which is due to the underlying architecture of Virtex-II. The operating frequency decreases with increasing value of B. This is a result of cascaded configuration of LUTs during design implementation. The cascaded configuration is required for the implementations that have a larger number of slices.

The operating frequency of the design on all the devices considered is low and cannot satisfy the real-time throughput requirements of the FHD video filtering. This is a result of the large combinational delay due to the absence of the pipeline registers. In order to increase the operating frequency of the design, we implement pipelined versions of these by introducing pipeline registers. Introduction of these pipeline registers increases the operating frequency of the design, but also increases the slice-register utilization at the same time. Hence, we implement the pipelined version of the proposed HAMF with different number of pipeline registers inside the adder stage, namely, stage II (Fig. 3.8) of the histogram-based median calculation block, to evaluate the effect on the register utilization and operating frequency. Fig. 3.16 shows the LUT utilization and Fig. 3.17 shows the registers/FF utilization for B = 2, 3 and 4 and L = 8, 10 and 14. It is seen from Fig. 3.16 that the LUT utilization increases only slightly with increasing pipeline latency, irrespective of the device used. Since Virtex-II is an older generation device, its LUT utilization is higher than that of Virtex-6 or Zynq-7, which have nearly the same amount of utilization. The LUT utilization increases significantly with B, due to the logic involved in processing more bits with higher values of B.

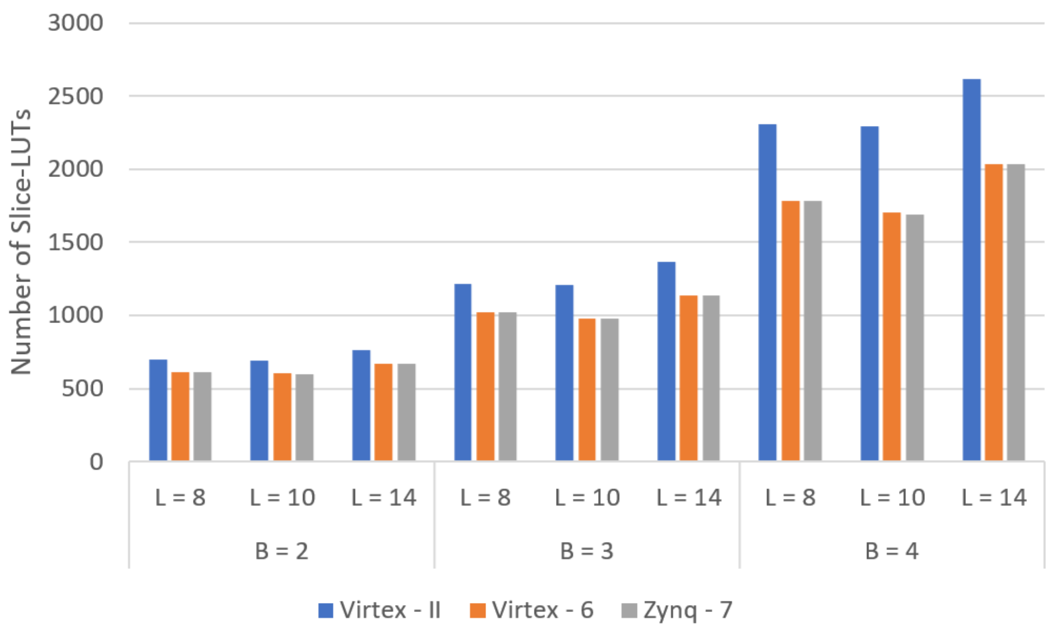


Figure 3.16 Slice-LUT utilization for different values of B and L

It is observed from Fig. 3.17 that the register/FF utilization increases with increasing value of B or L. The register utilization in the case of Virtex-2 implementation is greater than the utilization in the case of Virtex-6 and Zynq-7 implementation. The maximum register utilization is in the case of L = 14 and B = 4, since a large number of registers are required as a consequence of a greater number of pipeline stages and increased value of B.

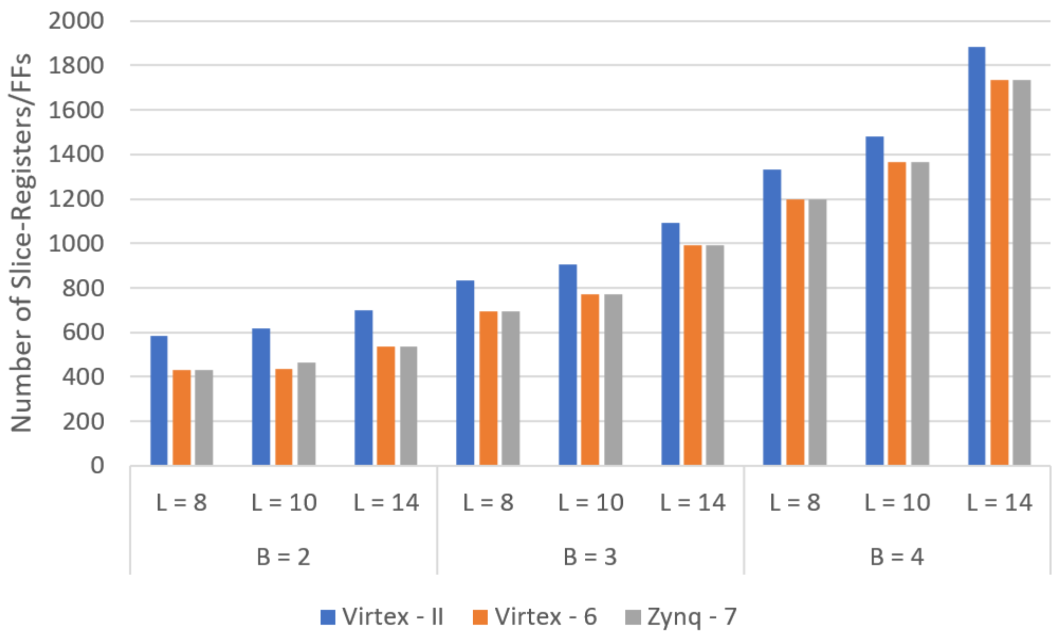


Figure 3.17 Slice-Register/FF utilization for different values of B and L

Fig. 3.18 shows the operating frequency of the design synthesized on the three FPGAs for different values of B and L. It is observed from this figure that the operating frequency increases with L, since the cascaded combinational blocks are divided into smaller blocks (see Fig. 2.12). With a higher pipeline latency, the delay of each combinational block is reduced, resulting in a higher frequency of operation. The operating frequency is calculated from the maximum combinational delay, as reported by the synthesis tool.

Fig. 3.19 shows the power as estimated by the Xilinx Xpower analyzer tool when the three FPGAs are used for implementation with different values of B and L. The operating frequency of the clock of the each of the designs is specified in the Xpower analyzer tool. It is seen from this figure that the power dissipation increases with increasing pipeline latency, and this is due to the higher operating frequency as seen from Fig. 3.18. There is a slight increase in the power with increasing value of *B*, irrespective of the device used. Also, the power requirements of Virtex-6 device is much higher than that for Virtex-II or Zynq-7 devices and is so mainly because of the underlying technology.

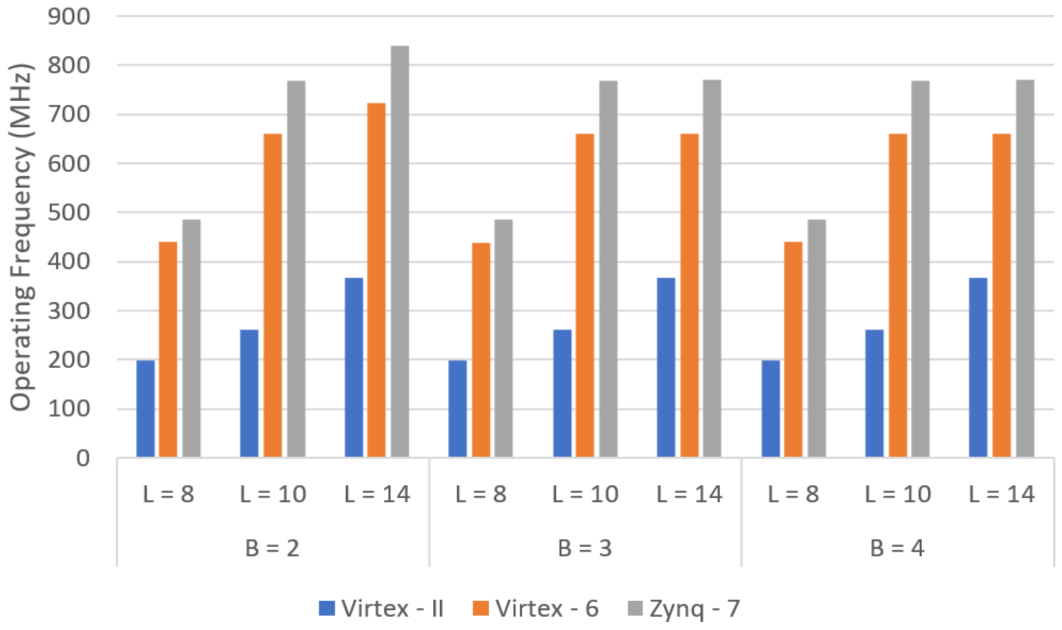


Figure 3.18 Operating frequency calculated from the maximum combinational delay for different values of *B* and *L*.

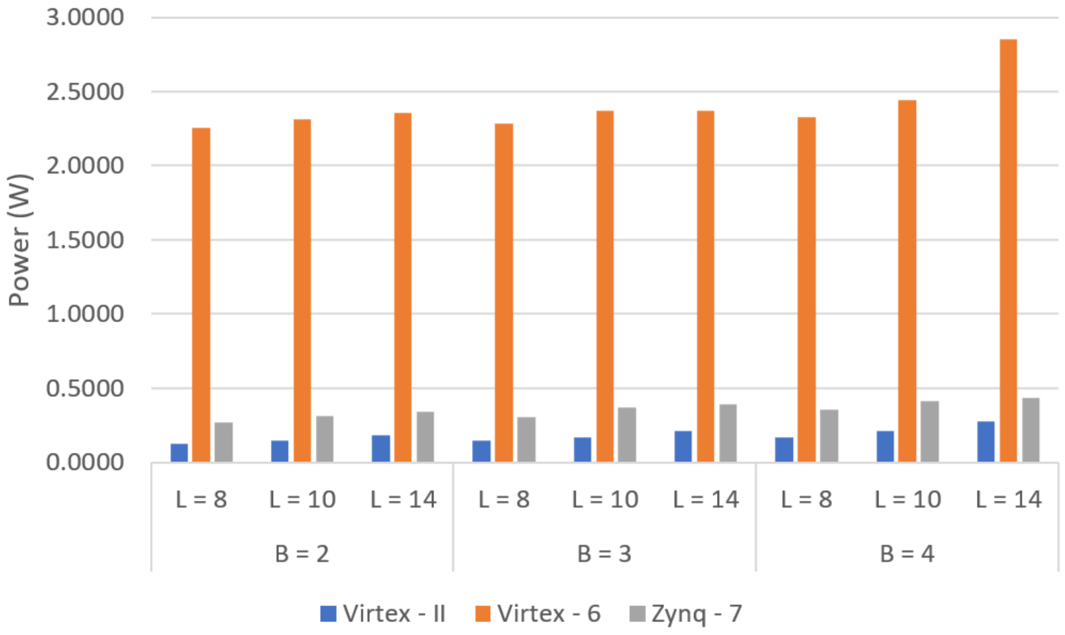


Figure 3.19 Power estimated by the Xpower analyzer for different values of *B* and *L*.

Although the proposed HAMF implemented with $B = 4$ has a lower operating frequency compared to that with $B = 2$ and $B = 3$, it is still acceptable for filtering FHD videos in real-time. As mentioned earlier, the quality of the image filtered using the algorithmic level implementation of the proposed HAMF with $B = 4$ is better than those filtered with $B = 2$ and $B = 3$. Hence, we calculate the maximum operating frequency of the proposed HAMF implementation on hardware with $B = 4$, by specifying the timing constraints in the implementation tool. We carry out this on all the FPGAs devices considered for different values of L.

Using the maximum operating frequency, the throughput of the proposed HAMF hardware implementation is calculated using Eq. (2.5). The frame size F_s for an FHD video is 1920×1080 and since all the implementations are for a single core architecture, $\nu = 1$. Fig. 3.20 shows the throughput of the proposed HAMF hardware implementation in FHD video frames per second on the three FPGA devices considered, for different values of L.

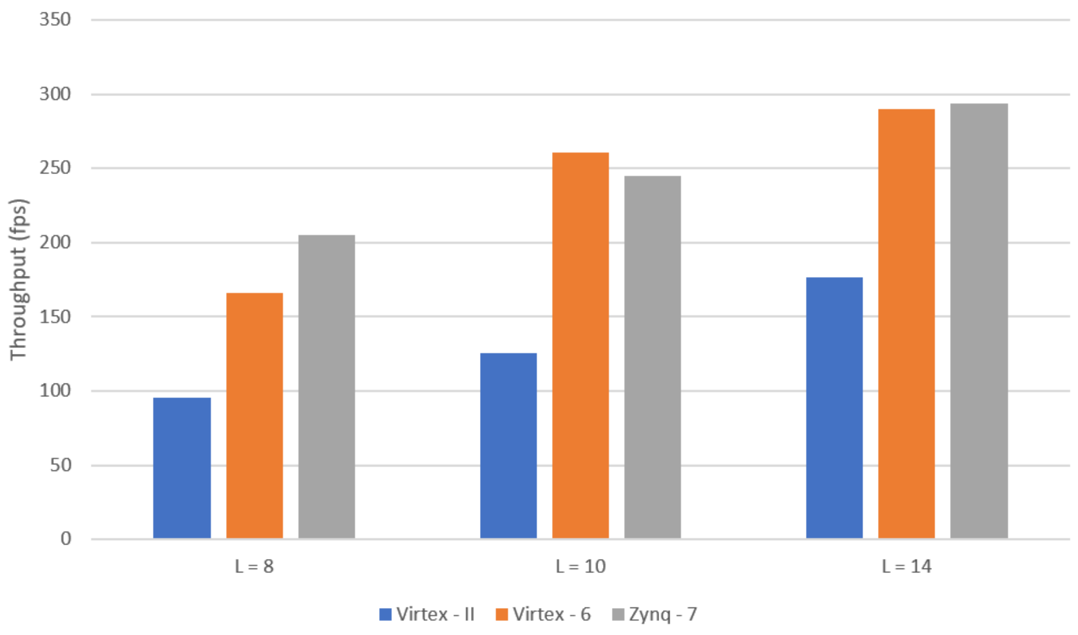


Figure 3.20 Throughput for FHD videos for different values of L.

It is observed from Fig. 3.20 that all the hardware implementations of the proposed HAMF with different pipeline latencies can satisfy the real-time requirements of FHD video filtering at the rate of 60 fps. The throughput increases with increasing pipeline latency and is higher for newer generation FPGAs (Virtex-6 and Zynq-7) than that provided by the implementation on Virtex-II. It must be also noted that the increase in the throughput is at the cost of the area, as per the discussion in the previous sections.

To compare the proposed HAMF implementation with ASMF in terms of the hardware performance, we implement ASMF with full synchronous pipelining with W_s = 5 on the three FPGAs considered. Table 3.3 shows the hardware performance of ASMF in terms of slice-LUTs, slice-registers/FFs and operating frequency on these FPGAs. For the purpose comparison, the corresponding performance values for the proposed HAMF implemented with B = 4, L = 14 and W_s = 5 are also presented in this table. It is seen from this table that the proposed HAMF implementation provides a performance that is significantly superior to that ASMF implementation on all the three FPGAs.

Table 3.3. Hardware implementation results of the proposed HAMF and ASMF.

FPGA Device	**ASMF**			**HAMF (B = 4 and L = 14)**		
	LUTs	**Registers/FFs**	**Frequency (MHz)**	**LUTs**	**Registers/FFs**	**Frequency (MHz)**
Virtex-2	5436	3552	246	2621	1885	366
Virtex-6	4953	3552	455	2032	1736	661
Zynq-7	4953	3552	583	2032	1736	770

3.5 Summary

By combining multiple techniques, namely, bit-plane-slicing, decision based median filtering and histogram-based median calculation, a hybrid architecture for median filtering is proposed and then implemented on different generation FPGAs. The hardware implementation of the proposed HAMF can operate at very high frequencies when implemented with super-pipelining. The proposed HAMF processes only the most significant B-bits of the pixel values. As a result, the quality of the images filtered by the algorithmic level implementation of the proposed HAMF is slightly lower than that using the ASMF. Through an analysis of the results of implementation of the proposed HAMF with $B = 4$ and different values of L, it has been shown that a trade-off could be achieved between the area and the throughput. The proposed HAMF implementation with $B = 4$ and $L = 14$ provides a significant improvement in terms of the area and operating frequency compared to that of the ASMF implementation. The hardware implementation of the proposed HAMF satisfies the real-time requirements of FHD videos at the rate of 60 fps; however, it is very slow to be considered for real-time filtering of UHD videos. The frame resolution of a 4K UHD video is four times that of an FHD video frame and hence, window sizes higher than 5 may be needed for filtering UHD videos. In order to fulfil the needs of real-time UHD video filtering, we provide an efficient architecture of the hierarchical histogram-based median filter and its FPGA implementation in the next chapter.

Chapter 4

Design and Implementation of the Hierarchical Histogram-based Median Filter

4.1 Introduction

The hybrid architecture for median filtering presented in Chapter 3 can operate at high frequencies and fulfil the real-time throughput requirements of FHD videos and is area efficient when implemented on different generation FPGAs. However, the hardware implementation of the proposed HAMF is for a fixed window size and the optimization achieved in its implementation is at the cost of filtered image quality. The histogram module used in HAMF uses cascaded adders to generate the histogram values. As a result, the operating frequency of the proposed HAMF will decrease with increasing window size. In many applications, output image quality is as important as the hardware performance, and a compromise in image quality may not acceptable. Frame resolutions have increased from FHD to UHD in order to provide better quality of images and there are no hardware implementations of median filters, that can satisfy the real-time requirements of UHD videos for large window sizes. Hence, in this chapter we present a hardware-based architecture for median filter and its FPGA implementation that can satisfy the real-time throughput requirements of UHD videos for window sizes up to 15 [60]. The proposed hardware architecture is based on the hierarchical histogram-based median filter (HHMF) [33], which uses histogram-based operations for computation of the median. As opposed to the histogram-based median filter such as [32], HHMF calculates the histogram of the input data in two steps.

In a histogram-based method, each element in the input array consisting of N elements, is read and the count of the corresponding bin is incremented. For 8-bit input values, there are 256 such bins. After all the input values are read, values of bins starting from bin_0 are added together until the result of addition is greater than $(N - 1)/2$ and the bin whose value was added last is the median. This technique is better than the sorting-based median calculation, as the number of stages of calculation does not depend on N [28]. However, hardware implementation of histogram-based median filter results in a lower speed due to involved memory operations and cyclic reading of input pixels [29].

As mentioned earlier, the algorithms implemented on GPU for high performance such as in [31] and [32] are based on histogram calculation and hence the histogram-based technique can be used to create a winning architecture for real-time filtering of UHD videos. A median filter designed using the conventional histogram-based technique is almost impossible to be implemented on hardware without the use of memory bins, since for large window sizes, the resource requirements will exceed that available in FPGAs. However, if the memory bins are employed in hardware implementation as in [30], the throughput will get be very low.

The HHMF first processes the upper-half most-significant bits (MSBs) of the input data and then the lower-half least-significant bits (LSBs) to obtain the median in two steps. The HH median filtering algorithm presented in [33] is implemented on GPUs and provides a very high throughput. HH median filtering is based on the same bin approach that is used in histogram-based median filter, the difference being that the number of bins is reduced due to the reduction in the range of input values, since each pixel is split into two parts. The number of storage bins required for processing all the bits of pixel values is 2^W (= 256 for $W = 8$), where W is number of bits required for representation of the value of a pixel. However, if the pixel values are split into 2 parts, each of $W/2$ bits, the number of bins is reduced to $2 \cdot 2^{(W/2)}$ (=32 for $W = 8$). Due to the reduced number of bins, it becomes feasible to implement a hardware architecture that provides higher speed. Registers can be used to serve the purpose of bins instead of a ROM based storage.

The performance of an algorithm implemented on a GPU is heavily dependent on the GPU architecture and more importantly, on its count of processing cores. Although GPU is a common platform for computationally demanding applications, the cost of GPUs is higher than a typical FPGA (e.g. Artix-7) used in DSP applications. For low cost products, a GPU may not be a desired solution as supporting peripherals are needed in addition to the GPU itself. High power consumption of GPUs compared to FPGAs [61] makes them unsuitable for many applications. At this point it is worthwhile mentioning that multi-processor system on chip (MPSoC) architectures are becoming more popular for many real-time implementations of applications, such as 4K video processing. These devices provide a very high degree of flexibility due to their hardware-software co-design feature.

The present architectures of median filters implemented on FPGAs cannot be scaled for UHD applications due to their dependency on sorting-based operations. Moreover, the architectures that implement median filtering using non-sorting-based techniques use memory as bins to store histogram count, which results in a very low frequency of operation due to timing overheads added by memory read and write operations. Such architectures cannot satisfy the real-time requirements of a UHD video system. Non-sorting-based histogram calculation algorithms, such as HHMF, can satisfy the throughput requirements of a UHD video system when implemented on GPUs. As discussed above, GPUs have their own limitations such as cost and power requirements. Designing an FPGA-based architecture for HHMF is a challenging task, since such an architecture may face limitations such as scalability and ability to satisfy real-time requirements for applications like UHD videos.

We propose, for the HH-based median filter, an efficient hardware architecture that can be mapped to any programmable logic device. The proposed hardware architecture can perform median filtering operations at a very high speed for window sizes up to 15. The proposed architecture implements full-synchronous pipelining to achieve maximum frequency of operation. A novel approach to histogram calculation and accumulation module is used to ensure full-synchronous operation of the pipeline. We implement, for the first time, a hardware architecture for HHMF that provides results that are superior to that of NVIDIA Tesla and Parker SoC GPUs in terms of the throughput. The proposed hardware architecture also satisfies the pixel clock requirement (600 MHz) for a UHD video system when mapped to Xilinx MPSoC by implementing a single pipelined core. We analyze the hardware performance of the proposed median filter implemented with 1, 2 and 4 cores on Artix-7 FPGA for different window sizes and compare the throughput performance with that of the HHMF algorithm on GPUs.

The chapter is organized as follows. Section 4.2 discusses the HH median filtering algorithm. Section 4.3 presents the proposed hardware architecture for HHMF along with its analysis. An FPGA implementation of the proposed architecture is presented in Section 4.4. Section 4.5 presents the hardware implementation results and comparative analysis. Summary of the chapter is presented in section 4.6.

4.2 Hierarchical Histogram Median Filter

The HH-based median filtering algorithm calculates the median of the input window in two parts. The steps involved in the median calculation using HHMF are shown in Algorithm 1. The higher nibble of the median, *MH* is calculated using steps 1-3, while the lower nibble, *ML* is calculated using steps 5-7. The value H_{i-1} calculated in Step 4 is used in step 7 to calculate *ML*.

Algorithm I: Hierarchical Histogram-Based Median Calculation

```
Number of input data → $N$ (odd), Range of a nibble (4-bits) → 0 to 15
1: Extract the higher nibble for each of the $N$ values.
2: Calculate the histogram values $H_0 – H_{15}$ for the extracted nibbles.
3: Add the histogram values $H_0 – H_{15}$ successively until the addition exceeds $(N - 1)/2$.
   Index $i$ of the last $H_i$ added is the higher nibble $MH$.
4: Store the value of $H_{i-1}$ in accumulator $\boldsymbol{A}$.
5: Extract the lower nibble for each of the $N$ values, for which its higher nibble is equal
   to $MH$.
6: Calculate the histogram values $H_0 – H_{15}$ of the extracted nibbles.
7: Add the histogram values $H_0 – H_{15}$ successively to $\boldsymbol{A}$ until the addition
   exceeds $(N - 1)/2$. Index $i$ of the last $H_i$ added is the lower nibble $ML$.
8: Combine $MH$ and $ML$ to obtain the median $M$.
```

We now illustrate the above algorithm by an example. Fig. 4.1 shows the processing of sample input data using HH median filtering consisting of $N = 9$ data values of a square window of size $W_s = 3$. In the first step, all the higher nibbles of the input data are extracted for $H_0 – H_{15}$ calculation. Histogram values starting from H_0 are added in the accumulator, and after the addition of $\boldsymbol{H_7}$, the result of accumulator is greater than 5. As a result, MH is **0x7**. Accumulator of the second block is initialized with H_6. For each input data, if the value of the higher nibble is equal to 0x7, then the corresponding lower nibble is retrieved. Histogram values $H_0 – H_{15}$ are calculated for these values and added to the accumulator successively. Addition of $\boldsymbol{H_3}$ results in the accumulator value to be greater than $(N – 1)/2$. As a result, ML is **0x3.**

4. 3 Proposed Hardware Architecture

This section presents the details of the proposed hardware architecture for implementing HHMF. Features of the proposed hardware architecture, which makes it feasible for real-time UHD video filtering, are also highlighted in this section. The proposed hardware architecture is analyzed for implementation feasibility with respect to the area and latency.

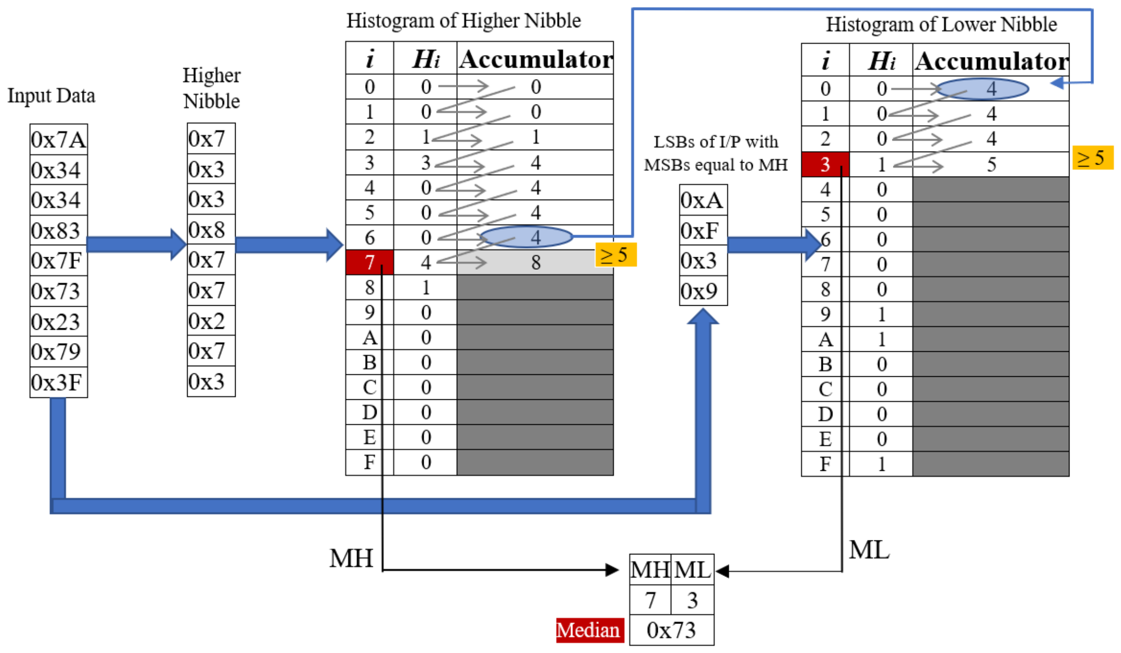


Figure 4.1 An example of the HHMF Algorithm.

4.3.1 Architecture Overview

In the median calculation using the HH method, the median is calculated in two parts, as illustrated in the previous section by an example. After *MH* calculation, for each element in the input data, only if the value of the higher nibble is equal to the value of *MH*, then the corresponding lower nibble is retrieved. If the input vector is read again after the calculation of *MH*, the purpose of a pipelined architecture is defeated. Hence, we design an architecture in which there is no more than one reference to the input data vector. This reference is in the first cycle of the operation, where all the input elements of the vector are read and buffered.

Fig. 4.2 shows the proposed hardware architecture which consists of two blocks, namely, HMH (**H**istogram of **M**edian **H**igh) and HML (**H**istogram of **M**edian **Low**), the former corresponding to the higher and the later to the lower nibble calculation of the median. The input value of the comparator of HMH is C_h and is calculated as $[(N - 1)/2] + 1$, where N is the total number of input elements. The input value of the comparator of HML is C_l, which is calculated as $[(N - 1)/2] + 1 - B_{MH-1}$, where B_{MH-1} is the count value of bin MH - 1.

The pipelined buffer, shown above the pipelined HMH block in Fig. 4.2, stores the input data in a shift-register, which has the same latency as that of the pipelined HMH block. As soon as the value of *MH* is calculated, the HMH block feeds the desired values from the pipelined buffer to the HML block along with the value of C_l. The value of *MH* is passed to another pipelined buffer that is shown below the pipelined HML block in Fig. 4.2 and has the same latency as that of the HML block. Thus, both *MH* and *ML* are available at the output at the same time. As a result, the pipeline can work at full efficiency without any stall or delay.

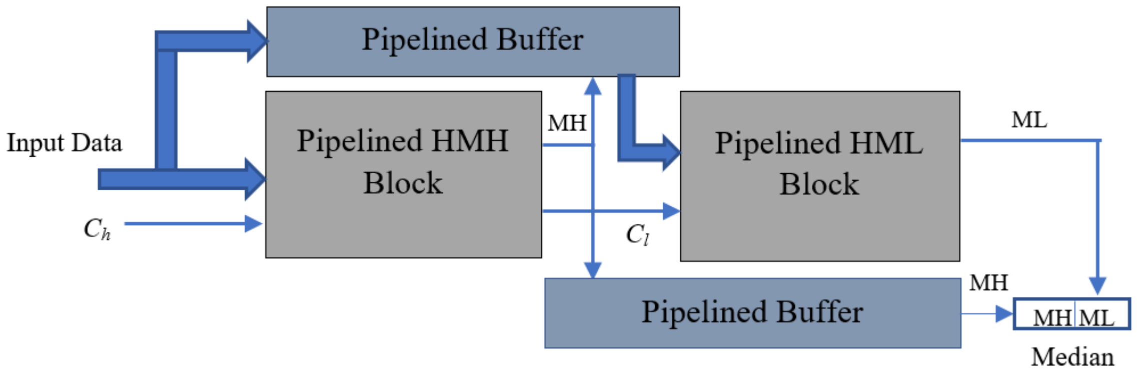


Figure 4.2 Proposed hardware architecture for HHMF.

4.3.2 Architecture for HMH and MHL Blocks

Fig. 4.3 shows the architecture of the HMH and HML blocks. First, the input data is fed to a comparator module that performs parallel comparison of all the input data elements to all the possible values in the range of 4-bit numbers, i.e., 0 to 15. The inputs for the HMH block are the higher nibbles of all the N input values. However, for the HML block, K ($\leq N$) input values are fed from the pipelined buffer and the remaining $N - K$ input values are forced to assume the highest value, 15, in the range 0 to15, to ensure proper median calculation.

The output of the comparator module is a matrix consisting of N×16 binary values. The histogram calculation module performs the addition of all the elements in each column of this N×16 matrix to generate a 1×16 matrix. This is the histogram of the input data and represents the count of each element in the range (0 to 15) of 4-bit numbers. The output of the histogram calculation module is fed to the accumulate-and-compare (AC) module, where the histogram values are accumulated starting from the value of H_0. This module calculates the MH or ML value by comparing the accumulated histogram values with C_h or C_l. Compared to the HML block, the HMH block contains an additional module (shown in dotted lines in Fig. 4.3) that calculates C_l, which is the comparator value for the HML block.

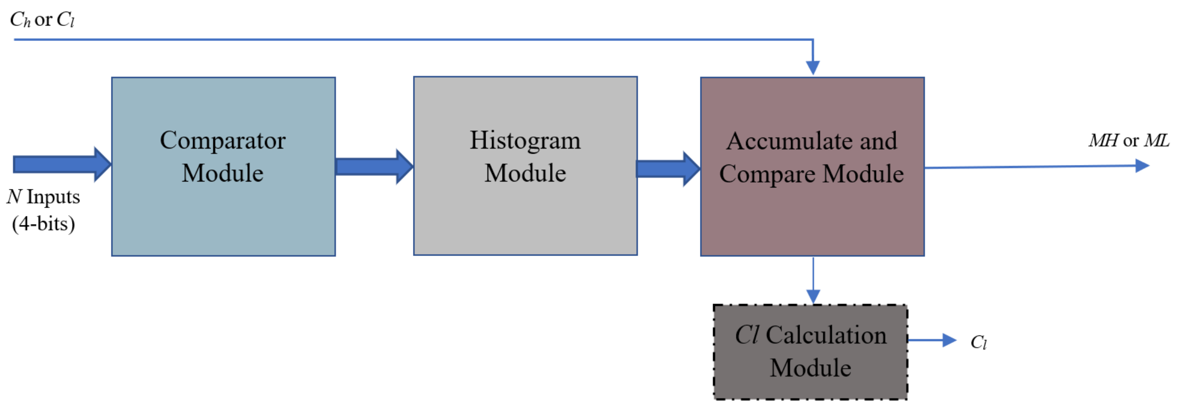


Figure 4.3 Architecture of the HMH/HML block.

A) Comparator Module

The comparator module consists of N units, each containing 16 comparators; one such unit is shown in Fig. 4.4.

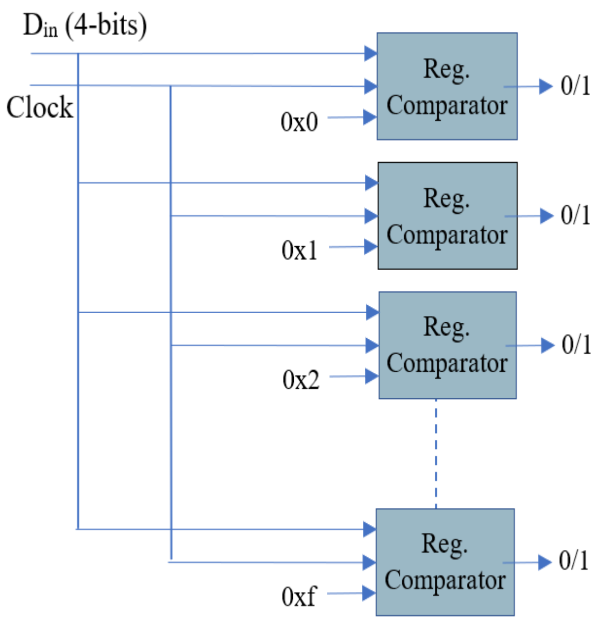


Figure 4.4 A Comparator Unit.

Each comparator in this unit compares the input element to each value in the range 0 to 15. Although the size of the comparator module increases with *N*, the delay due to comparison in the proposed hardware architecture is constant, irrespective of the value of *N*. Each comparator is clocked with the system clock, which fulfils the need of a pipelined architecture.

B) Histogram Module

The histogram module adds all the i^{th} bits (i = 0 to 15) in all the *N* units of the comparator module to generate the histogram of the data. This is accomplished by 16 adder units, operating in parallel, each adder unit dedicated to a particular value of *i*. Each adder unit adds *N*-bits to generate a $\lceil \log_2 N \rceil$-bit value. The structure of an adder unit is shown in Fig. 4.5, which uses registered adders. In contrast to a cascaded serial adder that requires *N*-cycles to perform the addition of *N*-bits, the proposed structure requires only $\lceil \log_2 N \rceil$ cycles to compute the output.

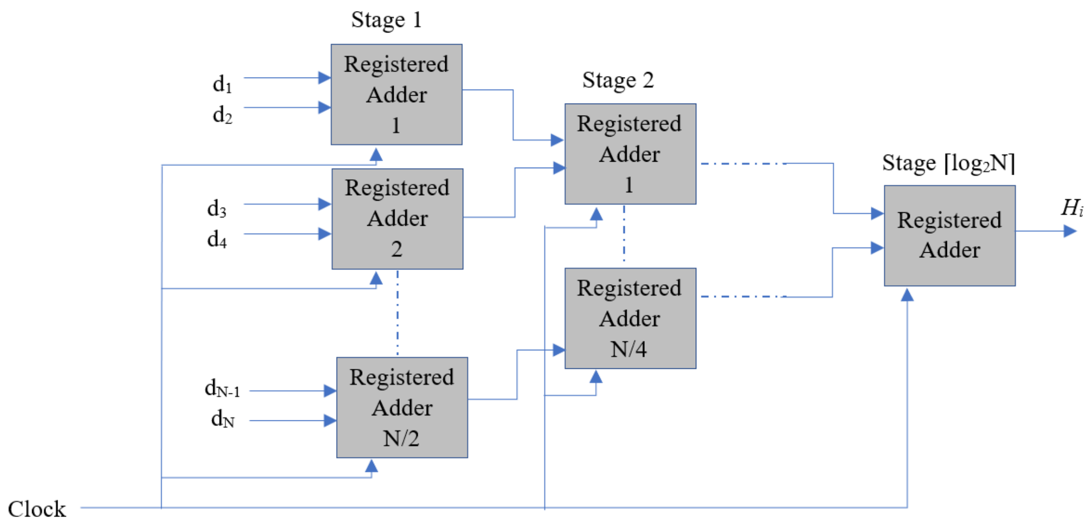


Figure 4.5 Adder Unit.

C) Accumulate-and-compare (AC) module

After the generation of the histogram, next step is to determine the values of *MH* and *ML* using the blocks HMH and HML, respectively. To calculate these values, the AC module adds the histogram values H_0 to H_{15}. Accumulation of these 16 values without the use of a pipelined architecture will result in a large delay. If the comparison with C_h or C_l is performed after each addition, then the output of the comparison may become true during any stage of the accumulation process depending on the values of the input vector. As a result, the pipeline cannot operate in synchronous with the other modules/blocks.

To achieve a fully synchronous pipeline, each clocked adder of the AC module at position P (0-15) has P cascaded registers at its input and $(15 - P)$ cascaded registers at its output, as shown in Fig. 4.6. Once the result of the accumulation becomes available (after 16 clock cycles), all the values accumulated by the various stages are compared at the same time with C_h (C_l) to generate a row vector representing the output of each of the comparators of the AC module shown in Fig. 4.6. The first occurrence of ‘1’ in this vector indicates the value of

MH (*ML*). This value is selected by the *MH/ML* selector unit. For example, if the output vector for *MH* is [0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1], then MH is 0xB.

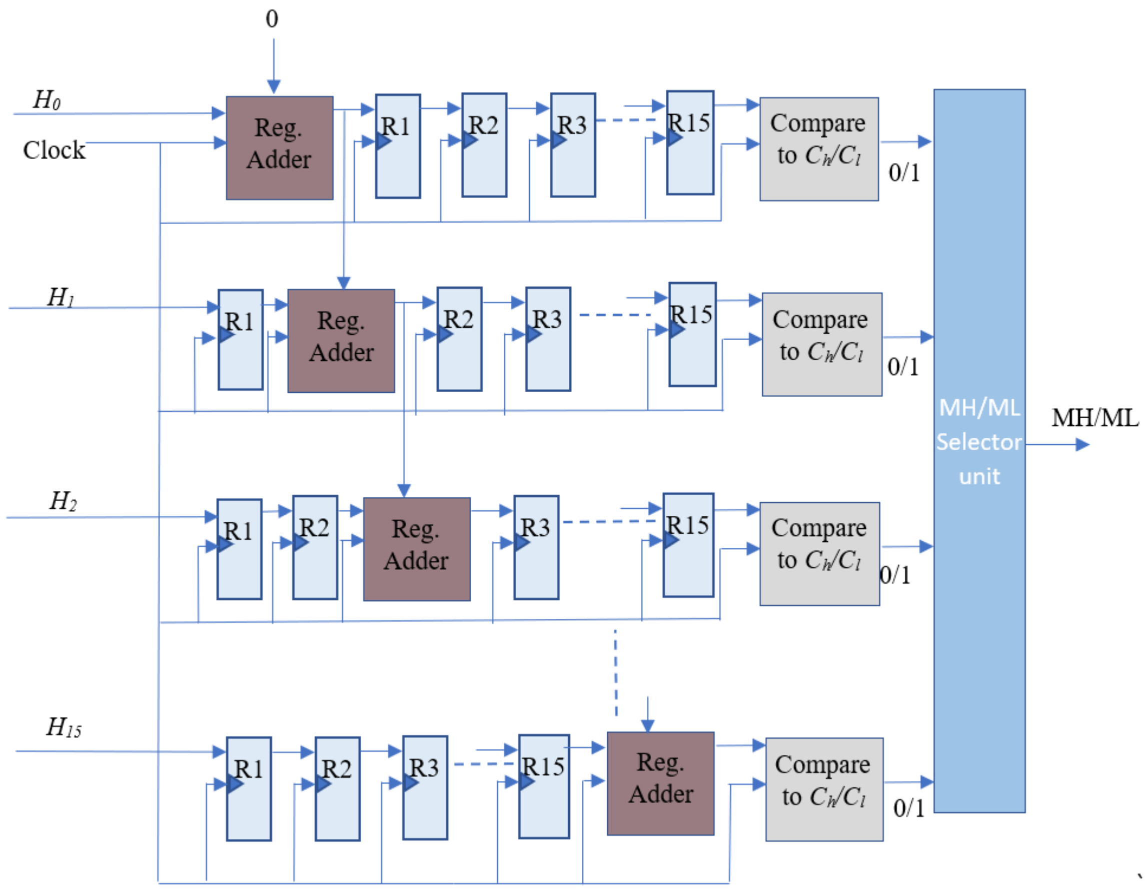


Figure 4.6 Accumulator and compare module.

4.3.3 Architecture Analysis

Various components used in the proposed hardware architecture are adders, comparators and registers. The adders and comparators are used for processing, while the registers are used for pipelining. In this section, we analyze, for the hardware implementation feasibility, the number of components required as the window size, W_s, increases. We also analyze the latency of the proposed pipelined architecture for different window sizes in terms of the number of clock cycles.

A) Area utilization in terms of the number of adders and comparators

Fig. 4.7 shows the number of adders and comparators required by the proposed architecture for different window sizes for a single core fully-synchronous pipelined architecture. A simple divide and conquer approach [62] can be used to simultaneously process an arbitrary number, *v*, partitions of the input image or video frame using *v* cores of the proposed architecture to increase the throughput by a factor of *v*. This will increase the area utilization by a factor of *v*.

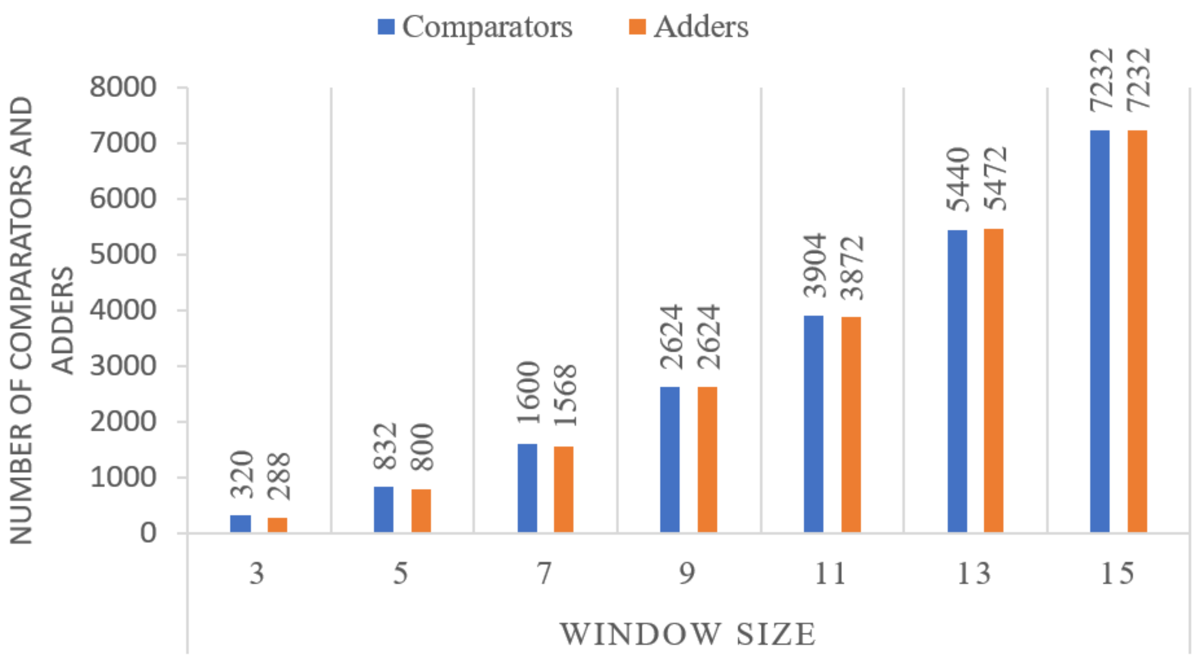


Figure 4.7 Number of adders and comparators as function of W_s

From Fig. 4.7 it is observed that the ratio of the number of adders or comparators required for window size W_s to that of W_{s-2} goes on decreasing as the value of W_s increases. This is due to the fact that the size of comparator module for each block is given by ($16W_s^2$), which increases rapidly with W_s. However, neither the size of the histogram calculation module nor that of the AC module increases as rapidly as that of the comparator module.

B) Area utilization in terms of the number of registers

The proposed architecture uses clocked comparators and clocked adders for its implementation. As a result, each of the comparators or adders is implemented along-with a register. The pipelined buffers shown in Fig. 4.2 consists purely of registers for buffering the input data (*N*) and *MH* value. Fig. 4.8 shows the number of registers used by the HMH and HML blocks, and that of the pipelined buffers as well as their sum as a function of the window size. It is observed from Fig. 4.7 and Fig. 4.8 that the number of registers required by the proposed architecture is larger than the total number of adders and comparators required, and this is in view of the additional registers used by the pipelined buffers.

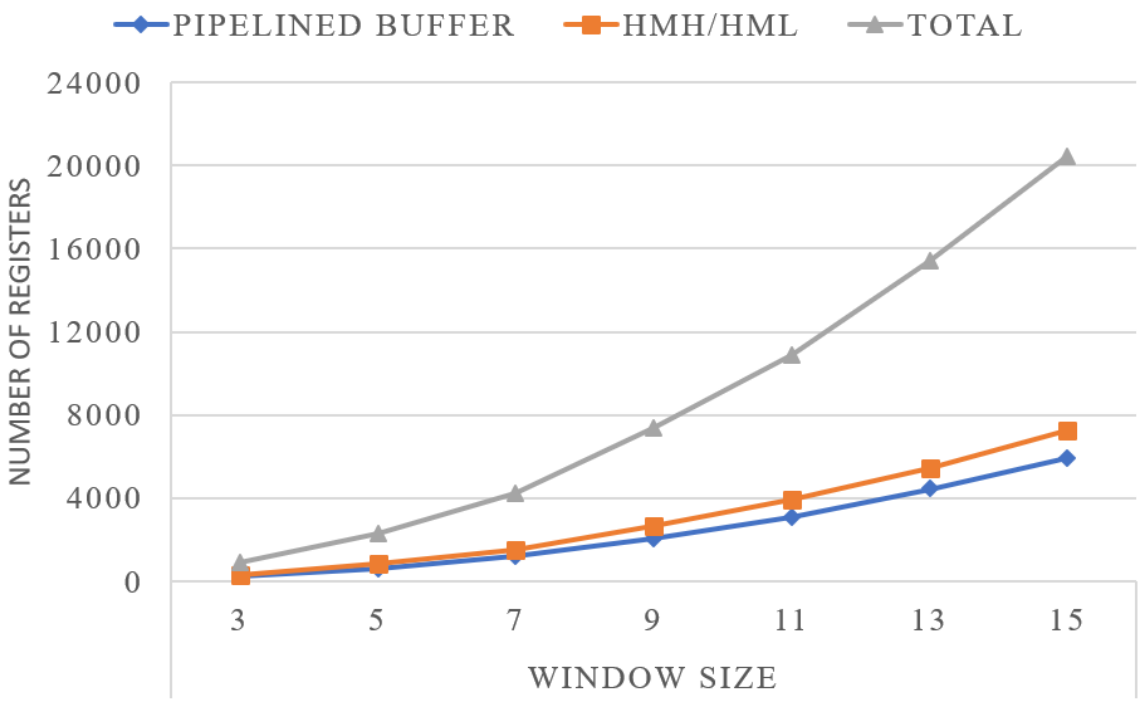


Figure 4.8 Number of registers as a function of W_s

C) Pipeline latency

The proposed architecture is designed in such a way that between a pair of cascaded adders and/or comparators there is a register. This approach is adopted in order to minimize the combinational delay and achieve a high throughput. As a result, a large number of registers are cascaded in the pipeline. The pipeline latency L of the proposed architecture is given by

$$L = 2(\lceil log_2 N \rceil + 18) \quad (4.1)$$

where N is the number of elements in the input array, the integer 18 refers to the combined delay of the comparator and AC modules, which is constant irrespective of the value of N. The factor 2 in (4.1) is due to the fact that the HMH and HML blocks are cascaded and have the same clock latencies. Fig. 4.9 shows the latency of the proposed design for different window sizes.

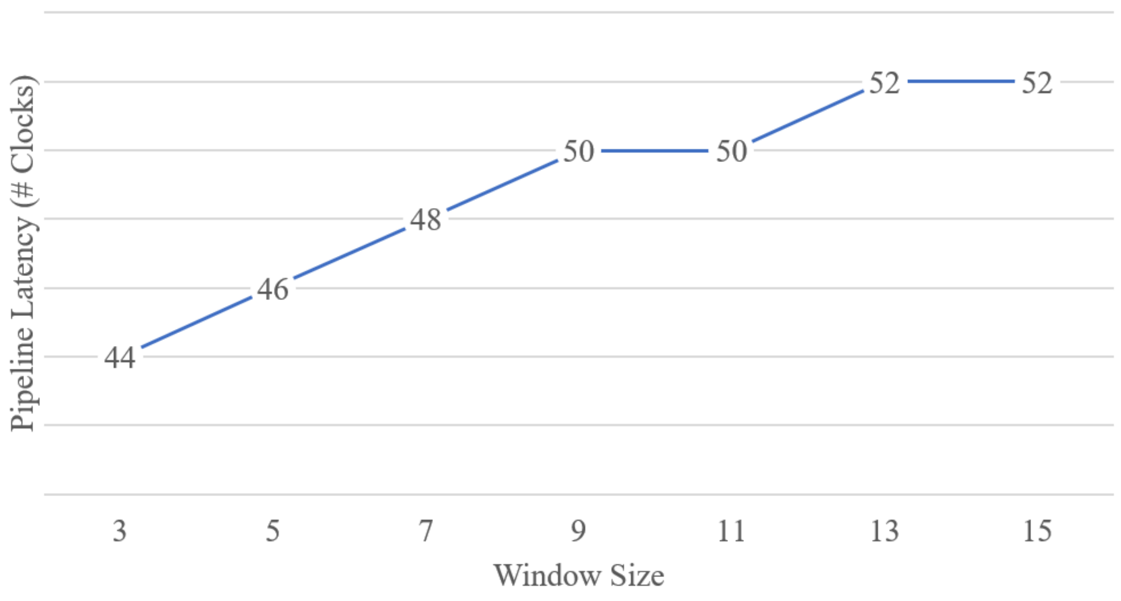


Figure 4.9 Pipeline latency as a function of W_s

It is clear from (4.1) that the latency is dependent mainly on the factor $\lceil \log_2 N \rceil$ and increases only when this factor increases. Although the latency of the proposed pipelined design is very high, the pipeline ensures that it operates continuously to provide a high throughput. The number of windows to be processed for the UHD video is massive ($> 49 \times 10^7$ for 4K and $> 199 \times 10^7$ for 8K at 60 fps); hence, the pipeline latency L, as given by (4.1) is negligible. For the sorting-based architecture, implemented with pipelining as in [34], the latency is given by W_s^2. Hence, for the largest window size considered in this paper ($W_s = 15$), the latency of a sorting-based median filter is 225 clock cycles, significantly larger than 52, which is the latency of the proposed architecture.

4.4 FPGA Implementation of Proposed Hardware Architecture

The implementation of the sorting-based median filter in [34] as a contextual image processing operation is the only text that targets real-time UHD video filtering, although only for 4K videos. Hence, to provide a comparative analysis, we map the proposed hardware architecture to the same device, namely, Zynq-7 UltraScale+ MPSoC. An MPSoC such as Zynq-7 UltraScale+ takes advantage of the software programmability by integrating ARM-based processors and the advantage of hardware reconfiguration using a FPGA fabric, both on the same chip. However, it is a costly solution for many applications. Hence, we evaluate the performance of the proposed median filter hardware for cheaper solutions, by implementing single and multiple cores of the proposed HHMF hardware architecture on *Pure* (Non-MPSoC) FPGA fabric such as Artix-7 FPGA.

4.4.1. Implementation on Zynq-7 UltraScale+ MPSoC

The proposed architecture is designed with RTL code written in VHDL on the Xilinx Vivado design suite platform. The code is mapped to Xilinx Zynq-7 UltraScale+ MPSoC for $W_s = 3$. Constraints are specified in the timing constraint editor to find the maximum delay (logic delay + net delay) of the design. Post implementation area utilization in terms of LUT, FF and BRAM are reported by the implementation tool along-with the maximum delay and power on the Zynq-7 UltraScale+ MPSoC device. Operating frequency of the design is calculated using the reported maximum delay.

Fig. 4.10 shows the device view of the design implemented on Zynq-7 UltraScale+ MPSoC along with a magnified view of the underlying logic elements used to implement the design. The shaded region in the top right corner of the device view is the proposed median filter implemented with FPGA slices. It is seen from this figure that the area utilization is very small, which leaves enough room for implementing the interfacing peripherals.

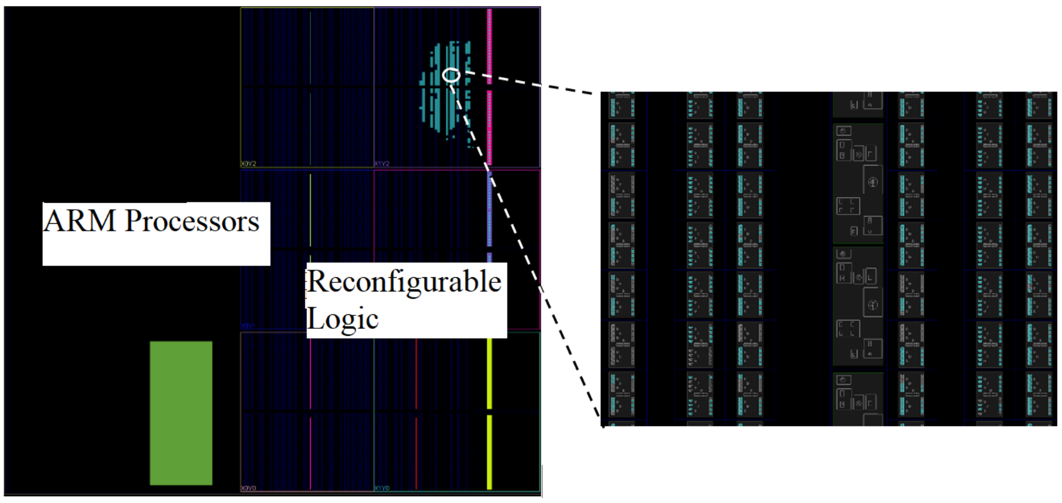
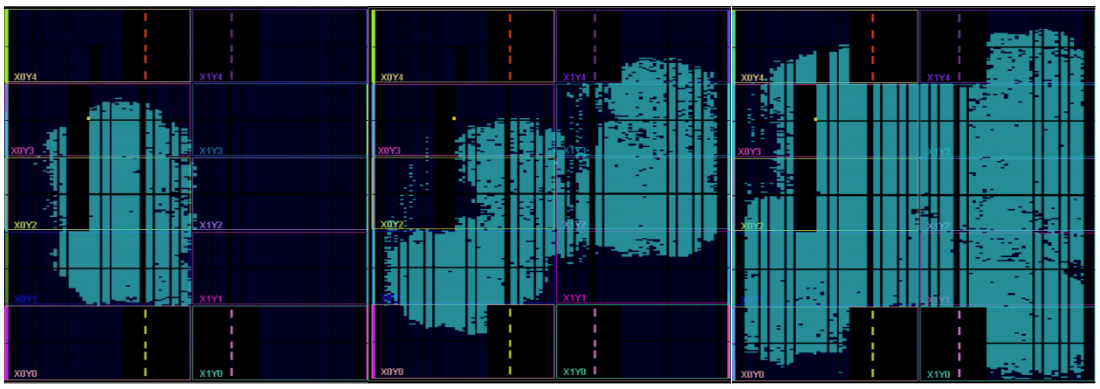


Figure 4.10 Device view of the implementation on Zynq-7 UltraScale+ MPSoC.

4.4.2 Implementation on Artix-7 FPGA

The RTL code for the proposed median filter is modular, since the parameters such as the window size, the number of the stages in the adder unit (see Fig. 4.5) of the histogram calculation module and the number of cores are variables. The code is mapped to Artix-7 FPGA for window sizes ranging from W_s = 3 to W_s = 15 and for 1, 2 and 4 cores of the proposed hardware architecture. Fig. 4.11 shows the device view of Artix-7 for 1, 2 and 4 cores of the proposed hardware architecture for W_s = 15, from which the scalability and maximum device utilization of the proposed FPGA implementation is observed. The area utilized by the proposed architecture when implemented on FPGA is shown by the shaded region in Fig. 4.11. It is seen from this figure that the maximum device utilization is in the case of the 4-core architecture, and there is still enough room for additional logic required by the interfacing hardware. In this case, the maximum utilization is 51% of the slice-LUTs and 36% of the slice-registers of the total present in the Artix-7, as reported in the post implementation device utilization report.



(a) (b) (c)

Figure 4.11 Device utilization of (a) 1 core (b) 2 core (c) 4 core architecture of the proposed HHMF hardware for $W_s = 15$ on Artix-7 FPGA.

4.5 Hardware Implementation Results

This section presents the FPGA implementation results of the proposed HHMF hardware architecture and analyses the area utilization, maximum operating frequency, power and throughput. Using the maximum operating frequency f_{max} (MHz) of the proposed hardware architecture, we calculate the throughput T in terms of the number of UHD video frames per second using (2.5). The frame size $F_s = 3840 \times 2160$ for 4K UHD format and v is 1, 2 or 4 which is the number of cores. The frame size $F_s = 7680 \times 4320$ for 8K UHD format. We calculate the throughput for different window sizes and compare it to that of HHMF when implemented on GPU.

4.5.1 Results of the implementation on Zynq-7 UltraScale+ MPSoC

The proposed HHMF hardware architecture can operate at a maximum frequency of 625 MHz, which satisfies the pixel clock requirement of a UHD video system when mapped to Zynq-7 UltraScale+ MPSoC. Table 4.1 shows the results of a single core implementation of the proposed HHMF hardware architecture in terms of FPGA area (LUT, FF, BRAM), frequency and power for $W_s = 3$, which processes a single input **p**ixel **p**er **c**lock (ppc) cycle. The throughput is calculated using (2.5). The number of input pixels required in a single clock cycle is also presented in this table. For comparative analysis, the corresponding results of CIPMF with 2 and 4 ppc for the same window size are also included in this table.

Table 4.1. Results of Implementation on Zynq-7 UltraScale+ MPSoC

Filter Hardware	**No. of Input Pixels**	**Area**			**Frequency (MHz)**	**Throughput (fps)**	**Power (W)**
		LUT	**FF**	**BRAM**			
Proposed -1ppc	9	1534	1538	0	625	75	0.384
CIPMF - 2ppc	12	1446	1264	3	300	72	0.641
CIPMF - 4ppc	18	2295	1605	3	150	72	0.535

The proposed median filter hardware provides a throughput higher than that of either implementations of CIPMF. The CIPMF (4ppc) implementation requires approximately 50% more LUTs and consumes about 40% more power compared to the proposed implementation and operates at 150 MHz Although the CIPMF (2ppc) implementation utilizes a slightly lower number of LUTs and FFs, its power consumption is twice that of the proposed implementation at the operating frequency of 300MHz, which is less than one-half of the proposed one. The number of input pixels required by the proposed hardware implementation in a single clock cycle is 9 in contrast to that of CIPMF, which requires 12 and 18 pixels to be read in a single clock cycle for 2 and 4 ppc format, respectively. Reading a higher number of pixels in a clock cycle requires more interfacing hardware, which increases the hardware complexity of the overall design.

4.5.2 Results of the implementation on Artix-7 FPGA

Fig. 4.12 shows the resource utilization in terms of the FPGA slice-LUTs and slice-registers of the proposed HHMF hardware architecture with different number of cores and window sizes when implemented on Artix-7 FPGA.

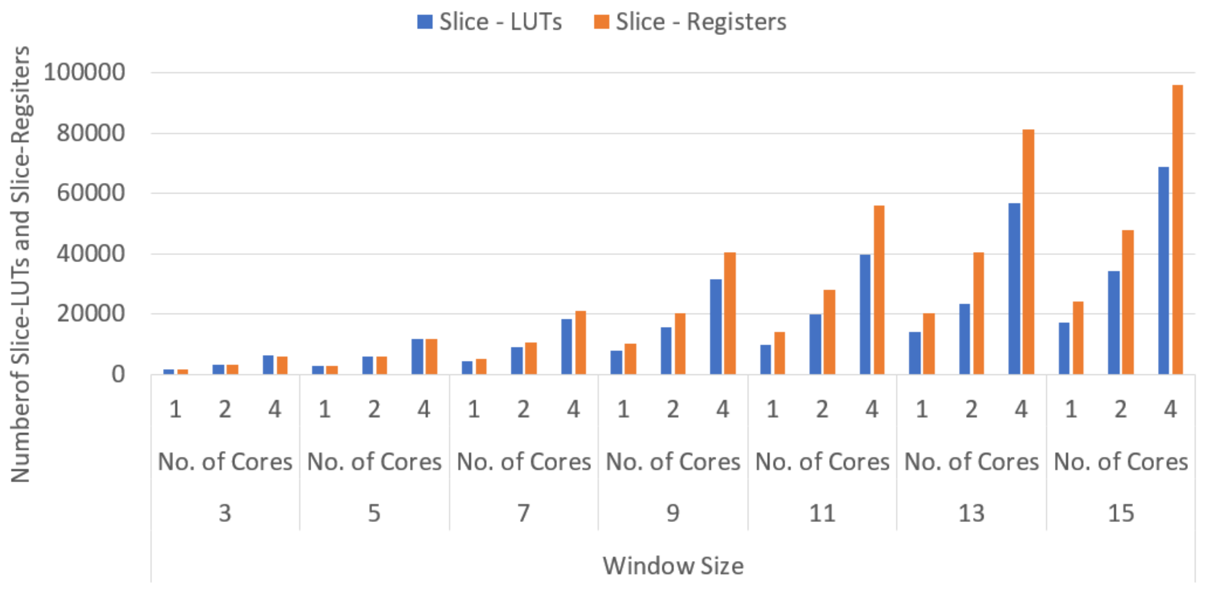


Figure 4.12 Resource utilization of the proposed HHMF hardware on Artix-7 FPGA

The Artix-7 FPGA implements the logic using 6-input LUTs. For lower window sizes many LUTs are under-utilized; however, for higher window sizes more logic fits into a single LUT resulting in its full utilization. The number of LUTs and registers are almost equal for lower window sizes; however, for higher window sizes the number of registers needed is significantly more than the number of LUTs. This is due to the increase in the number of registers utilized by the pipelined buffer, the size of which increases as the square of the input window size.

The HHMF algorithm, originally designed for GPU, provides a very high performance in terms of the throughput on NVIDIA GPUs such as Tesla and SoC parker as presented in [33], but such implementations are costly as noted earlier. We wish to investigate as to whether the throughput offered by these GPUs can be achieved by the proposed implementation on FPGA. Hence, we calculate the throughput of our implementation using (2.5) and compare it with the throughput reported by [33] for NVIDIA Tesla and SoC Parker GPUs. Fig. 4.13 shows the throughput values obtained by the FPGA implementation of the proposed HHMF hardware architecture. Performance reported by [33] is also presented in the same graph for comparison.

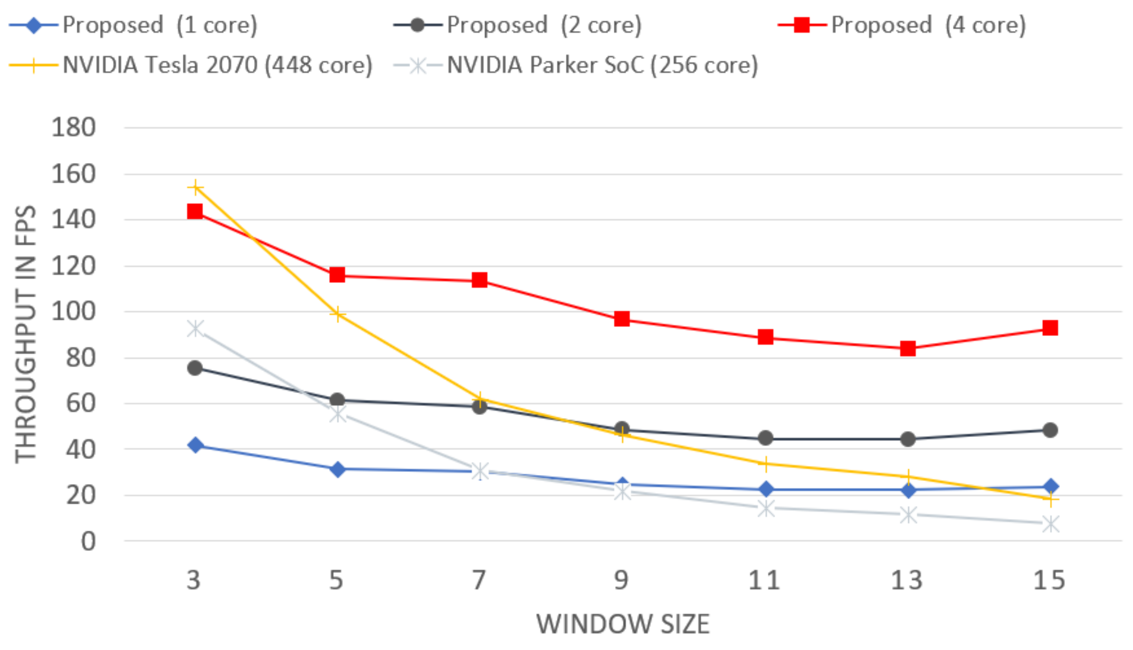


Figure 4.13 Throughput as a function of the window size

It is seen from Fig. 4.13 that the proposed 4 core architecture is the only one which can satisfy the real-time requirements of a 4K UHD video 60 fps for all window sizes in the range considered. The performance obtained by the proposed 4-core HHMF hardware is superior to that of NVIDIA GPUs, except in the case of NVIDIA Tesla GPU with W_s = 3, wherein the throughput of the proposed one is slightly lower. In general, the throughput decreases with increasing window size, irrespective of the implementation. However, the throughput of the proposed HHMF hardware is not adversely affected by the increasing window size. In particular it is observed that for large window sizes, the throughput of the proposed 4 core implementation is much higher than that offered by the GPUs. The throughput of the proposed deign when implemented on FPGA is based on the post-implementation delay reported by the tool. The value of the delay is large for lower window sizes, since the net/routing delays are larger compared to the logic delays. However, as the window size increases, there is no significant increase in the net/routing delay, but the logic delay goes on increasing due to the cascaded LUT configuration. For 15 × 15 window size, the throughput of the proposed HHMF hardware is slightly higher than that for 13 × 13 window size. This is due to the fact that these filter implementations have the same latencies and placement of the logic on FPGA results in lower net delays for $W_s = 15$ compared to that for $W_s = 13$.

Although the number of cores in a processing system is one of the factors which decides the throughput, a greater number of cores adds more complexity to the design in terms of the overall architecture. In terms of the number of cores, the implementation of the proposed median filter architecture with 4 cores can provide a throughput higher than that of the NVIDIA TESLA 2070 GPU which has 448 cores. The proposed median filter when implemented with 2 cores provides a throughput higher than that provided by the 256 core NVIDIA Parker SoC.

The power requirements of the proposed implementations are estimated by the Xilinx Xpower analyzer tool with vector-less analysis of the implemented netlist. In order to get a better estimate of the power, we specify the operating frequency of the respective implementations during the power analysis [51]. Fig. 14 shows the total on-chip power reported by the Xpower analyzer tool for various implementations of the proposed HHMF hardware architecture with different number of cores and window sizes. It is seen from this figure that the power increases with increasing window size or the number of cores. It is to be noted that although the power estimated by the Xpower analyzer may be slightly higher than the actual power utilized by the design [51], an FPGA based implementation is more energy-efficient with one order of magnitude lower energy than the same implementation on a GPU, for the same image processing task [63], [64].

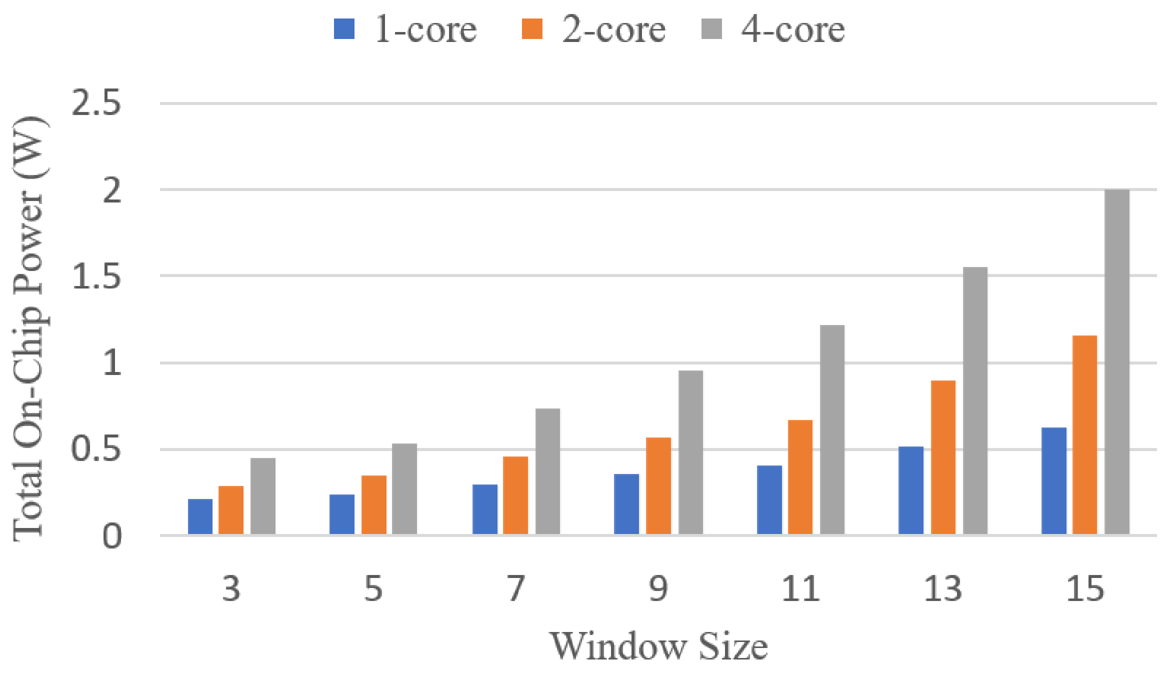


Figure 14.4 Total on-chip power of the proposed HHMF hardware on Artix – 7 FPGA.

The implementation of the proposed HHMF hardware on Artix – 7 FPGA can satisfy the real-time requirements of 8K UHD video at frame rates of 30 and 60 fps up to a window size of 9. For window sizes greater than 9, the slices required for the implementation exceed that available in the FPGA used for implementation. Fig. 4.15 shows the number of cores of the proposed HHMF hardware required for real-time filtering of 8K UHD videos at the above-mentioned frame rates as a function of W_s ($\leq$ 9). As observed from this figure, 8-core implementation of the proposed HHMF hardware can satisfy the real-time requirements of 8K UHD video at frame rate of 30 fps whereas a 12-core implementation can satisfy the real-time requirements of the same video format at frame rate of 60 fps.

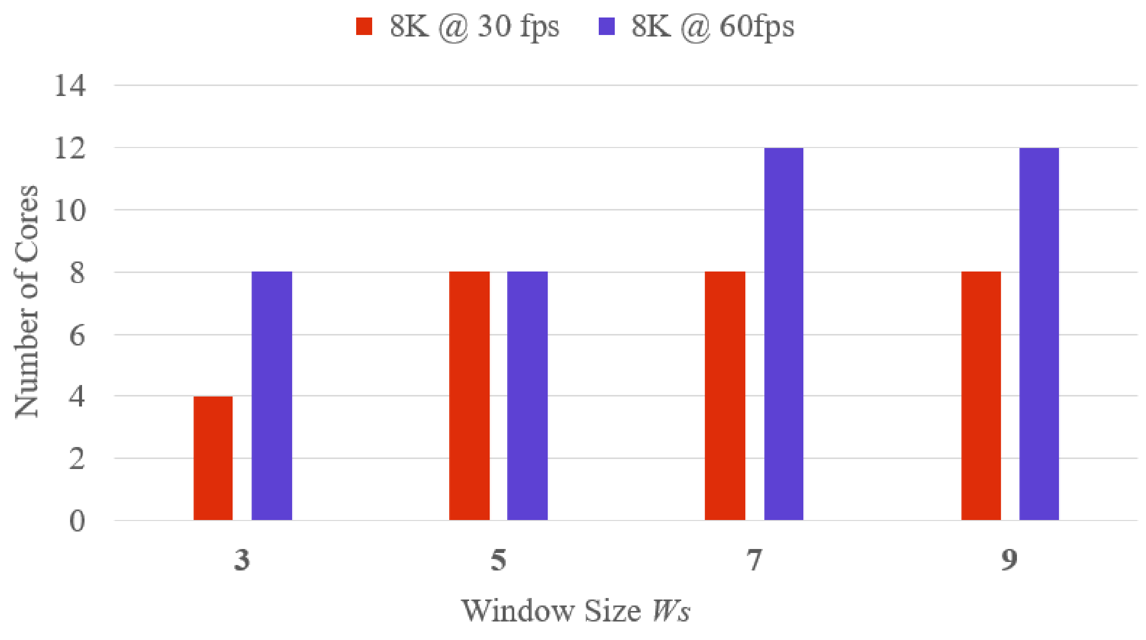


Figure 14.5 Number of cores required for filtering 8K UHD videos as a function of $Ws \leq 9$.

4.6 Summary

We proposed and implemented an efficient and fully-synchronous pipelined architecture for hierarchical histogram-based median filter, which is known to provide a high throughput when implemented on GPUs. The proposed architecture for the HHMF is designed such that it provides high operating frequencies and is scalable. We implemented the proposed median filter hardware on Artix-7 FPGA for window sizes ranging from 3 to 15 and different number of cores to evaluate the performance in terms of area and the throughput. Comparative analysis of the proposed implementations is carried out with the state-of-the-art median filter implementations to show the superiority of the proposed FPGA implementation in terms of the throughput. The 4-core implementation of the proposed HHMF hardware can satisfy the real-time requirements of 4K UHD videos at the frame rate of 60 fps for $W_s \leq 15$. The proposed filter hardware can also satisfy the real-time requirements of 8K UHD videos at the frame rates of 30 and 60 fps, when implemented with 8 and 12 cores, respectively for $W_s \leq 9$.

Chapter 5

Conclusion

5.1 Concluding Remarks

Median filters are known to provide high quality filtered signals when used for processing noisy signals. Due to the effectiveness of median filters, they are widely used for the removal of impulse noise and are extensively employed in applications involving speech, signal and image processing. As a result, several algorithms and architectures have been proposed and implemented for providing high quality filtered images and enhanced hardware performance. High-speed architectures and implementations are needed in order to satisfy the real-time throughput requirements of video formats such as full high definition (FHD) videos. Although many of the existing hardware-based implementations of median filters can satisfy the real-time throughput requirements of FHD videos, they are inefficient with respect area, when implemented on hardware. Moreover, the frame resolutions have now increased from FHD to ultra high definition (UHD), providing higher quality images and videos. However, the real-time requirements of UHD videos cannot be satisfied by the existing hardware architectures for FHD video filtering. As a result, efficient hardware-based architectures and implementations that can process in real-time, massive data contained in UHD video format are required.

In this thesis, we have presented one such architecture and its implementation for each of the video format considered. The proposed architectures have been implemented on FPGAs, which are a key device in the current digital consumer electronics. The architectures presented in this thesis make use of the concept of histogram-based median calculation instead of the sorting-based median calculation, which is used in most of the existing architectures. An architecture that uses sorting-based median calculation consists of cascaded stages of compare-and-swap (CS) units. This type of architecture is inefficient with respect to the area and operating frequency, since the number of CS units as well as the number of cascaded stages increases with increasing window size. However, in an architecture that uses histogram-based median calculation, the number of stages remains constant irrespective of the window size. Although the size of components used in the various stages of a histogram-based architecture increases with increasing window size, its overall area can be reduced by using the concept of bit-plane-slicing.

In order to satisfy the real-time throughput requirements of FHD videos, in the first part of the thesis, we have proposed a hybrid architecture for median filtering (HAMF) that combines histogram-based median calculation with the concept of bit-plane-slicing and adaptive switching median filter. The proposed architecture has been implemented on three different FPGAs, namely, Virtex-II, Virtex-6 and Zynq-7. These implementations have been optimized in terms of the area and can operate at a high frequency. The implementation of the proposed HAMF is suitable for low cost applications, where a very high throughput is required and a slight degradation in the quality of output image is acceptable. We have analyzed the effect of the number of bit-planes on the quality of the filtered image and on the hardware performance by implementing the proposed HAMF with three different values of number of the most significant bits used in median calculation (B). The analysis has shown that the quality of the images filtered by the algorithmic level implementation of the proposed HAMF decreases with decreasing values of B; however, the hardware implementation with a lower value of B is superior to that of an implementation with a higher value of B, in terms of area and operating frequency. Hence, a trade-off could be achieved between the image quality and hardware performance by an appropriate choice of the value of B. Although a subjective examination of the images filtered using a lower value of B such as 2 and 3, shows very little evidence of the input noise, the corresponding PSNR and SSIM values are significantly lower than that of the images filtered using ASMF. As a result, the implementation of the proposed HAMF with $B = 4$ is the optimum one. Analysis of the proposed HAMF implemented with $B = 4$ and different pipeline latencies has shown that there could be a trade-off between the area and the operating frequency. Hence, a suitable implementation may be selected depending on the parameter (area or frequency) to be optimized. Although the implementations of the proposed HAMF with different pipeline latencies can satisfy the real-time throughput requirements of FHD videos, they are still very slow for real-time filtering of UHD videos.

In order to fulfil the needs of real-time filtering of UHD videos and to overcome the disadvantages of HAMF, such as degradation in the filtered image quality and fixed window size, we have proposed an efficient architecture of the hierarchical histogram-based median filter (HHMF), in the second part of the thesis. This filter splits the values of pixels into two parts, namely, upper and lower nibbles and then calculates the median in two different stages. Due to this feature of HHMF, we could design an efficient hardware architecture, which was not possible to be accomplished for a histogram-based median filter. We have implemented the proposed architecture of HHMF on Xilinx Artix – 7 FPGA, for window sizes ranging from 3 to 15. Although the implementation of the proposed HHMF architecture with higher window sizes occupies more area when implemented on an FPGA, its operating frequency is very high, and the architecture is scalable. The proposed HHMF architecture has been implemented with full synchronous pipelining and its pipeline latency for higher window sizes, such as 9 to 15, is very low compared to the corresponding latencies of the sorting-based median filter architecture. A single core implementation of the proposed HHMF architecture can satisfy the real-time throughput requirements of FHD videos at the rate of 60 fps for window sizes ranging from 3 to 15. Implementation of the proposed HHMF architecture with 4 cores provides a high throughput which can satisfy the real-time requirements of 4K UHD videos at the rate of 60 fps for all window sizes in the range considered. Implementation of the proposed hardware architecture satisfies the real-time requirements of 8K UHD videos, up to a window size of 9 and frame rates of 30 and 60 fps, when implemented with 8 and 12 cores, respectively. The proposed HHMF implementation is more economical compared to that of the NVIDIA Tesla and Parker SoC GPUs and provides a throughput higher than that of the latter two, when implemented with 2 and 4 cores, respectively.

5.2 Scope for Future Work

One of the basic methods that has been employed in the proposed HAMF is bit-place-slicing, which results in the calculation of an approximate median. The hardware performance of HAMF is improved by using a limited number of the most significant bits for calculation of the approximate median. This method can be applied to the existing hardware-based median filters and the performance of the resulting hardware can be studied with respect to the quality of the filtered images and hardware parameters.

As mentioned in the thesis, the proposed hardware architecture of HHMF occupies more area for implementations with window sizes ranging from 9 to 15. Therefore, it would be of interest to study as to whether it is possible to optimize the architecture with respect to the area, for window sizes larger than 9.

Although the architectures presented in this thesis are implemented on FPGAs, they can be implemented in ASICs. Using VLSI CAD tools, it is possible to map the synthesized design of the proposed architectures to the *physical design* followed by ASIC implementations. The hardware performance of these implementations can be studied and compared with that of the FPGA implementations.

References

[1] M. Orlandić and K. Svarstad, “A low-complexity MPEG-2 to H.264/AVC wave front intra-frame transcoder architecture,” *Journal of Real-Time Image Proc.*, vol. 16, issue: 4, pp. 1007–1023, August 2019.

[2] H. Ge and J. Sha, “FPGA-based low-complexity high-throughput real-time hardware accelerator for robust watermarking,” *Journal of Real-Time Image Proc.*, vol. 16, issue: 4, pp. 813–820, August 2019.

[3] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Englewood Cliffs, NJ, USA: Prentice-Hall, 2008, pp. 155–157.

[4] M. Alenrex and R. Chatterjee, “Impulsive noise in images: A brief review,” *Computer Vision Graphics and Image Processing*, vol. 4, issue: 10, pp. 6-15, March 2018.

[5] Fabijanska and D. Sankowski, “Noise adaptive switching median-based filter for impulse noise removal from extremely corrupted images,” *IET Image Processing*, vol. 5, issue: 5, pp. 472 – 480, August 2011.

[6] S. Herzog, “Efficient DSP implementation of median filtering for real-time audio noise reduction,” presented at the *16th Int. Conference on Digital Audio Effects (DAFx-13)*, Maynooth, Ireland, September 2–5, 2013. Available online: http://dafx13.nuim.ie/proceedings.html.

[7] J. Suomela, “Median filtering is equivalent to sorting,” Available online: https://arxiv.org/abs/1406.1717.

[8] K. E. Batcher, “Sorting networks and their applications,” in *Proc. Spring Joint computer conference*, NY, USA, April 30-May 2, 1968, pp. 307–314.

[9] Sanny and V. K. Prasanna, “Energy-efficient median filter on FPGA,” in *Proc. Int. Conf. Reconfigurable Computing. FPGAs (ReConFig)*, Cancun, Mexico, Dec. 2013, pp. 1–8.

[10] Z. Vasicek and L. Sekanina, “Novel hardware implementation of adaptive median filters,” in *Proc. 11th IEEE Workshop on Design and Diagnostics of Electronics Circuits and Systems*, Bratislava, Slovakia, Apr. 2008, pp. 1 - 6.

[11] W. Chen, M. Beister, Y. Kyriakou and M. Kachelrieb, “High performance median filtering using commodity graphics hardware,” in *Proc. IEEE Nuclear Science Symposium Conference Record*, Orlando, FL, USA, January 2010, pp. 4142 – 4147.

[12] P. S. Battiato, “High performance median filtering algorithm based on NVIDIA GPU computing,” in *Proc. International Symposium for Young Scientists in Technology, Engineering and Mathematics*, Catania, Italy, September 2015, pp. 1-10.

[13] L. Hayat, M. Fleury and A. F. Clark, “Two-dimensional median filter algorithm for parallel reconfigurable computers,” *IEE Proc.-Vis. Image Signal Processing*, vol. 142, issue: 6, pp. 345 – 350, December 1995.

[14] R. M. Sanchez and P. A. Rodriguez, “Bi-dimensional median filter for parallel computing architectures,” in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing,* Kyoto, Japan, March 2012, pp. 1549 – 1552.

[15] R. M. Sanchez and P. A. Rodriguez, “Highly parallelable bi-dimensional median filter for modern parallel programming models,” *Journal of Signal Processing Systems archive*, vol. 71, issue: 3, pp. 221-235, June 2013.

[16] R. D. Chen, P. Y. Chen and C. H. Yeh, “Design of an area-efficient one-dimensional median filter,” *IEEE Transactions on Circuits and Systems—II: Express Briefs*, vol. 60, issue: 10, pp. 662 – 666, October 2013.

[17] R. D. Chen, P. Y. Chen and C. H. Yeh, “A low-power architecture for the design of a one-dimensional median filter,” *IEEE Transactions on Circuits and Systems II: Express Briefs,* vol. 62, issue: 3, pp. 266 - 270, March 2015.

[18] S. H. Lin, P. Y. Chen and C. K. Hsu, "Modular Design of High-Efficiency Hardware Median Filter Architecture," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, issue: 6, pp. 1929 - 1940, June 2018.

[19] L. Sekanina, Z. Vasicek and V. Mrazek, "Approximate circuits in low-power image and video processing: The approximate median filter," *Radio Engineering*, vol. 26, issue: 3, pp. 623 - 632, September 2017.

[20] S. Mittal, "A survey of techniques for approximate computing," *Journal ACM Computing Surveys (CSUR),* vol. 48, no. 4, pp. 62:1– 62:33, March 2016.

[21] E. Kalali and L. Hamzaoglu, "A low energy 2D adaptive median filter hardware," in *Proc. Design, Automation & Test in Europe Conference & Exhibition*, Grenoble, France, March 2015, pp. 725–729.

[22] E. Kalali and L. Hamzaoglu, "Low complexity 2D adaptive image processing algorithm and its hardware implementation," *IEEE Transactions on Consumer Electronics*, vol. 63, issue: 3, pp. 277 - 284, August 2017.

[23] V. Kumar, A. Asati and A. Gupta, "Low-latency median filter core for hardware implementation of 5 × 5 median filtering," *IET Image Processing*, vol. 11, issue: 10, pp. 927–934, Oct. 2017.

[24] P. T. Jelodari, M. P. Kordasiabi, S. Sheikhaei and B. Forouzandeh, "FPGA implementation of an adaptive window size image impulse noise suppression system," *Journal of Real-Time Image Proc*, vol. 10, pp. 1 – 12, July 2017.

[25] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware Interface*. Boston, MA, USA: Morgan Kaufmann, 2014, pp. 277- 290.

[26] T. S. Huang, G. Y. Yang and G. Y. Tang, "A fast two-dimensional median filtering algorithm," *IEEE Transaction on Acoustics, Speech and Signal Processing*, vol. ASSP-27. issue:1, pp. 13-18, February 1979.

[27] B. Weiss, "Fast median and bilateral filtering," *ACM Transactions on Graphics (TOG)* vol. 25, issue: 3, pp. 519-526, July 2006.

[28] S. Perreault and P. Hebert, "Median filtering in constant time," *IEEE Transactions on Image Processing*, vol. 16 , issue: 9, pp. 2389–2394, Sept. 2007.

[29] S. A. Fahmy, P.Y.K. Cheung and W. Luk, "Novel FPGA-based implementation of median and weighted median filters for image processing," in *Proc. International Conference on Field Programmable Logic and Applications*, Tampere, Finland, Aug. 2005, pp. 142–147.

[30] S. A. Fahmy, P. Y. K. Cheung and W. Luk, "High-throughput one dimensional median and weighted median filters on FPGA," *IET Computers & Digital Techniques*, vol. 3, issue: 4, pp. 384–394, July 2009.

[31] M. Malek and C. W. Sensen, "Instant feedback rapid prototyping for GPU-accelerated computation, manipulation, and visualization of multidimensional data," *International Journal of Biomed Imaging*, Article ID 2046269, June 2018.

[32] O. Green, "Efficient scalable median filtering using histogram-based operations," *IEEE Transactions on Image Processing*, vol. 27, issue: 5, pp. 2217–2228, May 2018.

[33] P. Szántó and B. Fehér, "Hierarchical histogram-based median filter for GPUs," *Acta Polytechnica Hungarica*, vol. 15, issue: 2, pp. 49–68, January 2018.

[34] M. Kowalczyk, D. Przewlocka and T. Kryjak, "Real-time implementation of contextual image processing operations for 4K video stream in Zynq UltraScale+ MPSoC," in *Proc. Conference on Design and Architectures for Signal and Image Processing*, Porto, Portugal, Oct. 2018, pp. 37–42.

[35] K. Benkrid and D. Crookes, "New bit-level algorithm for general purpose median filtering," *Journal of Electronic Imaging,* vol. 12, issue :2, pp. 263-269, April 2003.

[36] L. Tan and J. Jiang, *Digital Signal Processing: Fundamentals and Applications*. Burlington, MA, USA: Elsevier Academic Press, 2019, pp. 649-726.

[37] S. Khan and D. Lee, “An adaptive dynamically weighted median filter for impulse noise removal,” *EURASIP Journal on Advances in Signal Processing*, article no. 67, December 2017.

[38] R. Ha, P. Liu and K. Ji, “An improved adaptive median filter algorithm and its application,” in *Proc. Pan JS., Tsai PW., Huang HC. (eds) Advances in Intelligent Information Hiding and Multimedia Signal Processing. Smart Innovation, Systems and Technologies*, vol 64. Springer, Cham, pp. 179-186.

[39] T. Matsubara, V. G. Moshnyaga and K. Hashimoto., “A low-complexity and low power median filter design,” in *Proc. International Symposium on Intelligent Signal Processing and Communication Systems,* Chengdu, China, December 2010, pp. 1-4.

[40] Z. Zhang, D. Han, J. Dezert and Y. Yang, “A new adaptive switching median filter for impulse noise reduction with pre-detection based on evidential reasoning,” *Signal Processing*, vol. 147, pp. 173-189, Jun. 2018.

[41] M. M. Mano and M. D. Ciletti, *Digital Design with an Introduction to Verilog HDL*. Upper saddle river, NJ, USA: Pearson Education, 2013, pp. 113 - 116.

[42] D. Blaauw, K. Chopra, A. Srivastava and L. Scheffer, “Statistical timing analysis: From basic principles to state of the art,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 27, issue, pp. 589 – 607, April 2008.

[43] J. L. Hennessy and D. A. Patterson, *Computer Organization: A Quantitative Approach.* MA, USA: Morgan Kaufmann, 2012, pp. C-58 – C-63.

[44] N. P. Jouppi, “Superscalar vs. Super-pipelined Architecture,” *ACM Computer Architecture News*, vol. 16, issue: 3, pp. 71 – 80, June 1988.

[45] J. Subramaniam, R. J. Kannan and D. Ebenezer, "Parallel and pipelined 2-D median filter architecture," *IEEE Embedded Systems Letters*, vol. 10, issue: 3, pp. 69 – 72, September 2018.

[46] U. Sara, M. Akter and M. S. Uddin, "Image quality assessment through FSIM, SSIM, MSE and PSNR - A comparative study," J*ournal of Computer and Communications*, vol. 7, pp. 8-18, March 2019.

[47] R. Dosselmann and X. D. Yang, "A comprehensive assessment of the structural similarity index," *Signal, Image and Video Processing*, vol. 5, issue: 1, pp 81–91, March 2011.

[48] M. Beltrán, A. Guzmán, and F. Sevillano, "High level performance metrics for FPGA-based multiprocessor systems", *Performance Evaluation*, vol. 67, issue: 6, pp. 417-431, June 2010.

[49] C. S. Regazzoni and A. Teschioni, "A new approach to vector median filtering based on space filling curves," *IEEE Transactions on Image processing*, vol.6, pp. 1025 – 1037, July 1997.

[50] K. O. Boateng, B. W. Asubam and D. S. Laar, "Improving the effectiveness of the median filter," *International Journal of Electronics and Communication Engineering*, vol.5, pp. 85-97, Jan. 2012.

[51] "Power Methodology Guide," v14.5, Xilinx, USA, pp. 21 – 22, April 2013.

[52] "Vivado Design Suite user Guide: Power Analysis and Optimization," v 2019.1, Xilinx, USA, pp. 32 – 34, May 2019.

[53] D. Meidanis, K. Georgopoulos, I. Papaefstathiou, "FPGA power consumption measurements and estimations under different implementation parameters," in *Proc. Int. Conf. Field-Program. Technol. (FPT)*, New Delhi, India, Dec. 2011, pp. 1-6.

[54] U. Farooq, Z. Marrackchi and H. Mehrez, *Tree-based Heterogeneous FPGA Architectures Application Specific Exploration and Optimization.* NY, USA: Springer-Verlag, 2012, pp. 40 - 43.

[55] N. H. E. Weste and D. M. Harris, *CMOS VLSI Design: A Circuit and System Prospective.* Boston, MA, USA: Addison-Wesley, 2011, pp. 55.

[56] A. P. Chandrakasan and R. W. Brodersen, "Minimizing power consumption in digital CMOS circuits," in *Proc. IEEE*, vol. 83, Apr. 1995, pp. 498–523.

[57] A. Goel, M. O. Ahmad and M.N.S. Swamy, "Design of a 2D median filter with a high throughput FPGA implementation," in *Proc. 62nd IEEE Midwest Symposium on Circuits and Systems*, Dallas, TX, USA, Aug. 2019, pp. 1073 - 1076.

[58] Image Database: http://www.eecs.qmul.ac.uk/~phao/IP/Images/

[59] Video Database: https://media.xiph.org/video/derf/

[60] A. Goel, M. O. Ahmad and M.N.S. Swamy, "An Efficient Real-Time FPGA Implementation of the Hierarchical Histogram-Based Median Filter for UHD Videos," submitted to *IEEE Transactions on Circuits and Systems I: Regular Papers*.

[61] "GPU vs FPGA performance comparison," Whitepaper, Berten Digital Signal Processing, Cantabria, Spain, May 2016.

[62] Q. F. Stout, "Supporting divide-and-conquer algorithms for image processing," *Journal of Parallel and Distributed Computing,* vol. 4, issue: 1, pp. 95–115, February 1987.

[63] X. Lu, L. Song, S. Shen, K. He, S. Yu and N. Ling, "Parallel Hough transform-based straight line detection and its FPGA implementation in embedded vision," *Sensors*, vol. 13, pp. 9223-9247, July 2013.

[64] J. Fowers, G. Brown, P. Cooke and G. Stitt, "A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications," in *Proc. of the*

ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Monterey, CA, USA, February 2012, pp. 47–56.