High throughput VLSI architecture for gradient guided filter with approximated arithmetic operations

Wu, Lei

2018

Wu, L. (2018). High throughput VLSI architecture for gradient guided filter with approximated arithmetic operations. Doctoral thesis, Nanyang Technological University, Singapore.

<http://hdl.handle.net/10356/74100>

<https://doi.org/10.32657/10356/74100>

**High Throughput VLSI Architecture
for Gradient Guided Filter with Approximated
Arithmetic Operations**

Wu Lei

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Doctor of Philosophy

2018

PhD Thesis

version 1.0.0

2018

Nanyang Technological University

VIRTUS, IC Design Center of Excellence

School of Electrical and Electronics Engineering

Block S3.2, Level B2

50 Nanyang Avenue

639798 Singapore

Abstract

Guided image filtering has been applied widely for increasing demand of high performance filtering, especially for real-time image/video processing. Gradient guided filter improves the filtering quality, reducing the halo-artifacts problem due to its edge-aware characteristics. However, the gradient guided filter algorithm has high computation complexity and the computation involves global pixels, which hinder its VLSI implementation for real-time full-HD application. This work addresses these issues and a VLSI architecture is proposed for the gradient guided image filter. Several design techniques are developed and used in the design to achieve high computation speed and high throughput. A seamless dataflow is proposed for the complete system that consists of three main processing stages, specifically the preprocessing stage, the linear coefficient computation stage and the output stage.

The preprocessing stage applies a down-sampling technique with a large sampling rate to reduce computation cost in terms of circuit size, processing time and power consumption. The global parameter values are quickly derived with reasonable good global information maintained so that the quality of the filtering results are not sacrificed when these values are applied in the subsequent two stages.

The linear coefficient computation stage contains the most complex computations such as square root, division and exponential function. Down-sampling technique is applied with a sampling rate lower than in the preprocessing stage so as to balance the computation cost and filtering accuracy. In addition, the intensive arithmetic computation modules that dominate the critical path delay are redesigned by using adequate approximated operations. Specifically, novel non-iterative dividers are developed to replace original dividers for reducing delays in the critical paths. With the proposed non-iterative division,

the quotient of the division is modeled as a normalized curved surface. The curved surface is partitioned into small regions and approximated by smaller planes for efficient hardware implementation. Curve fitting method and mixed integer linear programming method are adopted and evaluated for local optimization of the approximation errors. In this way, the dividers are implemented with only simple arithmetic operations and a small look-up table. As a result, the operation is fast and the approximation errors are optimized while satisfying the accuracy requirement. The other intensive computation, the exponentiation function, is also simplified by piecewise linear approximation, and implemented with only shifters and adder trees. As such, the approximated computations are used to improve the computation performance and simplify the designs of the complex arithmetic modules.

The output stage employs parallel processing and operates at a frequency 16 times higher to restore the filtering results to its original full frame size. The linear coefficients obtained from the down-sampling stage are applied concurrently to all the 16 pixels in the window that the coefficients are computed for. As a result, not only the original image size is restored, the quality of the filtering results is maintained and high throughput is achieved at the same time.

Based on the STM 65nm CMOS technology, the implementation result shows the proposed VLSI architecture for the gradient guided image filter is able to support Full-HD image filtering at a throughput above 150 frames per second, achieving high throughput, small size and low power consumption comparing to the existing VLSI design for the original guided image filtering. This thesis describes the proposed VLSI architecture, the details of its design, the designs of the non-iterative dividers and the design of the piecewise linear approximated exponential function. The qualitative and quantitative analysis of the impact of the approximations on the filtering results and the performance comparisons of the proposed designs and the existing designs are included.

Acknowledgments

It would be a great opportunity to express my deepest gratitude to my supervisor, Prof Jong Ching chuen, for his meaningful help and insightful guidance during my PhD study in the last few years. From the very beginning, he encouraged me to ask a good question before answering one, bringing critical thinking into my academic life. I would also like to thank friends and other fellow students who share ideas with me, providing great atmosphere in both study and life.

Thanks to my parents and family, who support and encourage me as always in my life with their endless love.

Contents

Abstract	ii
Acknowledgments	iv
Contents	vii
List of Figures	x
List of Tables	xi
List of Abbreviations	xiii
1 Introduction	1
1.1 Background	1
1.2 Motivation	4
1.3 Research Objectives	6
1.4 Main Contributions	7
1.5 Organization of Thesis	9
2 Literature Review	12
2.1 Overview	12
2.2 Guided Image Filtering	13
2.2.1 Concepts of GIF	13
2.2.2 Developments of GIF	17
2.3 Hardware Implementation of Guided Image Filtering	24

2.4	Implementations for Arithmetic Divisions	27
2.5	Summary	31
3	Fast Gradient Guided Image Filter Design	32
3.1	Overview	32
3.2	Design Challenges	34
3.3	Dataflow Organization	37
3.3.1	Preprocessing Procedure	38
3.3.2	Linear Coefficients Procedure	39
3.3.3	Restoring Procedure	41
3.3.4	Sampling and Restoration	41
3.4	Proposed Hardware Architecture	42
3.4.1	Preprocessing Stage	43
3.4.2	Linear Coefficients Computing Stage	46
3.4.3	Parallel Restoring Stage	49
3.5	Performance Evaluations and Discussions	49
3.5.1	Filtering Analysis	49
3.5.2	Performance Evaluation on Hardware Design	52
3.6	Summary	54
4	Non-iterative Dividers Design	55
4.1	Overview	55
4.2	Modeling and Formulation for Division	58
4.3	Approximation Approaches	61
4.3.1	Curve Fitting Method	61
4.3.2	Mixed Integer Linear Programming	64
4.3.3	Stripe-based approximation	65

4.4	Hardware Architecture	68
4.4.1	Sign and Exponent Module with Overflow Flag	68
4.4.2	Mantissa Module	68
4.5	Performance Evaluation and Comparison	70
4.5.1	Performance Evaluation	70
4.5.2	Comparisons	73
4.6	Summary	74
5	Approximated Circuits and Trade-offs in Proposed Filter Design	76
5.1	Overview	76
5.2	Stripe-based Processing	78
5.3	Approximation of Exponentiation	82
5.4	Division Implementation in Proposed Filter Design	87
5.5	Performance Evaluation and Comparisons	89
5.6	Summary	94
6	Conclusions	95
6.1	Conclusions	95
6.2	Future work	101
	Publication	103

List of Figures

2.1	Concept illustration of the guided filtering process [1].	13
2.2	Comparison between bilateral filter (BIF) and guided image filter (GIF) on gradient reversal artifacts, simulated by using the algorithm provided by [1]. (a) BIF. (b) GIF. (c) BIF. (d) GIF.	15
2.3	Edge preserving smoothing using guided filtering with different parameter selections [1].	16
2.4	HDR compression comparisons for quantization artifacts [1]. (a) Input HDR image. (b) Compressed image with bilateral filter. (c) Compressed image with guided filter.	17
2.5	Illustration for halo-artifacts after smoothing with different filters [1]. (a) Input image. (b) Guided filter. (c) Bilateral filter.	18
2.6	Enhancement result of WGIF comparing to others [2]. (a) Input image with haze. (b) Dehazed result by GIF. (c) Dehazed result by WGIF.	19
2.7	Relationships among filtering windows in an image.	22
2.8	Intensity 1D analysis on filtering results from different filters, the input data comes from a middle row of the source image displayed [3].	23
2.9	A VLSI architecture for the original GIF [4].	24
2.10	Illustration of integrating process by scanning [5].	26
2.11	Logarithm and anti-logarithm approximation [6]. (a) Piecewise linear (PWL) approximation for logarithm. (b) Piecewise linear approximation for anti-logarithm.	29

2.12	Logarithmic arithmetic units. (a) Three stage logarithmic arithmetic unit architecture[7]. (b) Single stage division unit [8].	30
3.1	Proposed dataflow for the gradient guided filtering.	37
3.2	Hardware architecture of the proposed guided image filter.	44
3.3	Mechanism of box filter integrating.	46
3.4	Architecture of the box filters.	47
3.5	Hardware design for a_k and b_k kernel.	48
3.6	Comparison of the original guided image filter and proposed fast gradient guided filter in the application of detail enhancement simulation. (a) Input source image $I(p)$. (b) Detail enhanced result by original guided image filter. (c) Detail enhanced result by fast gradient guided filter.	50
3.7	The one-channel (green) intensity analysis on detail enhancement.	51
4.1	The adapted number representation.	58
4.2	The fraction division curved surfaces. (a) Before folding; (b) After folding.	60
4.3	(a) Normalized division curved surfaces evenly partitioned. (b) Percentage error distri- bution when approximated by planes using the curve fitting method.	62
4.4	The error distribution of approximation using the MILP approach.	66
4.5	Views from two axes on the error distribution of approximation using the MILP ap- proach. (a) Viewing from m_a axis. (b) Viewing from m_b axis.	67
4.6	The error distribution for the 32-partition approximation.	67
4.7	Proposed architecture for the non-iterative divider circuit.	69
4.8	The MAEP and ADP trade-offs for various partitions and coefficient wordlengths. . . .	73
5.1	Stripe-based processing.	79
5.2	Relation between computation cost and stripe width.	80
5.3	Simplified down-sampling illustration, with sampling ratio $s = 4$	81

5.4	Edge information conveyed by value γ_k from different implementations of the guided filter. (a) Guided image I . (b) γ_k from the original GDGIF design. (c) γ_k from the proposed FGGF design. (d) γ_k from the FGGF design with linear approximation of γ_k . (e) γ_k from the FGGF design with linear approximation γ_k and non-iterative dividers.	83
5.5	Linear approximation of the γ_k constraint.	84
5.6	Micro architecture of linear approximation for γ_k	86
5.7	The one-channel (green) intensity analysis on detail enhancement for proposed FGGF designs without and with approximations.	90
5.8	Detail enhancement for multiple image sources. (a) Guided source image. (b) - (d) Enhancement result by the original GIF, the proposed FGGF and the proposed FGGF with all approximation modules respectively.	91
5.9	Performance summary on frame rate, design area and energy cost per frame.	93

List of Tables

3.1	Comparisons with existing guided image filter design	52
4.1	MAEP and ADP trade-off of divider circuits with curve fitting method	71
4.2	Divider circuits performance evaluation	72
4.3	Performance comparisons with existing dividers	74
5.1	Hardware performance of the proposed FGGF designs with linear approximation module for γ_k kernel, in synthesis and layout.	87
5.2	Implementation results of the proposed FGGF designs with non-iterative divider. . . .	88
5.3	Hardware performance of the proposed FGGF designs with both approximation circuits.	92

List of Abbreviations

ABF	Adaptive Bilateral Filter
ADP	Area-Delay Product
AGIF	Adaptive Guided Image Filter
BF	Box Filter
BIF	Bilateral Image Filter
CMOS	Complementary Metal Oxide Semiconductor
CPD	Critical Path Delay
CT	Computerized Tomography
EOS	Extended Overlap Stripe
FGGF	Fast Gradient Guided Filter
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GDGIF	Gradient Domain Guided Image Filter
GIF	Guided Image Filter
GPU	Graphich Processing Unit
HDR	High Dynamic Range
IC	Integrated Circuit
LAU	Logarithmic Arithmetic Unit
LSB	Least Significant Bit

MSB	Most Significant Bit
MAC	multiplier-accumulator
MAEP	Maximum Absolute Error Percentage
SQRT	SQuare RooT
VLSI	Very Large Scale Integration
WLS	Weighted Least Squares

Chapter 1

Introduction

1.1 Background

In electrical engineering and computer science, image processing is a form of signal processing for which the input is an image, such as photographs or frames of video, and the output can be either an image or a set of characteristics related to the image [9]. Digital image processing is the use of computer algorithms to perform image processing on digital images. In the algorithms, an image is usually viewed as a 2D signal representation, upon which proper signal processing techniques are applied. The applications of image processing varies among human activities, including automatic visual inspection, remotely sensed scene interpretation, biomedical image analysis, content-based image retrieval, moving object tracking and so on [10].

Digital image processing originates in the newspaper industry from the time when pictures were transmitted via cable under the Atlantic ocean in the 1920s [11]. At that time the requirement is to improve visual image quality such as intensity level distribution. Ever since then, especially after digital computers are widely used, various image processing techniques have been ubiquitously used in different applications. Other than newspapers, space programs including the moon exploration and the medical imaging in diagnosis, such as computerized tomography (CT), have pushed the advancement of

digital image processing. Widely applied in different areas of specialization, digital image processing has made almost every progress dependent on the development of modern digital computers and supporting techniques, including data storage, display and transmission, for the fundamental fact that great computation power and data storage are demanded in these processes [11].

Common image processing algorithms include image acquisition, enhancement, restoration, compression, segmentation, recognition and many more. Each of these algorithms or steps corresponds to one or more components of an image processing system, including sensors, processors, displays, storage and transmission. Among them, the image enhancement is the process of image manipulation to have more suitable output image than the original one for a specific problem. Though all image enhancement processes generate new images based on original images, the techniques used are diversified, because they are application specific and the judge of processing quality is usually subjective.

Practically, a real application usually may not be able to process an acquired image directly. The image may be corrupted by random variations in intensity, variations in illumination, or poor contrast that must be dealt with in the early stages of vision processing [12]. Image filtering is intended to deal with such imperfect state of input images or frames of video as a preprocessing step to improve the image quality for later applications, so that the filtered image can be used [13]. The image or video quality are usually measured by experience of viewers by subjective methods, while there are some works done to develop objective measures as well [14], [15]. No matter what methods are employed for quality assessments, image filtering refers to the processing of an input image to produce either a better-looking output image by contrast, sharpness and/or signal-to-noise ratio enhancement, or to compute some low-level image features such as edges, corners, or spatial-gradient values that may be used in subsequent image analysis [13].

The image filtering process is often helpful for image enhancement as a preprocessing step. Image enhancement is the processing of images to help improve their usefulness [16]. The improvement can be used either for human viewers at perceptual aspects as in television, or to aid machine processing performance, as in object identification. Sometimes, the enhancement is conducted by modifying its

contrast or dynamic range. The other purpose of image enhancement is to reduce the degradation of an image. The image enhancement result can be also an intermediate step for later application such as edge detection and object detection [17], [18], [19].

Very Large Scale Integration (VLSI) systems contain millions of transistors and wire connections in a single chip that form a compact hardware system to perform a task of general or specific purposes [20]. The exponentially increasing density of integration pushed by the Integrated Circuit (IC) industry promises faster and more complex computations to be available for more and more advanced applications [21]. Application Specific Integrated Circuits (ASICs) are implemented for a specific problem when the problem is complex and software implementation is not able to meet real-time speed requirement. VLSI solution can be much more efficient than the conventional methods such as software algorithms executed in general purpose processors [22].

Many complex computations and operations are involved in the field of image processing, even the elementary operations of line identification [23] and texture perception [24] involve relatively sophisticated algorithms and mathematical functions. However, the speed of processing is critical in many problems, especially for real-time applications like video processing. The VLSI implementation of the processing algorithms for such specific applications is a feasible solution [25]. The demand of high throughput for real-time processing, and the complexity of accessing data and managing storage are the main challenges for designing such hardware systems [26].

Guided image filtering (GIF) is among the trending filtering techniques that have been applied in various fields of application, since its first introduction in [27] and subsequently more systematically analyzed and formulated in [1]. A guided image is given for such filtering process, used by a local linear model to compute the filtering output, unlike simple Linear Translation-Invariant (LTI) filters like Gaussian [28], [29] and Laplacian filters [30], [31]. More structural information can be provided by the additional guided image besides the input image alone. The guided image filtering aims at the applications where edge-preserving filtering is required, so that the more important structural information is preserved and at the same time detailed textures can be smoothed. The nature of simple local linear

model makes the algorithm for guided image filtering non-iterative, just like the bilateral filters [32], [33]. Moreover, the time complexity of the guided image filtering is $O(N)$, where N is the total number of pixels in a frame to be processed, and is independent of the kernel size and intensity range. The guided image filtering algorithm overcomes the gradient reversal artifacts that are often observed in the results obtained by bilateral filters.

1.2 Motivation

The guided image filtering technique has evolved considerably since its introduction for better quality at edge-preserving filtering. The original GIF [1] suffers from the problem of halo-artifacts, where the texture information is strong comparing to the structural information. Many research works have been reported to address this problem from different angles. Among them, the gradient domain guided image filtering (GDGIF) [3] is the latest technique that has two edge-aware parameters added for better recognizing the edge areas that are sharpened while other areas are to be smoothed. Experiments using the GDGIF demonstrate good filtering results that significantly reduces or avoids the halo-artifacts [1].

Guided image filtering has been widely applied in image processing. It has been adopted in many applications due to its good filtering quality, simple model and highly efficient and non-iterative computation [1]. The applications in the fields of computer graphics and computer vision include but are not limited to stereo matching, optical estimation, interactive image segmentation, saliency detection and illumination rendering. Many of these tasks, such as [4], [5], [34], [35], require real-time performance, especially in the field of video processing. Guided image filtering is usually a preprocessing step of these tasks, which involve intensive computation after the preprocessing stage. Therefore, having a real-time performance at the preprocessing stage, the edge-preserving filtering for instance, is essential to such tasks, to guarantee the real-time performance for the whole system.

However, in many cases, software implementation of such GIF algorithms may not be able to meet the performance requirement of real-time processing. Specifically, the guided image filtering becomes

slow when the frame size increases, which makes the real-time processing of full-HD video frames difficult. The more advanced GDGIF algorithm also introduces more complex computation steps in addition to the simple linear model computation, and more complex arithmetic operations for deriving the parameters, comparing to the original GIF. For instance, MATLAB simulation carried out in this work on a PC with a 4-core Intel i7 CPU shows that, a complete run takes 1.6841 seconds for the original GIF and 3.4000 seconds for the GDGIF to complete the processing a full-HD frame (1920×1080 pixels). The Python version gives a better speed of 0.6078 seconds for the original GIF, and 1.1952 seconds for the GDGIF. Even with the faster Python implementation, it is obvious that the processing speed is far behind the real-time performance requirement of 30 to 60 frames per second.

A highly efficient and compact VLSI design for the GDGIF may provide a good solution. Such design can be embedded in a computer vision system that demands high throughput or for a very high resolution video processing. VLSI circuits leverage higher hardware complexity for both higher computation speed and more internal data storage. To the best of the authors knowledge, there is no existing VLSI design for the GDGIF due to its computation complexity, although there are VLSI implementations of the GIF [4], [34]. Hence, a VLSI design of the GDGIF that meets the real-time processing requirement is one of the motivations of this work.

Modern computers and even mobile devices have multi-core processors, some possess advanced Graphic Processing Unit (GPU) that can carry out huge amount of computation workload. While these devices are highly efficient in parallel processing data, they are not the suitable for GDGIF. In GDGIF, massive pixel data in the filtering process has the characteristics of fine-grained parallelism. However, the shared global information across pixels requires additional processing. This fact makes the simple fine-grained parallelism provided by GPU much less effective. In addition, the GPU is bulky and power-hungry and not suitable for mobile applications. The coarse-grained parallelism provided by the multi-core CPUs shows even worse performance, as the abovementioned simulation results suggest. Field Programmable Gate Array (FPGA), as another option, has similar problems. FPGA also has high power consumption and is expensive for applications with high-volume production. Therefore, it is uncommon

to implement FPGA on mobile devices. As such, a VLSI design of the GDGIF is preferable to its equivalent GPU or FPGA version.

The challenges in designing a VLSI system for the GDGIF also motivate this work. The challenges include not only the computation of global parameters that requires all the pixels of a frame, which prevents pipeline processing and hinder throughput, but also the complex arithmetic operations such as square root, division and exponentiation, which cause long computation delays that jeopardize the processing speed and throughput, as well as incur high hardware cost. On one hand, the GDGIF is a good image filtering technique that is demanded in many real-time practical applications where fast processing speed and high throughput are needed. On the other hand, the VLSI implementation of a GDGIF poses the abovementioned design challenges. The demand and challenges motivate the work presented in this thesis.

1.3 Research Objectives

The overall objective of this research work is to develop a VLSI architecture design that is highly efficient in terms of both speed and hardware resources for fast gradient guided filtering process. The design is to meet the real-time performance and have comparable hardware complexity with existing GIF implementation. The objectives are detailed as below.

- (i) **A complete dataflow for VLSI design of the fast gradient guided filter is to be developed.**

Based on the original algorithm of the guided image filtering and its gradient domain development, a seamless dataflow is to be developed for feasible implementation of the efficient VLSI system. The dataflow plan is to allow efficient data accessing with low memory and hardware cost because the hardware implementation is different from the software algorithm that processing global parameters locally can be costly in hardware.

- (ii) **A compact VLSI architecture for the fast gradient guided filter (FGGF) is to be developed.**

The proposed architecture should be developed based on the planned dataflow, and allow the

trade-offs of hardware performance and complexity. The design is to achieve real-time processing with a high throughput. The design area is to be comparable to the existing architecture for the original GIF, with the additional computation functionalities included.

- (iii) **Real-time processing is to be guaranteed with performance bottlenecks addressed.** Since more information is included for the FGGF design comparing to the original guided filter, more complex arithmetic operations are required, resulting in slower processing speed and making the real-time processing requirement difficult to achieve. The performance bottlenecks of the design are to be analyzed and efficient arithmetic modules are to be designed to address the performance bottlenecks. According to [4], throughput of at least 30 frames per second or more should be guaranteed to achieve real-time processing.
- (iv) **Efficient divider circuits and exponential circuits are to be designed.** Division and exponential function are the two most complex arithmetic functions in the fast gradient guided filter system. Efficient circuits based on trade-offs of accuracy and speed performance are to be explored for these two functions in order to achieve the performance for real-time processing and at the same time maintaining the acceptable filtering quality. The area-delay product (ADP) of the two circuits should be better than the existing circuits and the circuits in Designware. The divider is to achieve 20% lower ADP. The fast gradient guided filter design is to achieve a higher throughput by at least 25% compared to the existing GIF design.
- (v) **Performance on the complete design is to be analyzed in detail.** The performance of the complete design, including the operating frequency, the throughput, the size as well as the quality of the filtering results is to be analyzed and assessed in detail.

1.4 Main Contributions

The main work presented in this thesis is on the newly proposed VLSI architecture for the fast gradient guided filtering system with the incorporation of a newly proposed non-iterative divider. The major

contributions are summarized as follows.

- **A VLSI architecture is proposed for the first time for the gradient guided image filtering.**

Although there is an existing VLSI design for guided image filtering [4] developed based on the original guided image filter algorithm, there is no VLSI design for the gradient guided image filtering. The gradient guided image filtering algorithm mitigates the problems of halo-artifacts and edge preservation that the guided image filter suffers, which is essential when gradient guided image filtering is applied to detail enhancement, high dynamic range compression, image matting/feathering, dehazing, joint upsampling, etc.

- **A dataflow scheme with preprocessing is introduced for the proposed FGGF design.**

One of the main challenges of designing such a VLSI system for gradient guided filtering is that several parameters used to derive the local linear model involves statistical results from the whole frame. Having these global parameters computed in the main stage would result in redundant computation and be time consuming, hence jeopardizing the speed and throughput of the system. The solution proposed is to have a quick scanning of the whole frame with proper down-sampling technique applied before the main computation stage. During the preprocessing, these global parameters are obtained from the down-sampled frame to have representative values, which are subsequently used by the filtering windows when processing the corresponding pixels in the main stage. Together with the output stage, a seamless dataflow is developed.

- **Down-sampling and parallel restoration are applied to balance computation cost and speed.**

Processing of full-HD (1920×1080) frame pixel by pixel can be costly in computation cycles. It is reported that proper down-sampling before a guided image filtering process will not result in visible degradation on the filtering output [36]. Hence, down-sampling is applied in both the preprocessing and the main computation stage for significant reduction of computation cycles required for processing a whole frame. The filtering algorithm also contains complex computations for dealing with edge-aware features, which results in long delay in the critical path and leads to

lower throughput than expected. A parallel output stage is proposed for two purposes. It restores the filtering output to the original frame size by applying the local linear parameters obtained in the down-sampled frame on all the pixels in the original frame. In addition, the output stage operates at a frequency that is 16 times higher and consequently significantly increase the overall throughput of the system.

- **Approximated circuits are designed and applied for speed up processing.** After the overall architecture of the proposed FGGF is developed, it is found that the longest delays are in the critical paths in several complex arithmetic modules, including the division and exponentiation in a parameter kernel. The delays in these modules create the main performance bottlenecks. To address the bottleneck issue, novel non-iterative dividers are proposed to speed up the processing at the sacrifice of computation accuracy. Optimization techniques are applied to trade off accuracy and area-delay products without compromising the quality of the filtering outputs. The required exponential function is analyzed and linear approximation is applied to replace the direct calculation. The inclusion of both the approximated circuits significantly improves the processing speed, hence the throughput, with almost no visible difference observed from the resultant outputs.
- **The performance of the whole design is analyzed.** The filtering quality, speed and hardware cost, are analyzed in details both qualitatively and quantitatively. The proposed design is found to achieve the expected filtering results at a throughput as high as 150 full-HD image (1920×1080 pixels) frames per second.

1.5 Organization of Thesis

The rest of the thesis is organized as follows.

Chapter 2 reviews the existing literatures related to the works presented in the thesis. The basic concept and formula of the guided image filtering is first introduced. The recent developments of the guided image filtering are summarized and discussed. In addition, the existing VLSI implementations

for guided filter and related bilateral filter are reviewed, with some essential techniques introduced to be employed in the proposed filter design later. Lastly, the arithmetic division circuits are discussed and the existing non-iterative dividers are reviewed.

Chapter 3 presents the proposed VLSI architecture for the gradient guided image filtering. The three main design challenges for a fast implementation of the gradient guided image filter, including the computation of the global parameters, the high computation intensity and the long delays in datapaths are discussed. A complete dataflow that includes a preprocessing step with down-sampling is introduced to address the challenges. Accordingly, the proposed VLSI architecture of the FGGF is presented with its three main stages described in details. The filtering quality, implementation results and hardware performance for the proposed design are evaluated and reported. Comparisons with the existing design of the original GIF are included.

Chapter 4 presents several techniques for designing fast non-iterative divider circuits, which are used to improve the speed performance of the FGGF design proposed in Chapter 3. The modelling of the fraction of the complex arithmetic division quotient as a curved surface is described. The curved surface representing the quotient is first partitioned and then approximated with small flat planes to reduce circuit complexity. Curve fitting method is first used to optimize the approximation error. Mixed integer linear programming (MILP) approach is also applied to obtain better accuracy. With the observation that non-linearity almost comes only from the mantissa of denominator, a stripe-based method is employed to reduce memory usage. The proposed methods and divider circuits are described. The trade-offs of accuracy and hardware complexity are evaluated and discussed.

Chapter 5 presents the improvements of the proposed FGGF design to achieve high processing speed and throughput. First the stripe-based processing scheme for a frame is discussed in more details taking the memory cost and computation cycles into consideration. Then the complex exponential module in the γ_k kernel of the FGGF is analyzed and replaced by a linear approximation module for size reduction and speed improvement. Incorporated with the approximated exponentiation module and the newly proposed non-iterative divider designs from Chapter 4, the filtering output quality of the design in the

application of detail enhancement is evaluated qualitatively and quantitatively. The use of different combinations of the approximated circuit modules in the proposed FGGF design are evaluated and reported.

Chapter 6 summarizes the works presented in the thesis and proposes some possible future research works related to the proposed FGGF design.

Chapter 2

Literature Review

2.1 Overview

The main objective of the thesis is on the hardware design of a fast guided image filtering system that can render filtering outputs with good quality, high computation efficiency and throughput for real-time applications. This chapter provides the background and theory for a thorough understanding of the evolution history of the guided image filtering. Literatures on the existing hardware implementations for guided filter and other related filter designs are reviewed as well.

In this chapter, the fundamental concepts of guided image filtering (GIF), together with its further developments in recent years are first reviewed in Section 2.2. The existing attempts on the hardware designs and implementations for the GIF are reported and summarized in Section 2.3. Some complex arithmetic operations that are involved in the design process, specifically the hardware implementation of division, are discussed in Section 2.4.

2.2 Guided Image Filtering

2.2.1 Concepts of GIF

The concept of the guided image filtering (GIF) is introduced in [27] and later summarized in [1]. GIF has become one of the mainstream edge-preserving smoothing method after the earlier bilateral image filtering algorithms. According to the convention from [1], the sources consist of a guidance image I and an input image p . The filtering process produces an output image q . In its local linear model, q is a linear transform of I as given by (2.1).

$$q_i = a_k I_i + b_k \quad \forall i \in \omega_k \quad (2.1)$$

where an output pixel q_i is a linear transformation of the guidance image I in a window ω_k with its center at pixel k , and a_k, b_k are constant coefficients. The rationale of guided filter maintaining edge information lies on the fact that the filtering result inherits differential of the guidance I , i.e., $\nabla q = a \nabla I$, as can be seen from (2.1) and in Fig. 2.1.

Figure 2.1: Concept illustration of the guided filtering process [1].

The filtering output can also be viewed as the filtering input image p with the texture information

or noise removed. Linear regression is applied upon the filtering input to derive the local linear model with its parameters a and b . In this way, (2.2) is employed as the cost function [1].

$$E(a_k, b_k) = \sum_{i \in \omega_k} ((a_k I_i + b_k - p_i)^2 + \lambda a_k^2) \quad (2.2)$$

where the parameter λ prevents a_k from being too large as can be seen from the solutions of (2.2) for a_k and b_k given in (2.3) and (2.4) respectively.

$$a_k = \frac{\mu_k(Ip) - \mu_k(I)\mu_k(p)}{\sigma_{k,r}^2(I) + \lambda} \quad (2.3)$$

$$b_k = \mu_k(p) - a_k \mu_k(I) \quad (2.4)$$

where $\mu_k(\tau)$ given in (2.5) is the averaged value of a specific expression τ within a square window centered at k with radius r by default, if not specified with a suffix 1.

$$\mu_k(\tau) = \frac{\sum_{i \in \omega_k} \tau}{(2r+1)^2} \quad (2.5)$$

With the guidance of this local linear model, the guided image filter can provide better quality for edge-preserving smoothing operation, compared to traditional linear translation-invariant filters like Gaussian [28], [29] and Laplacian filters [30], [31]. It has better behavior at edges [27], overcoming the problem of gradient reversal artifacts commonly observed in bilateral image filters (BIF) [37], as illustrated in Fig. 2.2. The gradient reversal artifacts are often observed in detail enhancement and High Dynamic Range (HDR) compression, because the base layer (filtering output) derived by bilateral filter may not be consistent with the filtering input p due to its Gaussian kernel unreliably averaging neighboring pixels.

Furthermore, the guided image filter is a fast and non-iterative algorithm with a time complexity of $O(N)$, depending on the total pixel numbers N in a frame but independent of the filter kernel size.

Figure 2.2: Comparison between bilateral filter (BIF) and guided image filter (GIF) on gradient reversal artifacts, simulated by using the algorithm provided by [1]. (a) BIF. (b) GIF. (c) BIF. (d) GIF.

The quality improvement and higher efficiency enable the guided image filter to perform well in many applications including edge-aware smoothing [38], [39], detail enhancement [40], [41], image reconstruction [42], [43], [44], HDR compression [45], image matting/feathering, dehazing [46], [47], [48], [49], stereo matching [34], [50], [51], [52], saliency detection [53], [54], image classification [55], etc.

Examples for edge-preserving smoothing are given in Fig. 2.3, where the kitty image are smoothed using the GIF with different parameter settings, which are not discussed further in detail. The edge-preserving filtering property of the GIF is self-explained from its local linear model in (2.1).

For simplicity, it is assumed that $I = p$, i.e., the guidance image is also the input image. At the edge areas, where the pixel intensities varies a lot within a filtering window, the term $\sigma_{k,r}^2(I)$ is very large comparing to the constant λ . In this way, a_k approaches 1 and b_k approaches 0, and the edge is preserved. On the other hand, at flat areas where the pixel intensity does not change much within a filtering window, the term $\sigma_{k,r}^2(I)$ becomes negligible comparing to the constant λ . This results in a_k approaching 0 and b_k , and q_i approaches $\mu_k(p)$ or $\mu_k(I)$, hence the flat area is smoothed.

The detail enhancement and HDR compression are conducted in similar ways. The former is conducted by magnifying the detail layer derived as the difference between the filtering input and output ($I - q$) [56], while the latter by compressing the base layer derived as the filtering output (q) [57]. Fig.

Figure 2.3: Edge preserving smoothing using guided filtering with different parameter selections [1].

2.4 shows an example application of the HDR compression, comparing the results from bilateral filter and GIF. Notice that there are the quantization artifacts of the result by the bilateral filter, but they are avoided when using the GIF. It is reported in [1], [3] that gradient reversal artifacts may also occur for HDR compression.

The guided image filtering process can be applied in various application fields, showing good quality and high efficiency. Many other examples can be given for the abovementioned applications using the GIF. The main interest of this thesis is focused on the edge-preserving filtering quality with halo-artifacts reduced. The detail enhancement is chosen for most of the illustrations in later chapters, since the edge information is reflected obviously in this application.

On the other hand, the guided image filter has its own filtering quality problem, which is known

Figure 2.4: HDR compression comparisons for quantization artifacts [1]. (a) Input HDR image. (b) Compressed image with bilateral filter. (c) Compressed image with guided filter.

as the halo-artifacts. Halo-artifacts are referred to as the artifacts of unwanted smoothing at edges [1], which usually occur at regions where high intensity texture area is smoothed. The halo-artifacts are contrast to the gradient reversal artifacts, which means unwanted sharpening of edges.

Fig. 2.5 is an example of this phenomenon that when strong textures are to be smoothed, the weaker edges would also be smoothed [1]. Both BIF and GIF have this problem, while global optimization filters such as Weighted Least Squares (WLS) filter [58] distribute such blurring more globally. The halos are suppressed at the price of global intensity shifting. As can be seen from this example, the halo-artifacts may cause some degradation of structural information after the filtering process. Thus many research works have been done to reduce the halo-artifacts.

2.2.2 Developments of GIF

To improve the quality of the guided image filtering, a weighted guided image filter (WGIF) is presented in [2] to achieve a new local filter as fast as the guided image filter, but at the same time edges are preserved as good as in a global optimization filter, like the weighted least square filter [58]. In the WGIF, an image $I(p)$ is decomposed into a base layer $\hat{I}(p)$ and a detail layer $e(p)$ as given in (2.6).

Figure 2.5: Illustration for halo-artifacts after smoothing with different filters [1]. (a) Input image. (b) Guided filter. (c) Bilateral filter.

$$I(p) = \hat{I}(p) + e(p) \quad (2.6)$$

Using the same local linear transform as the original guided image filter in (2.1), the cost function of the WGIF remains the same as (2.2). It can be observed that the data term, $(a_k I_i + b_k - p_i)^2$, measures the fidelity of the base layer, and the regularization term provides smoothness. The halo-artifacts are caused by the fixed regularization term (λ) [59]. To reduce the halo-artifacts, an edge-aware weighting given in (2.7) is introduced.

$$\Gamma_k(I) = \frac{1}{N} \sum_{i=1}^N \frac{\sigma_{k,1}^2 + \varepsilon}{\sigma_{i,1}^2 + \varepsilon} \quad (2.7)$$

where $\sigma_{k,1}^2$ and $\sigma_{i,1}^2$ are the variances of pixel intensities centered at pixel k and i respectively, and the ε is a small positive constant to prevent the division by zero. Thus, the cost function becomes (2.8).

$$E(a_k, b_k) = \sum_{i \in \omega_k} ((a_k I_i + b_k - p_i)^2 + \frac{\lambda}{\Gamma_k(I)} a_k^2) \quad (2.8)$$

This weighting is a measure of the importance of a pixel k , with respect to the whole guidance image. Its value is larger than 1 at edges, but less than 1 otherwise, realizing the edge-awareness.

Fig. 2.6 shows three examples for dehaze applications, for input images with haze taken at three cities of Beijing, Shanghai and Singapore. Though dehazed results are derived by both the original GIF and WGIF, obvious halo-artifacts can be spotted at column (b) for GIF, as discussed in Section 2.2.1. The results rendered by the WGIF show much better quality with halo-artifacts reduced significantly.

Figure 2.6: Enhancement result of WGIF comparing to others [2]. (a) Input image with haze. (b) Dehazed result by GIF. (c) Dehazed result by WGIF.

Inspired by the adaptive bilateral filter (ABF) [60], the shift-variant technique is integrated into the guided image filter to form the adaptive guided image filter (AGIF) [61]. The adaptive guided filter is designed to enhance edge slopes without producing halo-artifacts, and to reduce noise while preserving edges, thus image sharpening and denoising are achieved simultaneously.

The adaptive filter kernel under this method consists of a domain and a range filter. The shift-variant technique includes an offset in the range filter portion. The range filter is interpreted as a 1-D filter that processes the image histogram, where the offset can be shifted locally. The four shift operations are: 1) no shift; 2) shift towards the mean; 3) shift away from mean and 4) shift towards MAX/MIN values in the specific window. The objective of this approach is to adaptively adjust the offset and the constant parameters so that the algorithm will smooth pixels in homogeneous regions and sharpen pixels that lie on edge slopes.

Another method that emphasizes edge-aware adaptive filtering is the gradient domain guided image filtering (GDGIF) [3]. GDGIF is developed based on the WGIF [2]. It has deeper insight on the performance bottleneck and is a more effective solution. The typical applications of GDGIF include image detail enhancement, tone mapping of HDR images and image saliency detection.

It is stated in [3] that the linear model of the original GIF cannot preserve the image information well near some edges of structures. In addition, there are halo-artifacts near edges. These phenomena can be most apparent and obviously observed in the application of detail enhancement, the aim of which is to emphasize structure information.

Even though the edge-aware weighting is used in the WGIF [2] to represent the importance of a specific pixel with respect to the whole frame, it is still not able to handle edge structures when there is detailed pattern background. For such cases, the detailed pattern background might be mistaken as important structure edges and preserved. These undesirable filtering results should be avoided and the solution is to convert the global comparison into a combination of both global and local comparison to determine the importance of a specific pixel. Thus, the original edge-aware weighting in (2.7) evolves into (2.9).

$$\Gamma_k(I) = \frac{1}{N} \sum_{i=1}^N \frac{\chi_k + \varepsilon}{\chi_i + \varepsilon} \quad (2.9)$$

where

$$\chi_k = \sigma_{k,r} \sigma_{k,1} \quad (2.10)$$

and $\sigma_{k,r}$ and $\sigma_{k,1}$ are the local standard variances of a radius- r and a radius-1 filtering window respectively. A radius- r window is a window with size $(2r+1) \times (2r+1)$ and a radius-1 window is a 3×3 window. Since both the evaluation of intensity variation within the local region and the whole frame are taken into account, edges are more precisely located according to the multi-scale test in variances. This new edge-aware weighting is designed in such a way that it can better measure the importance of pixel k within the whole guidance image I [3].

The fact $\nabla q = a \nabla I$ also reveals that the smoothness of the filtering output relies on the value of a_k , implying that the closer a_k is to 0, the more smooth the region will be, and the closer a_k is to 1, the better preserved the edge information is. The parameter γ_k , as defined in (2.11), in the new cost function is set as an explicit first-order edge-aware constraint for regulating the value of a_k .

$$\gamma_k = 1 - 1 / (1 + e^{\frac{4(\chi_k - \mu_\chi)}{\mu_\chi - \min(\chi)}}) \quad (2.11)$$

where μ_χ is the mean of all χ_k . γ_k is formulated such that it approaches 1 if a pixel is in an edge area, but 0 in a flat region. In this way, the value of a_k is controlled by this first-order constraint, resulting in more complicated computation of the coefficients a_k and b_k as given in (2.13) and (2.14) respectively.

For better edge preserving in the resultant image based on the guidance image, the cost function (2.2) is upgraded to (2.12) in the GDGIF [3], adopting the newly added multi-scale edge-aware weighting of $\Gamma_k(I)$ and the explicit first-order constraint γ_k .

$$E(a_k, b_k) = \sum_{i \in \omega_k} ((a_k I_i + b_k - p_i)^2 + \frac{\lambda}{\Gamma_k(I)} (a_k - \gamma_k)^2) \quad (2.12)$$

where $\Gamma_k(I)$ is defined in (2.9) using multi-scale variances and γ_k is the first-order constraint in (2.11).

From the evolved version of the cost function (2.12), the linear coefficients, a_k and b_k , of the local model can be derived as below.

$$a_k = \frac{\mu_k(Ip) - \mu_k(I)\mu_k(p) + \frac{\lambda}{\Gamma_k(I)}\gamma_k}{\sigma_{k,r}^2(I) + \frac{\lambda}{\Gamma_k(I)}} \quad (2.13)$$

$$b_k = \mu_k(p) - a_k\mu_k(I) \quad (2.14)$$

As can be seen from the relationship of the local windows ω_{k_j} (dotted-border boxes) and a pixel i (at the center of the solid-border filtering window) in Fig. 2.7, the output q_i ((2.1) of the pixel i will have many values because there are many a_k and b_k values in the filtering window of radius r . To have one unique q_i for each pixel i , the average values of the coefficients a_k and b_k , $\bar{a}_i$ and $\bar{b}_i$ as given in (2.16), are used to derive q_i as in (2.15).

Figure 2.7: Relationships among filtering windows in an image.

$$q_i = \frac{1}{|\omega|} \sum_{k: i \in \omega_k} (a_k I + b_k) = \bar{a}_i I + \bar{b}_i \quad (2.15)$$

where $\bar{a}_i$ and $\bar{b}_i$, $\bar{a}_i$ and $\bar{b}_i$ as given in (2.16), are the average values of a_k and b_k respectively.

$$\bar{a}_i = \frac{1}{|\omega_i|} \sum_{k \in \omega_i} a_k; \quad \bar{b}_i = \frac{1}{|\omega_i|} \sum_{k \in \omega_i} b_k \quad (2.16)$$

Fig. 2.8 shows the smoothing results using different guided image filters, including the original GIF, the WGIF and the GDGIF. One line of pixels from a middle row of the source image displayed is

used to conduct such a 1D intensity analysis. As can be seen from Fig. 2.8 that the WGIF and GDGIF preserve the edges better than the original GIF. From the zoomed-in viewports in the dashed boxes, the GDGIF filtering result shows about identical shape of the input image at edge areas, while the original GIF and WGIF show distances from input at different extent. Hence, it is demonstrated that the edge-aware weighting and first order constraint together push the results closer to input data at edge areas, thus improves the edge-preserving property based on the WGIF.

Figure 2.8: Intensity 1D analysis on filtering results from different filters, the input data comes from a middle row of the source image displayed [3].

There are more and more research works on the guided image filtering, its application in different fields and hardware implementation for those applications requiring high performance. The algorithm [62] tries to control the underlying image structure at different scales, formulating the guided image filtering as nonconvex optimization problem, solved by the majorize-minimization algorithm. Two independent guidance for structure and texture are provided in [63] to balance the trade-off between preserving structure and suppressing texture. In [64], different cost terms based on image content around pixels are assigned for fast and accurate disparity estimation. In addition to the effort to improve the processing quality of the guided image filtering, there are also more hardware implementations of the guided image filters, including [65], [66], [67], [68] and as well others, which are discussed in details in the next section.

2.3 Hardware Implementation of Guided Image Filtering

The original guided image filtering (GIF) algorithm defined in (2.1)-(2.4) was implemented in a VLSI architecture as shown in Fig. 2.9 from [4], where several techniques were proposed to deal with the challenges in hardware mapping, including an integral image engine for reducing the computation cost when computing the sum of the pixel values in the filtering windows at all stages. It also proposed a double integral image architecture for trade-off of data bandwidth and on-chip memory, and a stripe-based method for efficient processing of large image frames. In the stripe-based method, a frame is divided into several stripes of manageable size and the computation is performed stripe by stripe instead of on the whole frame, reducing the computation complexity and memory requirement [4].

Figure 2.9: A VLSI architecture for the original GIF [4].

To process all the pixels in a stripe of a frame, the design operates at four stages applying the techniques mentioned above. The computation stages are summarized below.

- (i) **The Integral Image Engines (IIEs) compute the sum of windows.** As described in Section 2.2, the pixel-wise values for guided image and input image, including I , p , I^2 and $I \times p$ are

obtained first. Then the IIEs compute the corresponding sum within the filtering window and store the values obtained in the attached on-chip memory at this stage.

- (ii) **Coefficient Kernel Engine derives the linear parameters.** The parameters, a_k and b_k , for the local linear model can be derived according to (2.3) and then (2.4) in the Coefficient Kernel Engine when the corresponding integral values are ready in the on-chip memory.
- (iii) **The Integral Image Engines (IIEs) compute the sum of linear parameters.** The averaging process of the derived linear parameters a_k and b_k follows the same process as the sum of pixel values in the filtering windows, as in the first stage.
- (iv) **Output Kernel Engine computes the filtering output.** After the averaged parameters are obtained, the corresponding filtering output can be computed straightforwardly, according to (2.15) pixel by pixel.

In the first and third stage, the IIE plays an essential role in getting the integral value from all the pixels located in a filtering window, scanning pixel by pixel within the current stripe. The idea of the IIE design is extended from the histogram calculation proposed in [5]. As can be seen from Fig. 2.10, the integral process for the square $ABCD$ in Fig. 2.10(c) can be easily computed with the integral values of the rectangle area OX in Fig. 2.10(a), where X may stand for A , B , C , or D . Each of them can be naturally obtained with the neighboring integral values in the memory, when a filtering window scans through the area as illustrated in the memory view of Fig. 2.10(b). Similar mechanism is borrowed in [4] for a highly efficient implementation of the IIEs. This implementation of the original GIF is a scalable hardware design, of which the parameters can be customized for different applications.

The stripe-based method decomposes an input frame into several vertical stripes. At each cycle, proper pixel data are fed by the IO controller to the first stage to carry out the sums of I , p , I^2 and $I \times p$. The results are stored and transferred to the second stage for deriving the linear parameters a_k and b_k . The second stage not only provides the parameter values to the third stage, but also sends the parameters

Figure 2.10: Illustration of integrating process by scanning [5].

back to the IO controller to be stored in the memory, so that the following processing stage can retrieve these delayed parameter values for integral process. After all the averaged parameters are derived, the filtering output can be calculated according to the linear model (2.1).

It should be noted that the a_k parameter kernel does not follow exactly the computation in (2.3). A refactored expression given in (2.17) is used.

$$\begin{aligned}
 a_k &= \frac{\mu_k(Ip) - \mu_k(I)\mu_k(p)}{\sigma_{k,r}^2(I) + \lambda} \\
 &= \frac{\mu_k(Ip) - \mu_k(I)\mu_k(p)}{\mu_k(I^2) - \mu_k^2(I) + \lambda} \\
 &= \frac{|\omega| \sum_{\omega_k} Ip - \sum_{\omega_k} I \sum_{\omega_k} p}{|\omega| \sum_{\omega_k} I^2 - (\sum_{\omega_k} I)^2 + |\omega|^2 \lambda}
 \end{aligned} \tag{2.17}$$

This reformation of a_k reduces the number of divisions from 4 to only 1. The reformation is crucial as the division is a complex operation as discussed in the next section. The divisions are also a performance bottleneck of the whole design. The reduction of dividers not only simplifies the computation

complexity but at the same time reduces the potential error occurred due to the truncations during divisions. The GIF design can operate at the frequency of 100MHz , achieving a frame rate of 30 frames per second for full-HD (1920×1080) frame. This design has similar performance but smaller circuit area and on-chip memory comparing to the bilateral filter design in [5].

The original GIF was also implemented in FPGAs as a sub-module for the application of stereo matching [34], where high quality real-time edge-preserving filtering is required. The efficient design of the GIF sub-module in the stereo matching system can be used in other applications too. The stereo matching architecture provides high accuracy and performance for embedded vision applications, achieving a frame rate of 60 frames per second for $720p$ frames. Moreover, it demonstrates that the guided image filter can be incorporated as a sub-module for more complex real-time image processing applications.

2.4 Implementations for Arithmetic Divisions

Basic arithmetic operations have long been implemented in hardware architectures for complex applications that rely on massive and repetitive invocation of such operations at lower level [69]. Though adder(subtractor) and multiplier circuits have been studied and crafted into highly efficient arithmetic units with high performance and low cost, the design of divider circuits has attracted much less research due to lower usage than adders and multipliers [70] and due to the more complicated and difficult operations involved. Due to its complexity, divider is often a performance limiting module in many applications such as digital signal processors (DSPs) [71], [72]. This is also the case for other systems where real-time processing is required, especially in graphics and video processing [73]. The lack of fast and compact division module in integrated circuits has for long time been the bottleneck for some real-time multimedia applications, such as high performance graphics rendering pipelines [74], [75] as well as the image filters discussed in earlier sections.

Traditionally, division is achieved with two approaches, namely, shift-subtract and iterative convergence [70], [76], [77], [78], [79], [80]. The first category is also known as digit-recurrence, evolving

from the concept of manual long division to obtain one or several digits of the quotient each cycle [77], [81], [82], [83]. During the recurrence process, either restoring or non-restoring technique can be used, depending on whether the partial remainder needs to be restored if subtraction is unsuccessful [76]. The well-known SRT (Sweeney, Robertson and Tocher) method [84], [85], [86], [87], [88] is a representative speedup based on the shift-subtract approach, in which more quotient bits are retrieved each cycle. The SRT division is widely used in modern floating point units [84], [89], [90].

On the other hand, the iterative convergence approach tries to estimate an initial approximation, then converges to the actual result iteratively based on a functional algorithm narrowing the error along. The Newton-Raphson method [91], [92], [93], [94] is a typical example, which generates the reciprocal of the denominator by an iterative process, and finally obtains the quotient by multiplying it with the numerator [76].

Combinational arrays are also employed for division as they are for multiplication [95], but the rows in the division array are not independent of each other, preventing parallel processing. In floating-point division, the three operations, determining the sign, dividing the fraction and subtracting the exponent, are performed concurrently according to the IEEE 754 floating-point standard [96].

It is believed that the nature of division lends itself to speedup technologies less easily as multiplication does [76]. There are several specific problems that make division more difficult to be efficiently implemented. The potential high dynamic range of the result from division will sometimes demand for a large wordlength in fixed-point computation, which for other cases might be more than necessary. Besides, discontinuity from divide-by-zero requires special handling, which is less of a problem for other operations. Most critically, the recursive subtraction or iterative convergence nature of division results in the fact that series of intermediate results reply upon one another in a sequence, making the simultaneous computation that is applicable in some multiplication implementations impossible in division. The described prevailing implementations suffering from these problems, making the performance of division inadequate for some real-time applications such as 3D graphic computing [97] and the guided image filters discussed earlier.

Figure 2.11: Logarithm and anti-logarithm approximation [6]. (a) Piecewise linear (PWL) approximation for logarithm. (b) Piecewise linear approximation for anti-logarithm.

Many real-time applications have a considerable tolerance for computation accuracy, providing possible trade-off among accuracy, computation speed and hardware complexity. One such example is the 3D graphic processing [6], where multiplications and divisions are simplified as addition and subtraction operations in logarithmic domain using the Mitchell-based logarithmic approximation [98], [99].

The logarithm and anti-logarithm operations in [6] are conducted using the piecewise linear approximation as shown in Fig. 2.11 to convert the input values into logarithmic domain, and back to normal domain. As can be seen, the conversion is done for the fraction part ($1+m$) of single precision floating numbers. Five segments are used for logarithm and four for anti-logarithm.

Operation		Normal Arithmetic	Logarithmic Arithmetic
Multiplication	MUL	$z = x \cdot y$	$X+Y$
Division	DIV	$z = x / y$	$X-Y$
Reciprocal	RCP	$z = 1 / x$	$-X$
Square Root	SQRT	$z = \sqrt{x}$	$X \gg 2$
Reciprocal Square Root	RSQ	$z = 1 / \sqrt{x}$	$-X \gg 2$
Square	SQR	$z = x^2$	$X \ll 2$
Powering	POW	$z = x^y$	$Y + \log_2 X$
Addition	ADD	$z = x + y$	$X + \log_2(1+2^{Y-X})$
Subtraction	SUB	$z = x - y$	$X + \log_2(1-2^{Y-X})$

Figure 2.12: Logarithmic arithmetic units. (a) Three stage logarithmic arithmetic unit architecture[7]. (b) Single stage division unit [8].

A similar logarithmic arithmetic unit (LAU) is reported in [7]. With such LAU as shown in Fig. 2.12(a), the first step is always to convert the normal number representation into the logarithmic domain. Then the normal arithmetic operation listed in the table can be conducted equivalently in the logarithmic

domain with operations such as addition, subtraction and shift operation using Barrel Shifter (BSH) that are usually much easier to implement in hardware.

For many situations in image/video processing, the approximated computation would not cause significant visual degradation in the results, while at the same time offers performance improvement and hardware saving [74]. As for division, non-iterative dividers have been proposed to improve computation performance with reasonable approximation [7], [8], [100], and with accuracy and computing speed improved along. Kim et al. [7] realize a 3-stage binary logarithm based division circuit for their fixed-point 3D graphics system. The design has a 0.2% MAEP (maximum absolute error percentage). Low and Jong [8] propose a single stage divider based on the Mitchell logarithmic method as shown in Fig. 2.12(b), achieving similar accuracy with both the CPD (critical path delay) and ADP (area-delay product) reduced.

2.5 Summary

In this chapter, the concept of the guided image filtering is introduced and its model is reviewed in details for later analysis. The developments on the GIF in recent years are also reviewed, focusing on the filtering quality at edge preserving. Existing hardware designs for GIF and related local filters are summarized as the base for the proposed design for more advanced fast FGGF. Since the complex arithmetic division is involved, the operation of division is examined for different approaches of implementations. Non-iterative methods such as logarithmic arithmetic unit are introduced to provide insights for later designs.

Chapter 3

Fast Gradient Guided Image Filter Design

3.1 Overview

This chapter describes the design of the proposed fast gradient guided image filter in details. As stated in Chapter 1, the main goal of this work is to have a complete and highly efficient VLSI architecture design in terms of both speed and hardware resources for the GDGIF. First, the main design challenges of gradient guided filter are analyzed and addressed in Section 3.2, including the high computation complexity and the involvement of global information in local processing steps. Second, based on the original algorithm for the guided image filtering process [1], [27], its various developments for better filtering quality [2], [3], [59], [61], and higher computation efficiency [4], [34], [36], a solution that consists of three major stages, employing the down-sampling technique and parallel processing structure is proposed with complete dataflow organization described in Section 3.3. Then, the VLSI architecture of the proposed fast gradient guided filter (FGGF) design is presented in Section 3.4 according to its three main stages, namely the preprocessing, linear coefficients computation and parallel output stages, with detailed structure and modules. The performance of the proposed design in terms of both filtering quality and hardware efficiency is evaluated in Section 3.5, the potential design space for introducing approximated computation modules for better efficiency is also discussed.

Based on the literature review, it is noted that existing hardware designs focus only on the original guided image filter, the proposed FGGF design is the first VLSI design for a high performance realization for the complex gradient domain guided image filter (GDGIF) algorithm, which reduces the halo-artifacts in edge-preserving filtering. In addition, the FGGF VLSI architecture for fast gradient domain guided image filtering has high throughput that satisfies the requirements of real-time applications. There are two main novelties that are applied in the design process of the proposed FGGF architecture in this chapter.

- **Multi-scale down-samplings are adopted to reduce computation cost.** In the preprocessing stage, a higher sampling ratio is applied to extract the global information, than that in the later stage for local models. The higher sampling ratio in the preprocessing stage reduces the hardware resources significantly, while maintaining the accuracy of statistical parameters in large a frame. At the same time, the lower sampling ratio, reveals more detailed variation in each of the scanning windows, avoiding image features being over smoothed. Down-sampling of pixel values with different ratios in the first two stages significantly reduce the computation cost in terms of both circuit size and processing time, without visually hampering the filtering quality.
- **Parallel structure with higher operating frequency is used to achieve high throughput.** At the output stage, full image size is restored from reduced linear coefficients due to down-sampling in the earlier stages. Parallel structure is employed and the same linear coefficient is used for all the sixteen neighboring pixels in a scanning window. An operation frequency sixteen times higher than the operating frequency in the earlier stages is applied at the output stage. With this approach, high throughput is achieved due to this parallel processing with higher frequency.

With the novel techniques mentioned above applied, the main contributions of the work in this Chapter include the followings:

- **Dataflow of the proposed FGGF is developed.** The algorithm for the gradient guided filtering is partitioned into three separate steps to handle the workload more efficiently so that hardware saving and performance improvement are achieved.

- **A complete VLSI architecture with three stages is designed.** The problem of the global information required in each local model is solved by introducing a preprocessing stage for fast computation of the global parameters. The main and complex task is conducted in the linear coefficient stage. An output stage is responsible for restoring the image size and boost throughput.
- **The performance bottleneck of the proposed design is evaluated for further optimization.** The analysis in the critical path delay for the proposed filter design gives insights for replacement of some complex computation modules with approximated processing to achieve higher computation efficiency.

The simulation of the proposed FGGF for the application of detail enhancement is conducted, comparing with the original GIF. Better quality of the resultant images is shown when dealing with structural information with high intensity variation. The proposed FGGF architecture is synthesized with Synopsys Design Compiler in the STM 90nm and 65nm CMOS technology. The results show that the proposed filter is able to support Full-HD (1920×1080) image filtering to achieve real-time processing, with higher performance than the existing hardware implementation [4] for the original guided image filter (GIF).

3.2 Design Challenges

As described in Chapter 2, the gradient domain guided image filter (GDGIF) is a novel guided filter which addresses the problem of halo-artifacts of original GIF when processing edges in images [3]. However, there is no existing VLSI architecture for the GDGIF algorithm, defined in (2.1), (2.9)-(2.16). The better filtering quality on edges and structural information comes at the price of high computation complexity. To implement the GDGIF in VLSI for real-time applications, there are three main challenges as discussed below.

- (i) **Global information is involved in local models.** With the introduction of the novel edge-aware weighting $\Gamma_k(I)$ and the explicit first-order constraint γ_k in GDGIF that involve χ_k (2.9)-(2.11),

pure sliding window approach adopted in the existing designs [4], [34] does not satisfy the requirement as the derivation of the linear coefficients depends on the global information, i.e., the pixel values in the whole frame rather than only in the local filtering window of radius r . The processing of such global information can significantly increase the computation workload, losing the main advantage of high efficiency of a local filter [32], [101], and the evolved algorithms [1], [2], [57], [102], [103], that aim for less artifacts, making it even more difficult to achieve real-time processing.

- (ii) **Higher computation intensity is required.** The parameters $\Gamma_k(I)$ (2.7) and γ_k (2.11) introduced in GDGIF demand more intensive computation at each pixel compared to the original process of (2.3) and (2.4). Specifically, three divisions are needed for the computation of the coefficient a_k (2.13). Moreover, it also depends on the computation of γ_k (2.11), which involves operations of reciprocal, exponentiation, division, averaging and computing minimum. Despite the need for global information, the mutual dependencies of the operations mentioned above could make it difficult for the proposed design to achieve real-time performance. The intermediate computation results will also cause longer delay in the critical datapath. Such problem cannot be solved by simple pipelining to increase the clock frequency as pipelining requires more registers and increases the latency of the replaced module and there is always a limit on the number of pipeline stages. Moreover, some complex arithmetic operations are difficult to partition into finer stages naturally. For instance, the optimization of integrating process of pixel values in a scanning window depends on analyzing both static and dynamic elements. In this case, if only pipelining technique is used, not only latency and registers would be increased, the hardware complexity could not be reduced.

- (iii) **Complex operation limits the operating frequency.** Though pipelines can be introduced to increase the overall operating frequency of the filter system, some complex operations such as exponentiation, division and square root are involved, which hamper the overall processing speed, due

to the long delay within these modules. The longest delay in these complex arithmetic modules can be the bottleneck for the overall operating frequency, which directly relates to the throughput in pixels per second, thus impeding the real-time processing.

To address the above challenges in the hardware implementation of the FGGF, several approaches are proposed.

First, the problem of generating intermediate global parameters in the GDGIF algorithm [3] is addressed in the proposed design by introducing a preprocessing stage before deriving the linear coefficients of a_k and b_k . Though this preprocessing will require extra clock cycles in addition to the original main computation time, the repetitive global information derivation is moved outside the linear coefficients computation stage for each pixel within each corresponding filtering window, avoiding the redundant workload at that stage.

Second, down-sampling technique is adopted to save computation cost both when deriving the global information and when determining the linear coefficients. Though partitioned into two parts, the preprocessing and the main computation, the cost of processing each pixel in either part remains high. Down-sampling by a ratio of s can speed up the processing by s^2 times, with almost no visible degradation in the resultant image [36]. Hence, down-sampling technique is applied in the proposed design in both the preprocessing stage and the main stage for deriving the linear coefficients. In the preprocessing stage, higher sampling ratio is used to obtain the global statistical estimate of a frame. Lower sampling ratio is employed when computing the linear coefficients in the main stage for better local variations.

Last but not least, parallelism is applied at the restoring output stage to achieve high throughput [104]. Due to subsampling, only part of the pixels are selected for computing the coefficients a_k and b_k in the main stage. Upsampling is required when applying the coefficients on all the pixels in the original frame to restore the output to a full image. As only simple operations are required in the output stage, a higher frequency can be used at this stage. Consequently, the overall throughput is increased as the throughput is determined by the operating frequency at the output stage, even though the complex

operations in the main stage might have long delays. In this way, the frame rate is increased significantly due to this high frequency employed at the parallel restoring output stage.

The details of the proposed FGGF design are described in the following sections.

3.3 Dataflow Organization

According to the approaches described in the last section, the dataflow of the proposed fast gradient guided filter (FGGF) is partitioned into three main processes, namely the preprocessing stage, the linear coefficients computing stage and the parallel restoring stage, as shown in the block diagram in Fig. 3.1. As illustrated in Fig. 3.1, highly down-sampled (sampling ratio s_0) pixels of the guided image I is fed into the preprocessing stage at an operating frequency f_o (the lower frequency) to quickly get the global information needed for later stages. Then, with a lower sampling ratio s_1 , the proposed filter uses both the guided image I and the input image p , together with the computed global parameters to determine the linear coefficients a_k (2.13) and b_k (2.14) within each filtering window. Once the linear coefficients are set, the parallel restoring stage accesses the guidance at a much higher frequency $f_a (= s_1^2 f_o)$ to restore the filtering output to full frame size.

Figure 3.1: Proposed dataflow for the gradient guided filtering.

The detailed dataflow of a complete process of the proposed gradient guided filter is given as the procedure *FastGradientGuidedFilter* in Algorithm 1, which executes three sub-procedures, *PreProcess*,

LinCoef and *Restore* in Algorithms 2 - 4, representing the three stages described above. The constants λ and ε correspond to those described in Equations (2.3)-(2.9) in Chapter 2. Within the main procedure of *FastGradientGuidedFilter*, the values other than the inputs and the output are the intermediate outputs of the sub-procedures, which are explained in the following subsections and sub-procedures.

Algorithm 1 Fast gradient guided filtering

Input: Guided image I , filtering input image p , down-sampling rates s_0 and s_1 , sliding window radius r , constants λ and ε

Output: Guided filtering result q

- 1: **procedure** FASTGRADIENTGUIDEDFILTER($I, p, s_0, s_1, r, \lambda, \varepsilon$)
  - 2: $mean_\chi, min_\chi$ and $\tilde{\chi}_{i,s_0} \leftarrow PreProcess(I, s_0, r, \varepsilon)$
  - 3: $\bar{a}$ and $\bar{b} \leftarrow LinCoef(I, p, s_1, r, mean_\chi, min_\chi, \tilde{\chi}_{i,s_0}, \lambda, \varepsilon)$
  - 4: $q \leftarrow Restore(I, \bar{a}, \bar{b})$
  - 5:   **return** q
-

3.3.1 Preprocessing Procedure

The preprocessing step of the proposed dataflow is described as the procedure *PreProcess* in Algorithm 2. As can be seen, this procedure derives the three global estimated parameters from the guided image I down-sampled by s_0 , including the minimum value $mini(\chi_{s_0})$ (2.10), the arithmetic mean value $mean_\chi$ of χ_{s_0} and the harmonic mean value $\tilde{\chi}_{i,s_0}$, which are the global parameters needed in (2.9) and (2.11), where the harmonic mean is represented by the term $\tilde{\chi}_{i,s_0} = (sum(1/(\chi_{s_0} + \varepsilon))/N_{s_0})^{-1}$.

As can be seen from Algorithm 2, down-sampling is first carried out on I at a sampling ratio of s_0 to reduce the total clock cycles needed for computing the global parameters. The operation $f_{box}(x, r)$ represents a box filtering process that averages the pixel values of x within a window of radius r . Lines 3 to 8 in the procedure compute the variance of I for different radius, i.e., r and 1, then derive the χ values of the down-sampled guided image, before computing the global parameters, i.e., min_χ , $mean_\chi$ and $\tilde{\chi}_{i,s_0}$.

Algorithm 2 Fast gradient guided filtering - Preprocessing

Input: I, s_0, r, ε **Output:** Mean of χ values $mean_\chi$, minimum of χ values min_χ , harmonic mean of χ value $\bar{\chi}_{i,s_0}$

```
1: procedure PREPROCESS( $I, s_0, r$ )
2:    $I_{s_0} \leftarrow$  down-sampling  $I$  by  $s_0^2, r = r/s_0$ 
3:    $mean_{1,I_{s_0}} \leftarrow f_{box}(I_{s_0}, 1)$ 
4:    $mean_{1,I_{s_0}^2} \leftarrow f_{box}(I_{s_0} \odot I_{s_0}, 1)$ 
5:    $mean_{I_{s_0}} \leftarrow f_{box}(I_{s_0}, r)$ 
6:    $mean_{I_{s_0}^2} \leftarrow f_{box}(I_{s_0} \odot I_{s_0}, r)$ 
7:    $var_{1,I_{s_0}} \leftarrow mean_{1,I_{s_0}^2} - mean_{1,I_{s_0}} \odot mean_{1,I_{s_0}}$ 
8:    $var_{I_{s_0}} \leftarrow mean_{I_{s_0}^2} - mean_{I_{s_0}} \odot mean_{I_{s_0}}$ 
9:    $\chi_{s_0} \leftarrow \sigma_{1,I_{s_0}} \odot \sigma_{I_{s_0}}$ 
10:   $mean_\chi = sum(\chi_{s_0})/N_{s_0}$ 
11:   $min_\chi = minimum(\chi_{s_0})$ 
12:   $\bar{\chi}_{i,s_0} = (sum(1/(\chi_{s_0} + \varepsilon))/N_{s_0})^{-1}$ 
13:  return  $mean_\chi, min_\chi$  and  $\bar{\chi}_{i,s_0}$ 

//  $mean_\chi$ : equivalent to  $\mu_\chi$  earlier in (2.11)
//  $\odot$ : element-wise multiplication
//  $\sigma$ : standard deviation = the square root of the corresponding  $var$ 
//  $N_{s_0}$ : total number of pixels for the down sampled frame =  $240 \times 135$ , when  $s_0 = 8$ 
```

3.3.2 Linear Coefficients Procedure

The *LinCoef* procedure in Algorithm 3 represents the linear coefficient computing stage, which takes the guidance I , the input image p , the global parameters that are computed in the preprocessing stage, a radius r and the constants as input and computes the averaged coefficients $\bar{a}$ and $\bar{b}$ in the linear model (2.16).

Down-sampling process is also applied in this stage but with a lower sampling rate s_1 . The variance

---

Algorithm 3 Fast gradient guided filtering - Linear coefficients

---

Input: $I, p, s_1, r, mean_\chi, min_\chi, \tilde{\chi}_{i,s_0}, \lambda, \varepsilon$
Output: Avaraged linear coefficients $\bar{a}$ and $\bar{b}$

```

1: procedure LINCDEF( $I, p, s_1, r, mean_\chi, min_\chi, \tilde{\chi}_{i,s_0}, \lambda$ )
2:    $I_{s_1}$  and  $p_{s_1} \leftarrow$  down-sampling  $I$  and  $p$  by  $s_1^2, r = r/s_1$ 
3:    $mean_{1,I_{s_1}} \leftarrow f_{box}(I_{s_1}, 1)$ 
4:    $mean_{1,I_{s_1}^2} \leftarrow f_{box}(I_{s_1} \odot I_{s_1}, 1)$ 
5:    $mean_{I_{s_1}} \leftarrow f_{box}(I_{s_1}, r)$ 
6:    $mean_{I_{s_1}^2} \leftarrow f_{box}(I_{s_1} \odot I_{s_1}, r)$ 
7:    $mean_{I_{s_1} p_{s_1}} \leftarrow f_{box}(I_{s_1} \odot p_{s_1}, r)$ 
8:    $mean_{p_{s_1}} \leftarrow f_{box}(p_{s_1}, r)$ 
9:    $var_{1,I_{s_1}} \leftarrow mean_{1,I_{s_1}^2} - mean_{1,I_{s_1}} \odot mean_{1,I_{s_1}}$ 
10:   $var_{I_{s_1}} \leftarrow mean_{I_{s_1}^2} - mean_{I_{s_1}} \odot mean_{I_{s_1}}$ 
11:   $cov_{I_{s_1} p_{s_1}} \leftarrow mean_{I_{s_1} p_{s_1}} - mean_{I_{s_1}} \odot mean_{p_{s_1}}$ 
12:   $\chi_{s_1} \leftarrow \sigma_{1,I_{s_1}} \cdot \sigma_{I_{s_1}}$ 
13:   $\Gamma \leftarrow (\chi_{s_1} + \varepsilon) / \tilde{\chi}_{i,s_0}$ 
14:   $\gamma \leftarrow \exp^{\frac{4(\chi_{s_1} - mean_\chi)}{mean_\chi - min_\chi}}$ 
15:   $a \leftarrow (cov_{I_{s_1} p_{s_1}} + \frac{\lambda}{\Gamma} \gamma) / (var_{I_{s_1}} + \frac{\lambda}{\Gamma})$ 
16:   $b \leftarrow mean_{p_{s_1}} - a \cdot mean_{I_{s_1}}$ 
17:   $\bar{a} \leftarrow f_{box}(a, r)$ 
18:   $\bar{b} \leftarrow f_{box}(b, r)$ 
19:  return  $\bar{a}$  and  $\bar{b}$ 

```

---

and covariance are computed within the sliding window with radius of r and 1, before deriving the corresponding χ values. The Γ and γ values are then computed corresponding to s_1 down-sampled pixels. The coefficients a_k and b_k can be derived according to (2.13) and (2.14) and then averaged among other neighboring values within its filtering window. Note that similar box filtering processes

but with different radius are required at this stage. The box filtering operation is efficiently implemented in the proposed hardware design as detailed in Section 3.4.

3.3.3 Restoring Procedure

The parallel restoring stage is represented by the *Restore* procedure in Algorithm 4. This stage is simple. It just takes the guidance image I and the linear coefficient means and up-samples the coefficient means at the rate of s_1 and outputs the filtering result q for the full size image.

Algorithm 4 Fast gradient guided filtering - Restore

Input: $I, \bar{a}, \bar{b}$ and s_1

Output: Guided filtering result q

1: **procedure** RESTORE($I, \bar{a}, \bar{b}, s_1$)

2:    $\bar{a}$ and $\bar{b} \leftarrow$ up-sampling $\bar{a}$ and $\bar{b}$ by s_1^2

3:    $q \leftarrow \bar{a} \cdot I + \bar{b}$

4:   **return** q

// Up-sampling restores original frame size

To restore the image to its full size, the derived linear coefficient means $\bar{a}_k$ and $\bar{b}_k$ are first up-sampled, i.e., each coefficient is repeatedly used for $s_1 \times s_1$ times. Then the coefficient means $\bar{a}_i$ and $\bar{b}_i$ are combined with the guided image I , according to the local linear model (2.1), to produce the final filtering results.

3.3.4 Sampling and Restoration

In general, when down-sampling is applied, there are trades off between the computing resource saving and possible loss of information in the image frame.

For fast computation of the global parameters in the preprocessing stage and at the same time adequate filtering quality, it is desired to have a large sampling rate s_0 while still keeping sufficient pixel

information in a full-HD frame. For a full-HD frame of size 1920×1080 or larger, the MATLAB simulation shows that down-sampling at $s_0 = 8$, i.e., 1 in 64 pixels, can still provide accurate global information without causing noticeable visual degradation. Hence, the lengthy 1920×1080 pixel-wise calculation of the standard deviation and χ_k (2.10) is reduced by 64 times to a computation involving just 240×135 pixels in the preprocessing stage, which is described in details by Algorithm 2.

To maintain the filtering quality, a lower rate $s_1 = 4$ is used since more pixels within the original filtering window can be taken into account for the linear model of a guided image filter. Consequently only one pixel in a 16-pixel square is used for the computation of the linear coefficients, reducing the computation complexity of this stage from the original $O(N)$ to $O(N/s_1^2)$ [36], where N is the total number of pixels in a frame, and at the same time maintaining the resultant image with no visible degradation.

As it is required to restore the image back to the original full-HD size, parallel processing is used to up sample the image at the final output stage. For every pair of $\bar{a}_i$ and $\bar{b}_i$, 16 pixels of the original image are processed in this parallel restoring stage, which is operated at a frequency $f_a = s_1^2 f_o = 16f_o$, i.e., 16 times faster than the operating frequency f_o for the linear coefficient computing stage. The adoption of different operating frequencies in the stages gives an additional advantage of the proposed design. The linear coefficient computing stage has many highly complex computations such as exponentiation, square root and division, while the restoring output stage has only simple computations (addition and multiplication). Hence, the long critical path delay in the linear coefficient computing stage will not jeopardize the throughput, since the overall throughput can be increased by the parallel processing in the output stage.

3.4 Proposed Hardware Architecture

With the proposed dataflow and processing algorithms in Section 3.3, simulation is first conducted for thorough analysis on the computation workloads distribution among different steps in the algorithm. It

is found that the extraction of global information from down-sampled pixels in the preprocessing stage consumes a small portion of the total computation time. Together with the output stage that contains only fast linear operations that can be processed in parallel and with a higher operation frequency, high throughput is achieved.

Therefore, the VLSI architecture for the fast gradient guided image filtering is designed as shown in Fig. 3.2. There are three main stages, namely the preprocessing stage, the linear coefficients computing stage and the parallel restoring stage, as circumscribed by the dash lines. The three stages correspond to Algorithms 2, 3 and 4 respectively.

A frequency divider is responsible for deriving the slower operating frequency f_o from the faster accessing frequency f_a . The preprocessing stage and the linear coefficients computing stage run at f_o , while the parallel restoring stage operates at f_a .

An IO controller for coordinating the pixel inputs is included. It manages the status of the filter system with a finite state machine (FSM). The FSM switches in the four states of Idle, High sampling, Low sampling and Parallel restoring. In different states, the pixel values from the guided image I and the filtering input p are fed to proper modules at corresponding frequency arbitrated by the IO controller. Buffers for data transfer across stages and box filtering are provided.

With the help of the frequency divider and the IO controller, the filter system derives global parameters at the preprocessing stage, then uses the parameters to determine the linear coefficients using a sliding window approach with a stripe-based method, and flushes the filtering output back to original size in the parallel output stage. The details of the operations in the three stages are described in the following sub-sections.

3.4.1 Preprocessing Stage

The preprocessing stage takes in the down-sampled guided image I and computes the three global parameters before the main processing starts. As indicated in Step ① in Fig. 3.2, the down-sampled pixels are collected as a new frame for the computation of χ_{k,s_0} (2.10) as intermediate results[3]. Each box

Figure 3.2. Hardware architecture of the proposed guided image filter.

filter (BF) here with an input τ and a corresponding radius of r as indicated in Fig. 3.2 computes the $\mu_k(\tau)$ term in (2.5), including summing the pixel values within a window of radius r and deriving the mean. The multiplier-accumulators (MACs) compute the variance or covariance needed, and the square root (SQRT) modules then compute the standard deviations (σ s) required later. Subsequently, the three global parameters $\bar{\chi}_{i,s_0} = (\frac{1}{N} \sum_{i=1}^N \frac{1}{\chi_i + \epsilon})^{-1}$, μ_χ and $\min(\chi)$ are derived in the Harmonic Mean Kernel, Mean and Comparator modules respectively. Note that the parameter $\bar{\chi}_{i,s_0}$, is derived in this form here due to the refactorization of a_k to (3.3) as explained later in Section 3.4.2.

As can be seen from Algorithms 2 and 3, the operation of averaging is required many times at this stage and the next stage. It can also be observed that, besides the division by the size of the corresponding square window, these averaging operations (except the one for deriving μ_χ) all take place in a square window of different radius (defined as the distance in pixels from the center to the edge of the window). Computing the sum of all the pixel values within a window by simply adding their values for each scanning window results in repetitive and unnecessary calculations, as majority of the pixels remains the same when the scan window moves from one pixel to the next.

Thus, a parameterized module is designed for integrating the pixels in a window of any radius. The design is based on the integral image engine (IIEs) presented in [4], which evolves from the joint bilateral filter introduced in [5] and is highly efficient in both computation speed and memory usage.

The integrating process within the box filters is illustrated in Fig. 3.3, where the orange dashed scan window ω with its bottom right pixel at S is moving from top to bottom and left to right. When the scan window moves one pixel to the right, the integral (summation) of all the pixels, denoted as II_O^S , within the rectangle bounded by its top left pixel O and bottom right pixel S , is computed and stored. II_O^S can be computed by adding the right column of pixels (green shaded area) to the previously stored integral at D , II_O^D . The total pixel values of the additional column is $I_S + II_{O'}^{S'} - II_O^{D'} - I_Q$, where I_S and I_Q are the pixel values at S and Q respectively. Consequently, II_O^S can be computed as (3.1), involving only two additions and two subtractions. Thus, the integral of the scan window ω can be computed with just a simple subtraction as in (3.2).

Figure 3.3: Mechanism of box filter integrating.

$$II_O^S = II_O^D + I_S + II_{O'}^{S'} - II_{O'}^{D'} - I_Q \quad (3.1)$$

$$\sum_{\omega} I = II_O^S - II_O^A \quad (3.2)$$

With the box filter, the summation of all pixel values within a scanning window is simplified into just five additions and 76 (equal to the stripe width including the extended null stripe) registers for storing the integral results for one row within a stripe. With the input pixel values I_S and I_Q fetched by the IO controller, the integral result $\sum_{\omega} I$ is produced and buffered according to (3.2) and (3.1), by the box filter in Fig. 3.4, where the registers are shifted to the left one position each cycle.

3.4.2 Linear Coefficients Computing Stage

The linear coefficients a_k (2.13) and b_k (2.14) are computed in Step ② after the preprocessing as indicated in Fig. 3.2. At this stage, the values of $\sigma_k^2(I)$, $cov(Ip)$, $\mu_k(I)$ and $\mu_k(p)$ are first computed with the box filters and the MAC modules. χ_{k,s_1} (2.10) is derived in a similar way as in the preprocessing stage as can be seen from Fig. 3.2. Note that $\Gamma_k(I)$ (2.9) used to compute a_k (2.13) is not directly generated due to the refactorization of a_k to (3.3) as explained later in this section. Subsequently, γ_k (2.11)

Figure 3.4: Architecture of the box filters.

is computed by the γ_k kernel with the standard Synopsys DesignWare [105] modules for division and exponentiation to derive the results needed. a_k (2.13) and b_k (2.14) are then computed by the a_k and b_k kernels respectively, and their means $\bar{a}_i$ and $\bar{b}_i$ (2.16) are computed by a box filter with a radius of r_{s1} . The box filters are designed to minimize the on-chip memory by adopting the same design concept of the integral image engine proposed in [4] and [5] and described in details in Section 3.4.1.

To reduce the main hardware resources, specifically the buffering registers, the stripe-based method (SBM) as in [4], [5] is adopted here for the linear coefficients computation. In the SBM, a frame is divided into several vertical stripes and only one stripe is input and processed at a time. The stripe width for the proposed design is set to 60 pixels, with one extended null stripe (column) each of width 8 pixels added to the left and right sides, as illustrated in Fig. 3.3 in Section 3.4.1. Note that the stripe in the preprocessing stage has only half the width of the stripes in this stage due to different down-sampling rates $s_0 = 8$ and $s_1 = 4$. During the processing, the sliding window scans stripe-wise, resulting in only one stripe of pixels being processed.

The a_k kernel is another complex arithmetic module although the b_k kernel is relatively simpler. As can be observed from (2.13), three divisions are involved in the computation of a_k . In the proposed design, (2.13) is rearranged as (3.3) to reduce the three divisions to only one. The design provides three benefits. Specifically, the hardware resource required is reduced. The critical path delay in the a_k kernel

is shortened and at the same time accuracy loss due to truncations in the intermediate computations is minimized.

$$a_k = \frac{\chi_{k,s_1} [\mu_k(Ip) - \mu_k(I)\mu_k(p)] + \lambda \gamma_k \bar{\chi}_{i,s_0}}{\chi_{k,s_1} \sigma_{k,r}^2(I) + \lambda \bar{\chi}_{i,s_0}} \quad (3.3)$$

The term $\mu_k(Ip) - \mu_k(I)\mu_k(p)$ in (3.3) is the covariance of guidance image I and input image p , denoted as $cov(Ip)$ in Fig. 3.2. $\bar{\chi}_{i,s_0}$ is actually the harmonic mean of χ_i with a down-sampling ratio of s_0 . $\bar{\chi}_{i,s_0}$ is computed in the preprocessing stage as $\bar{\chi}_{i,s_0} = (\frac{1}{N} \sum_{i=1}^N \frac{1}{\chi_i + \varepsilon})^{-1}$ as shown in Algorithm 2.

The a_k kernel is designed according to (3.3) and shown in Fig. 3.5. The numerator and denominator of (3.3) are computed before the division. For simplicity, a small power-of-two value $\lambda = 1/32$ is chosen. The b_k kernel simply consists of a multiplier and an adder.

Figure 3.5: Hardware design for a_k and b_k kernel.

The word length of a_k and b_k is chosen based on the word length analysis in [4]. a_k has 1 integer bit and 8 fraction bits and b_k has 8 integer bits, which is the same as the image pixel values in both I and p . Since 8 bits are used to represent the pixel values, the values of $cov(Ip)$, $\sigma_{k,r}^2(I)$ and χ all have a word length of 16. Thus, both the numerator and denominator of a_k in (3.3) have 32-bit word length. A divider from Synopsys DesignWare is used here with intermediate quotient truncated to 9 bits as required for a_k in the proposed design.

3.4.3 Parallel Restoring Stage

When the input image I is fed to Stage 2, $s_1^2 = 16$ pixels are also fed into the output kernel and buffered. Once a pair of the values $\bar{a}_i$ and $\bar{b}_i$ are ready, they are synchronized with the buffered pixels in the shift registers as shown in Fig. 3.2 and the output kernel can simultaneously compute all the filtering output q_i (2.15) for s_1^2 pixels. This stage operates at a frequency f_a that is s_1^2 times faster than f_o of the other two stages. In this way, every s_1^2 filtering output pixels q_i are simultaneously produced and buffered before being shifted out serially. With the parallel processing in the output stage, the linear coefficients computing stage may have long delay ($1/f_o$) at each pipelined stage, but the overall throughput of the system is dictated by the short delay ($1/f_a$) of the simple and parallel output kernel.

3.5 Performance Evaluations and Discussions

3.5.1 Filtering Analysis

The gradient guided image filter implemented in this work has an advantage in filtering quality over the original guided image filter [27]. The original guided image filter suffers from halo-artifacts and the image cannot be represented well at some edges [1]. The phenomenon is most apparent in the applications for image detail enhancement [3], as can be seen in Fig. 3.6 where the instances of detail enhancement are shown. In the resultant images in column (b), some obvious halo-artifacts appear along the edges as indicated by the arrows. The halo-artifacts are resulted due to the constant parameter λ in (2.3), forcing the guided filter to smooth at those edges.

The proposed architecture takes into account the edge-aware weighting $\Gamma_k(I)$ (2.9) and an explicit first-order edge-aware constraint γ_k (2.10). As a result, the edges are better preserved and smoothing is accomplished on only the flat areas. It is evident from Fig. 3.6 (c), obtained from the MATLAB simulation with the exact down-sampling ratios and data word length as in the proposed architecture, that the unnatural bright edges or halo-artifacts are significantly reduced or avoided.

Figure 3.6: Comparison of the original guided image filter and proposed fast gradient guided filter in the application of detail enhancement simulation. (a) Input source image $I(p)$. (b) Detail enhanced result by original guided image filter. (c) Detail enhanced result by fast gradient guided filter.

In addition to the visual observations above, quantitative analysis is conducted on the intensity for the filtering results in detail enhancement application. Fig. 3.7 shows the analysis results, where the three images from top to bottom are the original guided image, the resultant image by the original guided image filter and the resultant image by the proposed architecture respectively. The intensity for the filtering results in detail enhancement application are compared. The three graphs compare the respective green channel intensities, which are extracted from the simulation, with a horizontal line of

Figure 3.7: The one-channel (green) intensity analysis on detail enhancement.

pixels in the middle of the illustrated area sampled from each of the images. The halo edges can be obviously observed in the image obtained with the original guided filter and the intensity graphs further verify that the unwanted pulses at edging areas (circled) are significantly reduced by the edge-aware fast gradient guided filter.

3.5.2 Performance Evaluation on Hardware Design

Despite its wide range of applications, the implementation of the guided image filtering algorithm in ASIC is limited. As can be seen from the literature review in Chapter 2, the only existing ASIC design of the guided image filter is reported in [4]. An FPGA based stereo matching based on the guided image filter is reported in [34], where similar structure to [4] is used. Note that the filters implemented in both [4] and [34] are the guided image filter, not the gradient guided image filter. Thus halo-artifacts are not considered and no computation on global parameters is involved.

Table 3.1: Comparisons with existing guided image filter design

	Kao's design [4]	Proposed design		
	Layout	Synthesis	Synthesis	Layout
Technology	TSMC-90nm	STM-90nm	STM-65nm	STM-65nm
Frame Size	1920×1080	1920×1080	1920×1080	1920×1080
Filter Window	31×31	33×33	33×33	33×33
Stripe Width	120/180	60/76	60/76	60/76
Frequency f_o (MHz)	100	50	64.5	64.5
Design Area (μm^2)	818800	750913	394207	591042
Power Consumption (mW)	22.52	11.64	13.61	17.54
Energy Consumption / frame (mJ)	0.75	0.19	0.18	0.23
On-chip Memory (Byte)	3206	2261	2261	2261
Frame Rate (fps)	30	60+	75+	75+

The proposed VLSI design of the fast gradient guided filter is synthesized and implemented in the STM 90nm and 65nm CMOS process using the Synopsys Design Compiler and IC Compiler. The results are summarized in Table 3.1. The layout in 90nm technology is not obtained due to lack of cell layout library. As there is no existing ASIC implementation of the fast gradient guided filter, the closest ASIC design on the guided image filter in [4] is included in the table to provide some performance indications. MATLAB simulation in this work on a PC with a 4-core CPU shows that a complete run

costs 1.6841 seconds for the original GIF and 3.4000 seconds for the GDGIF to process a full-HD frame (1920×1080 pixels). The Python version provides a better speed of 0.6078 seconds for the original GIF and 1.1952 for the GDGIF. It is evident that the processing speed is far behind the real-time performance requirement of 30 to 60 frames per second.

The proposed design is able to process a Full-HD stream of input images at real time of more than 60 frames per second. This increase of frame rate is the result of the down-sampling process adopted and the parallel output stage. The filtering window size is set to 33×33 , so that the radius of the square window 16 can be easily down-sampled at the ratio of both $s_0 = 8$ and $s_1 = 4$. The down-sampling process has impact on the stripe width for hardware processing, reducing the computation resources for each stripe. Since complex arithmetic modules such as exponentiation, division and square root are required in the design, which are implemented with Synopsys DesignWare modules, the proposed filter design has longer delay of $20ns$ for synthesis in $90nm$ technology than the design of [4] in similar CMOS $90nm$ technology. As a result, the operating frequency of the linear coefficients computing stage is at only $f_o = 50MHz$. The lower frequency, however, has benefited the power consumption. Together with the lower memory requirement and fewer memory accesses, the proposed architecture consumes less than half the power despite it has higher computation complexity.

As can be seen from the literature review in Section 2.4, the design of [4] is an implementation of the original GIF, which is different from the proposed design for the more advanced GDGIF. The proposed design provides more functionality and involves much higher computation intensity. However, since there is no existing implementation for the GDGIF, the result from [4] provides valuable good indicative comparison in many aspect of parameter specification and performance measures.

It can be seen that the proposed design for FGGF achieves better filtering quality for edge-preserving processing, with comparable or lower cost on the hardware resources and power consumption. However, the proposed design can only operate at a frequency that is a half of that in [4]. Further improvement on the processing speed is possible. The increased complexity of arithmetic operations actually neutralizes part of the advantages brought in by the down-sampling and parallel restoring approach. According

to the analysis on the critical delay paths, it is found that the delays are dominated by the operations of division and exponentiation, which possess delays of $18.14ns$ and $17.38ns$ respectively. Thus, if the delays of these two operations can be reduced, there can be significant improvement on the overall throughput. Several methods are proposed to simplify the designs of the divider and the exponentiation module by approximating the computation with negligible loss of accuracy. As such, the delays are reduced. The operating speed is increased. Consequently, higher overall throughput is achieved. The details of the proposed methods, the designs and their performance evaluations are presented in the following chapters.

3.6 Summary

The design of the proposed fast gradient guided image filter is described in this chapter. The design challenge on the computation involving global information is addressed by introducing the preprocessing stage. Down-sampling technique is adopted to reduce the redundant cost in both preprocessing stage and the main processing stage for the computation of the linear coefficients. Parallel restoring mechanism is employed at the output stage to provide high throughput for the system to satisfy the requirement of real-time applications. The structure of the filter hardware architecture together with details of important modules at different stages are described. The filtering quality is analyzed in the context of image detail enhancement. The performance of the design is also presented and compared to closest counterpart. The implementation results show that the throughput reaches more than 60 frames per second, which is two and a half times higher than the existing ASIC design for guided image filtering[4]. However, the performance of the design can be further improved if the speed bottlenecks in the divider and the exponential module can be removed. Chapter 4 addresses the design of dividers and the performance improvement is further discussed in Chapter 5.

Chapter 4

Non-iterative Dividers Design

4.1 Overview

As discussed in Chapter 3, the division operation is one of the performance bottlenecks of the proposed FGGF hardware architecture, in terms of delay in a pipelined structure. Comparing to the design in [4], the proposed design in Chapter 3 can only work at half of the operating frequency, which limits the overall throughput of the system. According to the analysis on the critical delay paths, the operation of division is one of the speed limit modules, which possesses delay of $18.14ns$. The delay bottleneck caused by the division has to be removed before the performance of the FGGF design can be improved.

This chapter focuses on the discussions of the designs of fast divider circuits. It is noted that a wide range of applications such as image/video processing, like the guided image filtering, involve arithmetic division and demand high performance, but at the same time can tolerate some degrees of error or allow small deviations from their theoretical results. This chapter presents three approaches for non-iterative arithmetic division circuit design, leveraging computation accuracy for high performance.

In this chapter, the division process in floating-point representation is formulated by a specific yet typical format as described in Section 4.2. The mantissa of the quotient is then separately modeled and normalized into a 3D unit box as a curved surface for further analysis. Based upon this formulation

and modeling, the target curved surface representing the quotient is partitioned into smaller regions and then approximated with simple and linear operations as described in Section 4.3. Three approaches are proposed and applied here for better computation accuracy and less hardware cost, including the curve fitting method, the mixed integer linear programming and stripe-based approximation. The proposed hardware architecture for the non-iterative divider design is described in details in Section 4.4. The performance of proposed approximation approaches and hardware architecture are evaluated and compared with other existing designs in Section 4.5.

As can be seen from the literature review on the existing division implementations as arithmetic modules, most of those are based on either the digit recurrence method or the iterative convergence algorithm. There are only a few non-iterative designs evolving from logarithmic circuits design. The main contributions of this work include the followings:

- **The floating point division is formulated and modeled into a curved surface confined by a unit 3D box that is easy for analysis.** This procedure ensures the consistent format of inputs and output of a division process, and views the quotient mantissa as a function of two variables of the same range, addressing the concerns of high dynamic range.
- **The approximation of the curved surface is addressed by curve fitting method and optimized by the mixed integer linear programming and striped based approximation.** A complete procedure for analyzing the approximation problem is developed. The procedure addresses the current performance bottleneck of the division operation.
- **A non-iterative hardware architecture is proposed.** According to the approximation approaches, the corresponding VLSI architecture is achieved with only simple operations and parameters in a small look-up table.
- **Proposed design space can serve various applications.** The proposed approaches and the corresponding architecture allow the trade-off of computation speed and accuracy, enabling the use of the divider circuits in various applications with different speed and accuracy requirements.

The novelties of the work presented in this chapter are summarized below.

- **Division operation is approximated by simple linear operations.** The quotient of division is modeled as a curved surface and partitioned into regular regions. The curved surface in each region is approximated by a plane that best fits the curved surface.
- **Mixed integer linear programming is adopted in the optimization.** After the division process is modeled into a curved surface in a unit box, the curve fitting method provides a natural solution to find the approximation plane. However, due to wordlength limits, the problem is actually defined in a discrete space, which is more suitable to be solved by the MILP approach. Accuracy is improved with the adopting of the MILP optimization.
- **A stripe-based approach is proposed to reduce the non-linearity.** Early effort in the work tries to approximate the quotient with uniform partition the two operands. Closer observation shows that the non-linearity comes from only one of them, which makes the stripe-based approximation more appropriate. The adopting of the stripe-based method reduces the hardware complexity of the proposed approximation divider.

Extensive simulations were conducted to evaluate different optimization methods. Thorough analysis on the computation accuracy and the approximation error distribution were performed. Trade-offs on accuracy, speed and hardware cost by varying the operand wordlengths are also evaluated with the designs synthesized by Synopsys Design Compiler in STM 90nm CMOS technology. The results obtained show that the proposed divider designs can be applied in a range of application scenarios that have different tolerances for computation error. One of the proposed divider circuit is selected and used in the FGGF system presented in Chapter 3. The selection criteria and the performance improvement of the modified FGGF system are discussed in Chapter 5.

4.2 Modeling and Formulation for Division

The standard floating-point number representation [96] is useful for solving one of the main problems when implementing the division operation, that is the high dynamic range of the quotient. In the representation, a number is partitioned into a fraction and an exponent part, in addition to a sign bit. While it is a standard and commonly used in arithmetic modules including division, the 32-bit long wordlengths (23 bits for fraction) of the single-precision format may well exceed the need for many applications where the relative errors could be significantly larger than the least significant bit (LSB) of fraction part. This results in redundant bits in the representation and incur unnecessary computations and hardware costs.

To reduce the hardware resources required, a shorter representation is introduced and adopted in the proposed division arithmetic modules, yet providing sufficient accuracy for the quotient at the same time. In this representation, there are three parts, namely, the sign s , the exponent e , and the mantissa (also known as the fraction) m . An implicit 1 is in front of the radix point and mantissa, which keeps operands in the range of $[1, 2)$. For illustration and comparison purposes, the proposed divider has 22 bits for both the dividend and the divisor, with 1 sign bit, 5 bits of exponent and 16 bits of mantissa, as shown in Fig. 4.1. Note that e has a -15 bias to balance its negative and positive range and the value X is as given in Fig. 4.1.

$$X = s \cdot (1 + m) \times 2^{e-15}$$

Figure 4.1: The adapted number representation.

According to the number representation described above, the division operation can be carried out

simultaneously for the three parts s , e and m . The sign bit operation is just a trivial exclusively-or operation and the exponent part is basically a subtraction with consideration of the bias. The main computation is on deriving the non-linear mantissa part.

For large systems that involve cascading arithmetic modules, it is helpful to have inputs and outputs of these arithmetic modules represented in identical format, so that no conversion or parsing is needed between the arithmetic modules. This is also a requirement for the proposed division implementation as well.

The quotient Q of the division $X_a \div X_b$ can be represented as (4.1), where s_a , s_b , e_a , e_b , m_a and m_b are the signs, exponents and mantissas of X_a and X_b respectively. According to (4.2), since $1 + m_a$ and $1 + m_b$ are in the range $[1, 2)$, the resultant mantissa M (4.2) is on a smooth curved surface in the range of $(0.5, 2)$ as illustrated in Fig. 4.2(a).

$$\begin{aligned}
 Q &= \frac{X_a}{X_b} \\
 &= \frac{s_a \cdot (1 + m_a) \times 2^{e_a-15}}{s_b \cdot (1 + m_b) \times 2^{e_b-15}} \\
 &= s_Q \cdot M \times 2^{e_a-e_b}
 \end{aligned} \tag{4.1}$$

where

$$M = \frac{1 + m_a}{1 + m_b} \tag{4.2}$$

To confine M in the same representation as X_a and X_b as mentioned earlier, a normalization process can be performed as in (4.3) and (4.4) whenever $m_a < m_b$, resulting in the folded surfaces shown in Fig. 4.2(b).

$$\begin{aligned}
 Q &= \frac{X_a}{X_b} \\
 &= \frac{s_a \cdot (1 + m_a) \times 2^{e_a-15}}{s_b \cdot (1 + m_b) \times 2^{e_b-15}} \\
 &= s_Q \cdot M \times 2^{e_a-e_b-e_n}
 \end{aligned} \tag{4.3}$$

where

$$M = \begin{cases} 2 \times \frac{1+m_a}{1+m_b}, & m_a < m_b \\ \frac{1+m_a}{1+m_b}, & \text{otherwise} \end{cases} \quad (4.4)$$

and

$$e_n = \begin{cases} 1, & m_a < m_b \\ 0, & \text{otherwise} \end{cases} \quad (4.5)$$

With the normalized quotient, it can be seen that the mantissa of the quotient is equivalent to the curved surface bounded in the 3D unit box in Fig. 4.2(b). Hence the division operation can be conducted by locating the mapping point on the curved surface, corresponding to the two variables' tuple (m_a, m_b) from both axis.

Figure 4.2: The fraction division curved surfaces. (a) Before folding; (b) After folding.

Locating a mapping point on such a curved surface is a non-linear operation that can be complex and difficult to implement by simple hardware processing elements. If the whole curved surface in Fig. 4.2(a) is a flat plane, only linear operation and three parameters will be involved, making the implementation simple and straightforward. This simplified assumption however gives insights for the possibility for linear approximation.

4.3 Approximation Approaches

The assumption of a single flat plane to represent or approximate the normalized division mantissa is over simplified, because it ignores the differences brought by curvature between the estimated plane and the actual surface. The larger the estimated plane is, the larger the approximation error will be.

With this observation, the approximation error can be reduced by making the estimated plane smaller and by using many smaller planes to represent the curved surface of quotient. This approach leads to the partitioning of the original curved surface and each partitioned curved surface being approximated by a small flat plane. When the number of planes are large enough, or in order words, the planes are smaller enough, the approximation errors will reach an acceptable level. However, the more planes used to approximate the curved surface, the more complex will be the divider circuit. The following sections discusses three methods to optimize the approximation. The number of planes used for approximation, the methods to obtain the planes to minimize the approximation errors and the trade-off of accuracy and hardware cost are addressed.

4.3.1 Curve Fitting Method

The first natural approach is to partition the folded curved surface in Fig. 4.2(b) evenly along both the axis of m_a and m_b . In this way, the two folded curved surfaces are evenly divided as shown in Fig. 4.3(a) into certain numbers of smaller square regions. The small curved surface in each region is then approximated by a small flat plane represented by (4.6) and curve fitting method is used to find the best plane to represent the curved surface with smallest approximation error.

$$\hat{M} = 1 + p \times m_a - q \times m_b + r \quad (4.6)$$

where $\hat{M}$ represents the approximated result of folded M in (4.4). p , q and r are the constant coefficients within each partitioned region to form the corresponding plane. The minus sign before q comes from the fact that all those good enough planes have a negative coefficient of q . Thus the minus sign will

Figure 4.3: (a) Normalized division curved surfaces evenly partitioned. (b) Percentage error distribution when approximated by planes using the curve fitting method.

ensure all those three coefficients to be positive values for easy storage in the memory. 1 is explicitly separated from the constant r to satisfy the representation format adopted because there is always an implicit 1 as shown in Fig. 4.1.

It is worth mentioning that the quotient surfaces in Fig. 4.3(a) are disjoint along the diagonal where $m_a = m_b$. Large approximation error may be resulted in these regions if square planes are used to approximate them as in other regions, because the approximated mantissa may fall on the totally wrong location and the exponent may also be wrongly computed due to wrong carry bit. As such, each square region along the diagonal is further divided into two triangular regions, each approximated by a triangular plane. The triangular planes are represented by the same equation (4.6). The hardware implementation of (4.6) is straightforward and efficient as only simple arithmetic modules such as shifters, adders/subtractors and look-up tables are required.

As the original curved surfaces are approximated by small square/triangular planes, there are approximation errors, leading to loss of accuracy. However, the design is acceptable as long as the approx-

imation errors are within the accuracy tolerance specified in an application. The approximation error in terms of the Maximum Absolute Error Percentage (MAEP), η , given in (4.7) can be derived from (4.1), (4.2), (4.4) and (4.6). It can be seen from (4.7) that η is independent of the exponent.

$$\begin{aligned}\eta &= \left| \frac{\hat{Q} - Q}{Q} \right| \\ &= \left| \frac{\hat{M} - M}{M} \right| \times 100\%\end{aligned}\tag{4.7}$$

where $\hat{Q}$ denotes the approximated quotient of Q .

The accuracy and the complexity of the design are impacted by several parameters, specifically, the number of planes used for the approximation, the values of the constant coefficients p , q and r , as well as their wordlengths. It is possible to obtain designs that are more accurate but complex, or simpler but less accurate, providing a trade-off of accuracy and area-delay product (ADP). The parameters are obtained in the way described below.

The number of partitions is first decided and the curve fitting approach is adopted to find the coefficients p , q and r for the square/triangular plane to approximate the individual region of the curved surfaces and optimize the approximation errors. During the curve fitting process, uniform sampling points are selected from each partitioned curved surface. The function $[za, goodness] = fit([ma, mb], z0, 'poly11')$ from the MATLAB curve-fitting toolbox is then used to find a two-variable expression that best fits all the sampling points, and the approximation error for each region is then calculated. In the function, za and $goodness$ are respectively the returned fitting function, i.e., (4.6) in this case, and the fitting quality. ma and mb are the sets of sampling points and $z0$ is the set of values of the function corresponding to the sampling points. Type $'poly11'$ indicates that the fitted function type is a linear polynomial surface. It was experimented using different number of sampling points for the fitting and found that the approximation error varies negligibly when the sampling points are more than 100 points within each region. The coefficients p , q and r for each region are obtained after this curve fitting process.

In the experiments [100], two schemes of partitioning the curved surface, an 8×8 and a 16×16 partitions, each with different wordlength of the constants, were tried. Several divider designs with different accuracies and ADPs were obtained. The detailed results, including the wordlengths of the coefficients, the areas and delays are presented in Section 4.5. A typical error distribution using the curve fitting method with a partition of 16×16 is illustrated in Fig. 4.3(b), reporting a MAEP of 0.14%.

4.3.2 Mixed Integer Linear Programming

Now that the quotient curved surface is partitioned, and the partitioned curved surfaces are approximated with small flat planes so that the complex non-linear computation is converted into simple linear operations for efficient circuit implementation. However, the curve fitting method described in the last section is not necessarily the optimized fitting algorithm for each region. The curve fitting method minimizes the average error by minimizing the mean squares of the errors. For an approximated arithmetic module, the optimization goal pursued here should be the minimization of the maximum value of any possible computation error that might occur, rather than the average one. Thus, the curve fitting method in Section 4.3.1 is just a solution, but not an optimized one.

The mixed integer linear programming method (MILP) is more suitable in this case, as it can optimize the worst case error. The approximation error in this case is the distance between the flat planes mapping the square areas in the (m_a, m_b) space, and the original curved surface for the quotient. It is aimed to optimize the maximum distance within each square region to obtain a flat plane that best fits the curved surface with smallest error. Therefore, the optimization objective is defined as in (4.8).

$$\min : V = \max(|M_i - \mathbf{C} \cdot \mathbf{X}_i|) \quad (4.8)$$

where $\mathbf{C} = [p, q, r]^T$ is the vector containing the linear parameters that represent the geometric characteristics of the flat plane, and $\mathbf{X}_i = [m_a, m_b, 1]^T$ is the vector containing the sampling points on the curved surface. V is the maximum variation of approximation from the original data.

If multiple sampling points are taken from the curved surface, and a linear approximation (degree of 1) is applied to minimize the maximum of variation, a unique and best approximation exists according to the Chebyshev criterion [106]. It is obvious that the objective in (4.8) is consistent with minimizing the approximation error defined as MAEP in (4.7).

In circuit implementation, the wordlengths of the parameters and the quotient are to be constrained. Thus the optimization process is performed in discrete space rather than in continuous space. To derive the corresponding linear parameters, the MATLAB function *intlinprog()* is used, with vectors $\mathbf{C}$, $\mathbf{X}_i$ and M_i as main input arguments. According to the simulation, the best approximation result can reach 0.08% in terms of MAEP, with the same hardware architecture, hence cost, as the one obtained from the curve fitting method. Fig. 4.4 shows the error distribution of approximation using the MILP approach when the curved quotient surface is partitioned into 16×16 regions.

4.3.3 Stripe-based approximation

Certain patterns can be observed from the error distribution of the approximation using the MILP approach, as illustrated in Fig. 4.4. It is obvious that similar distribution repeats at each of the partitioned square regions, except along the diagonal line when $m_a \approx m_b$. Besides the repetition, a closer viewing from the two axes provides an insight into the patterns, which are shown in Fig. 4.5. It can be seen from Fig. 4.5(a) that the approximation error varies significantly along the axis m_b , comparing to those in Fig. 4.5(b).

In addition to the error distribution patterns, Fig. 4.3(a) also indicates that the cutting edges for partitions along the axis m_a are straight lines, while those along the axis m_b are curved lines. This closer view on the quotient surface in Fig. 4.3(a) and Fig. 4.2(a) provides an insight that the non-linearity of (4.2) is entirely caused by the denominator (or m_b), and not by both m_a and m_b .

Furthermore, it is noticed from Fig. 4.2(a) that for a small region of m_b , the non-linearity of M is quite insignificant, i.e., the curved surface for a small region of m_b in Fig. 4.2(a) and Fig. 4.3 can be viewed as a stripe of rectangular plane. The above observations imply that if there is no division

Figure 4.4: The error distribution of approximation using the MILP approach.

along the m_a axis, the approximation errors will not be significantly worsen when the curved stripes partitioned along the m_b axis are approximated by planes. At the same time, fewer planes are needed for approximating the entire curved quotient surface, reducing the number of coefficients required, and hence the circuit size.

Fig. 4.6 shows the error distribution for the 32-partition approximation. It can be seen that the error distributions are similar among the stripes and the worse errors are at the four corners in each stripe, resulted after the optimization by using the MILP. Note that for visual clarity, the orientations of the axes m_a and m_b in Figs. 4.2 and 4.6 are different.

With partition along just one direction, smaller storage is required in the circuit for the striped-based approximation approach. Trading-off between computation accuracy and hardware complexity as in the previous approaches, three partitions along m_b by 32, 64, and 128 are investigated, together

Figure 4.5: Views from two axes on the error distribution of approximation using the MILP approach. (a) Viewing from m_a axis. (b) Viewing from m_b axis.

Figure 4.6: The error distribution for the 32-partition approximation.

with wordlengths of the coefficients p , q and r in (4.6). The detailed simulation and synthesis results are given in Section 4.5, where a best MAEP of 0.11% is reported with lower circuit area cost.

4.4 Hardware Architecture

Based upon the approximation approaches described in Section 4.3, the hardware architecture for the non-iterative divider is proposed and shown in Fig. 4.7, where X_a and X_b are the dividend and divisor respectively, Q is the quotient and there is a 1-bit overflow flag. The architecture consists of two main units, namely, the simple Sign and Exponent Module and the core Mantissa Module.

Note that the architecture shown in Fig. 4.7 reflects only the circuit for the stripe-based approximation approach with the MILP as the optimization method described in the previous section. The architecture for the curve fitting method and the MILP approach with uniformly partitions along both axes as described in the earlier sections is similar to this one. The main difference is that the Most Significant Bits (MSBs) of both input mantissas are required to address the coefficients of p , q and r stored in the look-up table.

4.4.1 Sign and Exponent Module with Overflow Flag

This module generates the sign bit $Q[21]$ and the 5-bit exponent $Q[20 : 16]$ for the quotient. The sign bit is simply realized by an exclusive-or operation, i.e., $Q[21] = X_a[21] \text{ XOR } X_b[21]$. The exponent is produced by a 5-bit subtractor, performing $Q[20 : 16] = e_a - e_b - 1 + 15 + \text{carry}$. Note that the exponent is not approximated and 15 is for balancing the positive and negative range. *carry* is determined by the carry bit of the approximated mantissa $\hat{M}[16]$. The module also generates an *overflow* flag to indicate whether the resulting exponent is out of the valid range of $[-15, 16]$.

4.4.2 Mantissa Module

The mantissa module in the dashed box takes as inputs the 16-bit mantissas of X_a and X_b and produces the 16-bit mantissa M of Q . It consists of a look-up table, two multipliers, an adder, a shifter and a multiplexer.

Figure 4.7: Proposed architecture for the non-iterative divider circuit.

The look-up table stores the coefficients p , q and r in (4.6), obtained from the MILP (or curve fitting) optimization. The MSBs of the divisor mantissa m_b are used (6 bits for the 64-partition design illustrated in Fig. 4.7) to address the look-up table. For the proposed stripe-based method, the size of the look-up table is $n \times (n_p + n_q + n_r)$, where n is the number of partitions, and n_p , n_q and n_r are the wordlength of p , q and r respectively. The size is $n^2 \times (n_p + n_q + n_r)$ for the 2-dimensional partition designs. Thus, a smaller look-up table is needed for the 1-dimensional stripe based partition, resulting in smaller circuit.

In the circuit, p and q are fed into the two multipliers to derive $p \times m_a$ and $q \times m_b$ in (4.6). The multiplications are implemented as shift-and-add operations with hardwired shifters and carry-save adder trees. The redundant bits beyond the 16th bit are truncated for hardware simplicity and faster speed without compromising the accuracy. The products $p \times m_a$ and $q \times m_b$ and the coefficient r are then summed by a small adder tree to derive the linear approximation $\hat{M}$ in (4.6).

The adder tree to derive $\hat{M}$ may produce a *carry* at the MSB, when $m_a > m_b$. The *carry* is added to the exponent in the sign and exponent module. On the other hand, when the *carry* remains 0, a folding action is taken as in (4.4). This is implemented by a shifter that shifts $\hat{M}$ left by 1 bit (multiplied by 2). The *carry* also controls the multiplexer to select the appropriate final mantissa M of the final quotient Q according to (4.4).

4.5 Performance Evaluation and Comparison

4.5.1 Performance Evaluation

In this section, the trade-offs between the computation accuracy and hardware complexity are investigated for the curve fitting method, the mixed integer linear programming approach and the stripe-based approach.

Several non-iterative divider circuits are obtained with the curve fitting method with different accuracy and ADP trade-offs as listed in Table 4.1, where the number of partitions, the wordlengths of the coefficients, the MAEPs achieved and the synthesis results are included. The synthesis results are

Table 4.1: MAEP and ADP trade-off of divider circuits with curve fitting method

Partition Planes	Wordlength (bits)		h	MAEP	Delay (ns)	Area (μm^2)	ADP
	p and q	r					
8×8 (s=3)	6	6	6	1.87%	2.44	4896.5	11974
	7	10	7	0.88%	2.50	5054.4	12636
	8	10	8	0.67%	2.59	5240.9	13574
	9	10	8	0.62%	2.65	5380.4	14258
16×16 (s=4)	8	12	8	0.56%	2.63	9846.5	25896
	9	12	9	0.25%	2.69	10596.2	28504
	10	12	10	0.16%	2.72	11183.4	30148
	11	13	10	0.14%	2.85	12324.9	35126

obtained from coding the proposed architecture of Fig. 4.7 in Verilog RTL for synthesis in 90nm CMOS process with the Synopsys Design Compiler. Note that 90nm CMOS technology is used for comparisons with the existing designs. The MAEP achieved by the proposed approach ranges from 1.87% to 0.14%. The circuit with the best MAEP is obtained by partitioning the curved surfaces of the quotient into 16×16 square planes.

It is worth mentioning that the MAEP in Table 4.1 is reduced when the number of partitions and the wordlength of p, q and r are increased, resulting in larger ADPs. On the other hand, increasing the wordlength of the input mantissas does not help in reducing the MAEP. The wordlength of the input exponents has no impact on the MAEP but marginally on only the area. With fewer partitions and shorter wordlength, lower hardware complexity and shorter delay due to smaller look-up table and simpler adder trees are achieved, but the accuracy is limited.

Based on the evenly partitioned curved surface and the MILP optimization approach, accuracies of 0.31% and 0.08% in terms of MAEP are obtained for the partition scheme of 8×8 and 16×16 respectively. The circuits delays and areas are the same as their counterpart designs obtained based on the curve fitting method as shown in Table 4.1, because the architecture and circuit designs are exactly the same. The differences are the values of the coefficients p , q and r , and hence, the accuracy.

For the implementation for the stripe-based approximation approach, the proposed non-iterative divider in Fig. 4.7 was synthesized and analyzed for 32, 64 and 128 partitions. For each of the designs obtained, the optimal wordlengths of the coefficients are determined by simulation. The designs are

coded in Verilog HDL and synthesized in STM 90nm CMOS process with Synopsys Design Compiler. The MAEP, delay and area of the designs, together with their ADPs are listed in Table 4.2.

Table 4.2: Divider circuits performance evaluation

Partitions	Wordlength (bits)		MAEP	Delay (ns)	Area (μm^2)	ADP
	p and q	r				
32	7	9	0.46%	2.43	5018.4	12195
	8	10	0.44%	2.48	5297.2	13137
	9	11	0.44%	2.52	5484.6	13821
64	8	11	0.35%	2.50	6248.8	15622
	9	12	0.26%	2.56	6429.9	16461
	10	12	0.22%	2.63	6697.4	17614
	11	13	0.22%	2.68	7014.3	18798
128	9	11	0.17%	2.69	8792.7	23652
	10	12	0.13%	2.72	8979.5	24424
	11	13	0.11%	2.79	9155.6	25544
	12	13	0.11%	2.85	9328.4	26586

The trade-offs of the accuracy and the ADP are also investigated as illustrated in Fig. 4.8, which shows the designs obtained with different partitions and their corresponding MAEPs and ADPs. It is evident that the finer the partition is, the better accuracy (smaller MAEP) will be. The MAEP achieved ranges from 0.46% to 0.11%. The designs with lower MAEPs require more partitions and longer coefficient wordlengths, which in turn demand larger look-up tables and computation modules, leading to larger ADP. In other words, with longer coefficient wordlengths, the accuracy is improved at the sacrifice of higher hardware complexity and longer computation delay.

Note that with a fixed number of partitions, the accuracy will reach a saturation point, beyond which the MAEP cannot be further reduced even with longer coefficient wordlengths as can be seen from Table 4.2. With the trade-offs, a design with high MAEP and small ADP is suitable for applications with low demand of accuracy and high demand of speed and size, while a design with lower MAEP suits applications with high accuracy requirement. The power consumption ranges from $522.4\mu\text{W}$ for the best 32-partition design to $875.6\mu\text{W}$ for the best 128-partition design.

It is also noted that the accuracies obtained using the stripe-based approximation are not necessarily better than its counterpart designs using the MILP method with 2D partitions. This degradation of

Figure 4.8: The MAEP and ADP trade-offs for various partitions and coefficient wordlengths.

accuracy is the result of the fact that, though each of the stripes partitioned is very narrow, its width Δ_{m_b} is not zero and the slope of the line or derivative $\partial M / \partial m_a$ is not the same at the position m_b and another position $m_b + \Delta_{m_b}$. This variation of slope along m_b makes the real situation not exactly the same as in the previous ideal assumption that the stripe is linear along the axis m_a . Thus, the feasibility of the stripe-based approximation for reducing the hardware resource cost and maintaining enough accuracy depends on the specific application requirements. Specifically, for the case on the proposed FGGF design, the feasibility depends on whether the filtering results have obvious visual degradation resulting from the approximation, which are evaluated and analyzed as presented in Chapter 5.

4.5.2 Comparisons

There are only a couple of existing non-iterative divider circuits [7], [8] and both of them perform the division as subtraction in the logarithmic domain. The proposed non-iterative division circuits have the accuracy and/or ADP advantages when comparing with the existing counterparts, as shown in Table 4.3. Comparing the stripe-based design by partition of 64 with its predecessors in [7] and [8] that have the

comparable MAEP, a reduction of ADP by 62% and 42% are respectively observed. Comparing the same design by partition of 128 with the single stage logarithmic method [8], the MAEP and ADP are reduced by 45% and 19% respectively. Comparing that with the design from the curve fitting method [100] in this design, there are reductions of MAEP and ADP by 21% and 27% respectively.

Table 4.3: Performance comparisons with existing dividers

Designs	MAE P	Delay (ns)	Area (μm^2)	ADP
Logarithmic 3-stage method[7]	0.21%	5.22	8939	46662
Logarithmic single stage method[8]	0.20%	2.85	10680	30438
Curve fitting method 16x16[100]	0.14%	2.85	12324.9	35126
MILP approximation 16x16	0.08%	2.85	12324.9	35126
Stripe-based approximation 64	0.22%	2.63	6697.4	17614
Stripe-based approximation 128	0.11%	2.79	9155.6	25544
DesignWare instantiation[105]	0%	18.14	3911	70946

The divider from Synopsys DesignWare can provide fully accurate quotient, i.e., 0% MAEP according to (4.7), but suffers from long delay (6.5 times longer), resulting in high ADP (more than 2.7 times larger). Thus the proposed non-iterative divider offers a feasible solution for the situations where high speed computation is required and small errors can be tolerated. At the same time, various design selections are available for specific application requirements, which are further discussed in Chapter 5.

4.6 Summary

The division operation is formulated and modeled in this chapter at the beginning. In the proposed approach, the quotient is modeled as a curved surface and approximated by small square planes or rectangular stripes. Several possible approximation approaches, including the curve fitting method, the mixed integer linear programming method and the stripe-based method are proposed for the optimization of errors and circuit complexities. The hardware architecture presented for the divider consists of only simple operations for reducing circuit complexity and improving circuit performance. Several divider

circuits with various accuracies and performances were obtained and evaluated. The circuits provide a trade-off between accuracy and ADP. The circuits are also compared with other existing designs for non-iterative divider circuit. It is evident that the proposed non-iterative divider provides a solution to applications with high performance demand but can tolerate small errors in the computation results. An application of such divider circuit is for the fast gradient guided image filter as discussed in the next chapter.

Chapter 5

Approximated Circuits and Trade-offs in Proposed Filter Design

5.1 Overview

This chapter describes the improvements of the fast gradient guided filter (FGGF) proposed in Chapter 3, with the two of its standard arithmetic modules replaced with approximated circuits. It is discussed in Chapter 3 and 4 that the delays of the division module and the exponentiation module are the operating frequency bottlenecks of overall FGGF design. A fast non-iterative divider is proposed in Chapter 4, but the exponentiation module is still a bottleneck. The exponentiation module is further studied and analyzed and a fast approximation circuit is proposed. The two approximation modules are integrated into the proposed FGGF architecture and the performance is evaluated in this chapter.

There are several aspects in the proposed FGGF presented in Chapter 3 where the architecture of the design is still not optimized. The stripe-based method for processing a full-HD frame will be discussed in details, considering both performance and down-sampling issues. The trade-offs brought in by the approximated computing, including the non-iterative division and the exponentiation, are discussed. The performance in terms of filtering quality, hardware cost and operating speed are analyzed, evaluated and

compared with existing designs.

The work in this chapter is based upon the previous chapters. The dataflow organization in Section 3.3 and the hardware architecture in Section 3.4 of the proposed FGGF presented in Chapter 3 remain the same. The division operation is replaced by the non-iterative divider presented in Chapter 4. The main contributions of this chapter include the followings:

- **The stripe-based processing procedure for the FGGF is evaluated.** The selection of the stripe width for full-HD frame for optimal processing speed and memory storage is discussed. With down-sampling technique adopted for the proposed FGGF in both the preprocessing and the main computation stage, optimal stripe width is decided to fit that situation.
- **The computation of the complex function γ_k is simplified by piecewise linear approximation.** The computation for γ_k (2.11) involves exponentiation operation and division, making it the most complicated operation in the filter system. This operation is studied and analyzed. A piecewise linear approximation is proposed to achieve the operation with negligible errors, resulting in simpler circuit and faster speed.
- **Operation speed bottleneck is solved with the help of non-iterative division and linear approximation for γ_k .** The longest critical delay in the FGGF design is limited by the implementation of standard exponentiation and division circuits from Synopsys DesignWare. The approximation for both operations causes some accuracy loss but reduces significant delay and increases the overall operating frequency of the filter system, promoting higher throughput.

The application of the FGGF in detail enhancement is used as an example to evaluate the filtering quality of the proposed FGGF. It is shown that the introduction of the approximation modules has little impact on the final output image quality. Several designs with different combinations of non-iterative divisions and/or linear approximation for γ_k are synthesized/implemented in layout. The designs are presented, analyzed and evaluated to obtain an optimal design for the proposed FGGF with a high

throughput of more than 150 frames per second when implemented in STM 65nm CMOS technology. The high throughput not only ensures real-time processing to support full-HD (1920×1080) image filtering, but also allows potential incorporation of even more complex functionalities that need to be accelerated in a hardware system.

5.2 Stripe-based Processing

As mentioned in Chapter 3, for a full-HD frame (1920×1080), the stripe-based processing technique is adopted. In the stripe-based processing, a frame is divided into several vertical stripes. Only one stripe is input and processed at a time as illustrated in Fig. 5.1 where the whole frame is divided into several stripes each of width w_s with an overlap of r pixels at either side of a stripe. If the stripe-based processing had not been applied, the proposed filter would have to internally store all the pixels within the frame, including the guided image pixels and the input image pixels, as well as the summation of the pixel values within each sliding window by box filters. Consequently, a huge amount of on-chip memory would be needed even with the down-sample technique is applied. According to [5], more than 99% of such memory can be saved by applying the stripe-based processing scheme, due to the insights that during each processing step that requires summing of pixel values, only a very small fraction of all pixel data are actually needed.

The internal mechanism of the box filter with its micro-architecture is discussed in details in Section 3.4.1. The box filter is a parameterized module. The scanning or stripe width can be as large as the frame width, or as small as $2r + 1$, where r is the scanning window radius. Increasing the stripe width will result in larger on-chip memory required, while narrower stripe width leads to more computation cycles needed for processing the whole frame due to overlap areas. Note that the on-chip memory needed increases linearly with the stripe width, because the data that need to be stored is linearly proportional to the integral values as illustrated by the orange rows in Fig. 5.1. The details of the scanning process for summing the pixels are described in Section 3.4.1.

Figure 5.1: Stripe-based processing.

The relation between the consumption of computing cycles and stripe width is illustrated in Fig. 5.2 based on the estimate given in (5.1) from [5] with the assumed radius of the scan window ω of 16.

$$Cycles = H_f \cdot W_f / w_s \times \lceil 2r + w_s \rceil \cdot 8 \quad (5.1)$$

where H_f and W_f are the height and width of the frame, and w_s represents the stripe width. Estimations in [5] and [4] are different but show similar decreasing trend of computation cycles with increasing stripe width, thus the latter one is not discussed here. Note that the computation cycles are measured in the same standard as in [5] and [4] for designs that are without down-sampling. Though this computation cost in terms of cycles cannot directly compare with cycles required in the proposed FGGF with double down-sampling, the amount remains a good estimation of the cost of computation for the consideration

of stripe partition. Similar to the selection in [5], the stripe width is selected as 60 pixels. This design point is chosen to balance the the computation cycles and the memory required. The memory is linearly increased as the stripe width and when the stripe width increases beyond this design point, the decrease of computation cycles is not as fast as before this point.

Figure 5.2: Relation between computation cost and stripe width.

The processing of the box filter is conducted stripe by stripe, with the whole frame partitioned into several vertical stripes. All the pixel processing is performed on the pixels within a stripe, as illustrated in Fig. 5.1. Due to the overlapping of the sliding window with neighboring stripes, the incoming pixel can be located in the neighboring stripes. The maximum width of the overlapping area is called the extended overlap stripe (EOS). The width of the EOS equals to two times the radius r of the sliding scan window, resulting from the integrating process that requires all data points within a scan window with radius r , i.e., a side length of $2r + 1$.

The radius of the scan window in both [5] and [4] are 15, i.e., one side length of 31 for their designs. Note that no down-sampling technique is applied in their designs. A simple down-sampling process is illustrated in Fig. 5.3 with a sampling ratio of $s = 4$ and the original radius $r = 4$. Due to the down-sampling process, with the sampling ratio 4 or 8, the original radius of a scan window have to be multiples of 8. Thus, to meet the requirement and for a fair comparison with the existing designs, the scan window radius is selected to be $r = 16$ pixels.

Figure 5.3: Simplified down-sampling illustration, with sampling ratio $s = 4$.

This means for the main computing stage described in Section 3.4.2, a single side of EOS will have a radius r_{s_1} as given in(5.2).

$$\begin{aligned} r_{s_1} &= \frac{r}{s_1} \\ &= \frac{16}{4} = 4 \end{aligned} \tag{5.2}$$

The stripe width of 60/76 (with/without EOS) reported in Table 3.1 is due to the EOS of 8 pixels at each side. The EOS has different width for different sampling ratios. Note that in Table 3.1 and all

other tables after it, the filtering window is 33×33 due to the down-sampling, which is different from the existing designs.

5.3 Approximation of Exponentiation

The parameter γ_k in (2.11) plays an important role as the explicit first-order edge-aware constraint as in (2.12). With γ_k , the filtering quality is significantly enhanced and the structural information on edges is better preserved. The value of γ_k regulates the value of the coefficient a_k (2.12) such that the critical edges of patterns in the image are maintained rather than smoothed. This is illustrated in Fig. 5.4, where three images are provided as examples for edge information analysis. Fig. 5.4(a) shows the original images while Fig. 5.4(b)-(e) are the γ_k values obtained from different variations of the GDGIF algorithm. Fig. 5.4(b) is the γ_k values from the original GDGIF [3], while Fig. 5.4(c)-(e) are the values from the proposed FGGF design without any approximations, with the linear approximation of γ_k and with both the linear approximation of γ_k and non-iterative dividers, respectively. Note that only Fig. 5.4(b) has the same size as the input images in row (a), but Fig. 5.4(c)-(e) are not, due to down-sampling in the proposed design.

For the proposed FGGF design, the filtering outputs in Fig. 5.4(d)-(e) show that the critical edge information is preserved even with down-sampling technique and approximation applied. Comparing Fig. 5.4(c) obtained without down-sampling and Fig. 5.4(d) obtained with down-sampling, it can be seen that only minor detailed information is lost. Thus for a large frame, the down-sampling technique is efficient and effective for computation in the edge-preserving filtering.

The computation of the γ_k value involves complex exponentiation and division functions. If it is directly computed, it will require significant amount of hardware resources as well as cause long delay in the datapath, thus jeopardizing the throughput needed for real-time processing. The design presented in Chapter 3 employs the floating-point exponential module from the Synopsys DesignWare. The worst delay in the design is $17.38ns$ in the STM 90nm CMOS technology due to the existence of the exponentiation module in the datapath.

Figure 5.4: Edge information conveyed by value γ_k from different implementations of the guided filter. (a) Guided image I . (b) γ_k from the original GDGIF design. (c) γ_k from the proposed FGGF design. (d) γ_k from the FGGF design with linear approximation of γ_k . (e) γ_k from the FGGF design with linear approximation γ_k and non-iterative dividers.

To have more efficient computation of the γ_k value in the hardware implementation, the expression (2.11) is analyzed in details. With the term π_t defined in (5.3), γ_k (2.11) can be expressed as (5.4).

$$\pi_t = \frac{4(\chi_k - \mu_\chi)}{\mu_\chi - \min(\chi)} \quad (5.3)$$

$$\gamma_k = 1 - \frac{1}{1 + e^{\pi_t}} \quad (5.4)$$

The term π_t can be computed with two subtractions, one simple left shift operation (multiplied by 4) and a division as the values of χ_k , μ_χ and $\min(\chi)$ are computed in the earlier stages. The relationship of γ_k and π_t is studied and shown in Fig. 5.5 as the red dashed line. Note that the range of π_t is confined in the range $[-8, 8]$ based on the possible range of χ_k and its statistical estimates.

Figure 5.5: Linear approximation of the γ_k constraint.

It is observed from Fig. 5.5 that γ_k has a simple relationship with the complex term π_t , which is usually termed the sigmoid function in literatures. Only small errors are expected if piecewise linear

approximation is applied for this smooth sigmoid curve. With the observation, a five-segment piecewise linear approximation is proposed to approximate the red dashed sigmoid curve in Fig. 5.5 taking the trade-offs between hardware complexity and computation accuracy into consideration. Though piecewise linear approximation is applied, the continuity of the function is maintained, the importance of the first-order constraint γ_k is not compromised. The detailed approximation function is provided in (5.5).

$$\gamma_k = \begin{cases} 0 & \pi_t < -4 \\ \pi_t/16 + 0.25 & -4 \leq \pi_t \leq -2 \\ 3\pi_t/16 + 0.5 & -2 < \pi_t < 2 \\ \pi_t/16 + 0.75 & 2 \leq \pi_t \leq 4 \\ 1 & \pi_t > 4 \end{cases} \quad (5.5)$$

where $\pi_t = 4(\chi_k - \mu_\chi)/(\mu_\chi - \min(\chi))$ as in (2.11).

In the piecewise linear approximation implementation, the original function of γ_k is approximated by five linear segments. The segments are determined to minimize the approximation error as well as to simplify their hardware implementation by choosing only power-of-two terms in the coefficients of the linear functions for the segments as given in (5.5).

The accuracy analysis shows that the maximum error due to the proposed approximation is less than 0.05. To minimize the cost function (2.12), a_k and γ_k should have close values, such that the square term $(a_k - \gamma_k)^2$ can be as small as possible. Hence, the approximation error of 0.05 is negligible considering the range of value a_k near structural edges within a frame is approaching a value of 1. The approximation will not impact the accuracy of the computation results.

At the same time, the reduced complexity will give benefits of not only the reduction of hardware resources but also the improvement of overall processing speed, provided the division delay can be reduced to the same magnitude. With (5.5), the complex computation of (2.11) involving exponentiation can be implemented with a straightforward piecewise linear approximation.

The micro-architecture for the γ_k kernel is designed accordingly and shown in Fig. 5.6. Since $3/16 = 1/8 + 1/16$ in (5.5), the resultant operations are straightforward and involve only shifters for the

input π_t and simple adder trees with several constants. In the kernel, the shifters and adder trees with the constants of the power-of-two terms can compute the five possible values in (5.5). The four MSBs of π_t is then used in a multiplexer to select the result from the computed values corresponding to its range.

Figure 5.6: Micro architecture of linear approximation for γ_k .

The γ_k kernel in the architecture of the FGGF design presented in Fig. 3.2 is replaced by the linear approximation γ_k module in Fig. 5.6. The design was synthesized in STM 90nm CMOS technology again but not implemented in layout due to lack of layout library. A layout in 65nm CMOS technology was implemented instead. The implementation results are summarized in Table 5.1.

The image frame is full-HD of size 1920×1080 and a scan window of radius $r = 16$ (31×33) with a stripe width of 60 pixels and an extended overlap stripe (EOS) of 8-pixel wide on each side. Note that for these three implementations, only the linear approximation for γ_k is used. Thus, the long delay of standard division is still a bottleneck of processing speed for the whole design. This makes the operating frequency f_o stays the same as the original design at $20ns$. Comparing with the original design (Table 3.1) without using the approximation γ_k module, only marginal saving of 3.2% on the design area is achieved. The frame rate, power consumption and energy cost per frame for the gradient guided filtering process remain almost the same. Hence, the efficient approximation γ_k module dose not benefit the performance of the whole design unless the division modules are also replaced with more efficient ones, as discussed in the next section.

Table 5.1: Hardware performance of the proposed FGGF designs with linear approximation module for γ_k kernel, in synthesis and layout.

	Proposed FGGF Implementation with Linear Approximation for γ_k Kernel		
	Synthesis	Synthesis	Layout
Technology	STM-90nm	STM-65nm	STM-65nm
Frame Size	1920 × 1080	1920 × 1080	1920 × 1080
Filter Window	33 × 33	33 × 33	33 × 33
Stripe Width	60/76	60/76	60/76
Design Area (μm^2)	726625	386530	586314
Delay (ns)	20	15.5	15.5
Frequency f_o (MHz)	50	64.5	64.5
Power consumption (mW)	12.15	13.94	18.49
Energy Per Frame (mJ)	0.20	0.18	0.24
On-chip Memory (Byte)	2261	2261	2261
Frame Rate (fps)	60+	75+	75+

5.4 Division Implementation in Proposed Filter Design

Many division operations are required in the proposed FGGF design as can be seen from the dataflow in Section 3.3. Some of them are divisions by constant values, which are turned into multiplications. This applies to all the box filters with different radius and the Mean Kernel in Fig. 3.2. True dividers are required only in the Harmonic Mean Kernel, the γ_k Kernel and the a_k Kernel, where both the numerator and denominator are variables.

As discussed in Section 3.4.2, a_k is set to have 8 valid bits in its fraction part, same as the image pixel values in both I and p . Accordingly the χ_k values have a wordlength of 16. For the non-iterative divider designs in Section 4.2, the number representation has a mantissa of 16 bits and an exponent of 5 bits. This is just right for the division in the Harmonic Mean Kernel and γ_k Kernel. However, the non-iterative divider is not suitable for computing the refactored a_k in (3.3) as both inputs have wordlengths of 32 bits due to the intermediate results before the division. As such, the refactoring is not implemented.

Table 5.2: Implementation results of the proposed FGGF designs with non-iterative divider.

	Proposed FGGF Implementation with Non-iterative Divider Modules		
	Synthesis	Synthesis	Layout
Technology	STM-90nm	STM-65nm	STM-65nm
Frame Size	1920 × 1080	1920 × 1080	1920 × 1080
Filter Window	33 × 33	33 × 33	33 × 33
Stripe Width	60/76	60/76	60/76
Design Area (μm^2)	751403	398062	598334
Delay (ns)	20	15.5	15.5
Frequency f_o (MHz)	50	64.5	64.5
Power consumption (mW)	12.59	14.02	18.12
Energy Per Frame (mJ)	0.21	0.19	0.24
On-chip Memory (Byte)	2261	2261	2261
Frame Rate (fps)	60+	75+	75+

Instead, the original solution in (2.13) is implemented, where Γ_k is first derived in the same way as an intermediate value. The much shorter delay of the proposed non-iterative divider makes this a feasible solution for performance improvement.

The results for the non-iterative dividers are approximated as reported in Chapter 4. Since a_k has only 8 bits for fraction, an accuracy in terms of MAEP better than the LSB (2^{-8}) is preferred. Hence, the stripe-based non-iterative divider with a partition by 64 along the axis m_b is adopted, which gives an error of 0.35% (Table 4.2) that is less than the preferred MAEP of $2^{-8} = 0.39\%$.

Fig. 5.4(e) is a simulation example showing that the incorporation of the non-iterative divider into the filter design does not affect the edge information detection when compared with the result in Fig. 5.4(d) from the design using the accurate divider from DesignWare. It can be seen that there is no visual difference in the results when the γ_k and the division are both approximated. The detail enhancement experiments presented in the next section also verify this.

The hardware designs for proposed FGGF with only the division approximated, without approximating the γ_k function are synthesized in Synopsys Design Compiler with STM 90nm and 65nm CMOS technology, and implemented in Synopsys IC Compiler with STM 65nm CMOS technology. The results

are tabulated in Table 5.2. Similar to the one in Section 5.3, no significant performance improvement is observed with this design even though the delay of the divider is only $2.50ns$. This is because the delay of the γ_k kernel remains large as the main speed bottleneck.

5.5 Performance Evaluation and Comparisons

As in Chapter 3, the detail enhancement application is used to evaluate the filtering quality of the proposed FGGF design with the approximated circuits for the γ_k and division modules, because the edge information is significantly amplified and prominent for visual inspection.

The same source image is used for a quantitative analysis and comparison between the proposed FGGF design with and without approximations, as shown in Fig. 5.7. The original guided image and input image and the detailed enhancement results by the proposed FGGF design and its approximated counterpart are displayed in order. The intensity of one channel (green) sampled at the same line from the results are compared and shown below with the differences between the two are depicted as well, as the black filled areas between the arrows. It is evident that the approximated design delivers almost the same output for the application, except some trivial differences indicated by the red arrows. The example demonstrates that the proposed approximated circuits will not have visible degradation for the same filtering process comparing to the original design proposed in Chapter 3.

Fig. 5.8 shows five example images with the detail enhancement results by different guided image filters. It is evident that the enhancements by the FGGF design with both linear approximation for the γ_k kernel and non-iterative divider designs as in Fig. 5.8(d) show no significant visual differences comparing to the proposed FGGF without using approximation circuits as in Fig. 5.8(c). The top three images also demonstrate that the halo-artifacts are reduced with the proposed FGGF designs, comparing with the original GIF result. However, this is not true for the bottom two images. This is because these two source images are relatively small, with size of 256×256 pixels only. For small frames, the edge aware feature of the proposed FGGF does not work well, since the edge-aware weighting $\Gamma_k(I)$ and

Figure 5.7: The one-channel (green) intensity analysis on detail enhancement for proposed FGGF designs without and with approximations.

first order constraint γ_k may have large variations after the down-sampling process. Hence, the bottom two rows show the limit of the proposed FGGF performance for smaller frames. For small frames, down-sampling might not be required in order to achieve high throughput for real-time applications.

The overall objective of the research is to have a complete and highly efficient VLSI architecture design in terms of both speed and hardware resources that are cost, for a fast gradient guided filtering

Figure 5.8: Detail enhancement for multiple image sources. (a) Guided source image. (b) - (d) Enhancement result by the original GIF, the proposed FGGF and the proposed FGGF with all approximation modules respectively.

process. The proposed VLSI design of the FGGF with the approximation modules is synthesized in the STM 90nm and 65nm CMOS technology using the Synopsys Design Compiler, and is implemented in 65nm CMOS technology using the Synopsys IC Compiler. The implementation results are summarized

in Table 5.3. The proposed designs listed are able to process Full-HD (1920×1080) frames at real time of more than 120 or 150 frames per second (fps). The frame rate is doubled compared to the one achieved by the original FGGF design presented in Chapter 3, since the longest delay in the critical path of the system is reduced with the adoption of the proposed approximation modules. The energy cost per frame remains stable comparing the original FGGF design.

Table 5.3: Hardware performance of the proposed FGGF designs with both approximation circuits.

	Proposed FGGF Implementation with Both Approximation modules		
	Synthesis	Synthesis	Layout
Technology	STM-90nm	STM-65nm	STM-65nm
Frame Size	1920×1080	1920×1080	1920×1080
Filter Window	33×33	33×33	33×33
Stripe Width	60/76	60/76	60/76
Design Area (μm^2)	722189	402445	596673
Delay (ns)	10	7.8	7.8
Frequency f_o (MHz)	100	128	128
Power consumption (mW)	21.96	25.42	46.94
Energy Per Frame (mJ)	0.18	0.17	0.31
On-chip Memory (Byte)	2821	2821	2821
Frame Rate (fps)	120+	150+	150+

The various synthesis and layout implementation results of the guided image filters are compared in Fig. 5.9 in terms of the design area, energy cost for processing a single frame and the processing speed in terms of frame rate. In the figure, the labels “65nm” and “90nm” denote the CMOS technology. The FGGF design is the design without approximation circuits presented in Chapter 3. The extensions “-exp”, “-div” and “-apr” respectively denote the FGGF designs with the exponentiation module approximated, with the divider module approximated and with both modules approximated.

It can be observed that the proposed FGGF designs with only one of the approximation circuits give only slight throughput improvement. On the other hand, the designs with both approximated circuits have significant improvement on the throughput. This is due to the fact that both the original divider module and the original γ_k module are the bottlenecks in the critical path. Thus, optimizing only one of them will not improve the performance. Consequently, the power consumption is increased because the power consumption is proportional to the throughput as the operation frequency increases after both the divider and the γ_k module are approximated.

Figure 5.9: Performance summary on frame rate, design area and energy cost per frame.

The proposed design with approximation modules for the FGGF design achieves significant performance improvement on the processing speed, and at the same time maintains the edge-aware filtering quality at acceptable level, as addressed earlier in this section. The speed advantage of adopting the approximation modules is obvious. The speed improvement allows for even more complicated functionalities to be included inside an even larger image processing system that can harness the benefit from the proposed fast guided filtering architecture.

In addition, the same VLSI architecture can be adopted for even larger frame and is still able to keep the processing at real time. A 4K resolution frame (4096×2160), for instance, can be processed with the proposed approximated FGGF design at $60fps$, with a little adjustment on the stripe width. Thus, the high throughput of the proposed design with the approximation modules allows it to be used for modern video processing applications with ultra-high resolution frames.

5.6 Summary

The design of the proposed FGGF design is examined in details in three aspects. First, the stripe based processing mechanism for full-HD frame is described, with the parameters trade-offs and relationship with down-sampling discussed. Second, the complex γ_k kernel is redesigned with a linear approximation approach to compute the result. Its corresponding filtering quality is evaluated and performance analyzed. This linear approximation module replaces the original exponential module from Designware, improving the processing speed significantly. The non-iterative divider design is included in the proposed FGGF as well for reducing the overall delay in the system. The modules involved division and the use of the divider in the a_k kernel discussed. Last but not least, with all approximated modules, the filtering quality of the FGGF system is evaluated, and the hardware implementations are obtained and analyzed, together with its application advantages discussed.

With the approximation modules incorporated in the proposed FGGF design, the critical path delay of the system is significantly reduced as demonstrated by the increase in operating frequency from $50MHz$ to more than $100MHz$. This enables a frame rate of more than 150 frames per second and also allows the design to incorporate even more functionalities as well as to process images of even higher resolutions while real-time processing is still achievable.

Chapter 6

Conclusions

6.1 Conclusions

The thesis presents a high throughput, highly efficient and low cost architecture for a fast gradient guided image filtering process. The filtering algorithm implemented in the proposed fast gradient guided filter (FGGF) design is among the latest developments of the guided image filtering concepts that improve on the quality at edge-preserving filtering. It is used in many applications such as detail enhancement, HDR compression, saliency detection, haze removal and so on. The design of the proposed FGGF satisfies the requirements of high throughput for real-time processing, efficient chip area, memory size and power consumption. Its performance is comparable to the existing implementation of the original guided image filter (GIF) design, which suffers from the halo-artifacts when texture is strong.

For a good understanding on the evolution of the guided image filtering techniques since its introduction, related literatures are reviewed in Chapter 2. The original concept of GIF is explained in details, based upon which the WGIF, AGIF and GDGIF are reviewed and analyzed. Edge aware weighting is included in the WGIF design to denote the importance of each pixel in a frame. In AGIF, the shift-variant technique is included to adaptively adjust the offset and constant parameters in kernel, so that only pixels at edges are sharpen while others smoothed. The latest GDGIF introduces a first-order

edge-aware constraint γ_k and a novel edge-aware weighting $\Gamma_k(I)$ that considers both global and local info, so that better edge-preserving quality is achieved. The existing hardware implementations for the original GIF are reviewed. The integral technique with a sliding filtering window and the stripe-based processing procedure are studied. The implementation of arithmetic division based on the traditional digit-recurrence and iterative convergence and the non-iterative approximated approach is examined in details.

Based on the software algorithm of GDGIF, a complete dataflow of the FGGF design is proposed, with a preprocessing stage for computing the global information required later for deriving the local linear model, a main computation stage for linear coefficients and the final restore of filtering output stage. A hardware design of the FGGF consisting of three stages is proposed. Three design challenges are addressed in the proposed architecture. First, the preprocessing stage avoids the repetitive and redundant global information computation. Second, down-sampling is adopted to save the high computation cost at the first two stages. Last, parallelism is applied to restore the filtering output at a higher throughput to overcome the complex operation and its long delay. Due to down-sampling and the parallel restoration, the proposed filter system operates at two frequencies for different modules. The integral engine with scanning filter window, together with the stripe-based processing technique are used to reduce the memory cost. The design was synthesized in STM 90nm and 65nm CMOS technology using the Synopsys Design Compiler, implemented in layout in 65nm using the Synopsys IC Compiler. Better performance is observed from these results when comparing with existing designs, in addition to the more advanced filtering features. The filter is able to achieve a throughput of up to 150 frames per second.

After the design of the FGGF was implemented, the computation bottleneck of the filter system was examined. The complex division modules and the γ_k kernel were found to have the longest delay. The modules were analyzed and replaced with approximated circuits. Specifically, non-iterative divider designs based on several approaches, including the curve fitting method, the mixed integer linear programming (MILP) and stripe-based partition techniques, are proposed in Chapter 4, to achieve acceptable computation performance at low cost of design area and low delay. In these designs, the division is represented with a customized number system similar to single precision floating point numbers,

where the computation can be carried out independently at mantissa and exponent part. The computation of the quotient mantissa is modeled as a curved surface. The basic idea is to represent the curved surface with flat planes for simple hardware implementation. To reduce the approximation errors introduced, the curved surface is partitioned into many smaller ones, with each approximated by a flat plane within its region. An optimal approximation is achieved with a shift from the curve fitting method to the MILP method. Closer observation shows that non-linearity comes only from the denominator, another improvement by replacing the evenly partitions to the stripe-based partitions along only the denominator mantissa is made to achieve designs with less area cost. The proposed approaches for divider designs result in several divider circuits with various combinations of accuracy and area-delay product, providing trade-offs to suit the requirements of different applications, including the FGGF design.

Linear approximation is adopted in the γ_k kernel to replace the exponentiation module that suffers from long delay. Only simple shifting and addition are used to achieve the linear approximation. Thus, much smaller area and delay are achieved. The filtering quality in the application of detail enhancement are examined to show that the linear approximation has little impact on both the structural information and the final result. The selection of proper arithmetic circuits for the complex division and the γ_k kernel in the filter system is examined with the consideration of the trade-offs of accuracy and computation complexity. The filtering qualities of different selections in the design space are compared and analyzed. An optimal FGGF hardware design that uses the approximated circuits, achieves good filtering quality and good performance is obtained. It is shown that higher throughput (150fps) can be achieved with the improved design, which allows not only even more complex arithmetic tasks to be included in the system for more complicated applications, but also makes the real-time processing of higher resolution (4K for example) possible.

Therefore, the research objectives set in Chapter 1 are achieved. The design area of the proposed filter is comparable to the existing GIF design. The filter is able to achieve a throughput of up to 75 frames per second without the approximation modules. The non-iterative divider has 42% less ADP comparing to the existing logarithm-based single stage method (Table 4.3). In addition, the throughput of the overall system is increased to 150 frames per second when the approximation circuits are

employed (Table 5.3). In a nutshell, the performance requirements are satisfied by the proposed filter design.

In summary, the contributions of the thesis are as below.

- (i) A VLSI architecture for gradient guided image filtering is proposed and implemented. The proposed design has the following features.
  - It is the first VLSI design for gradient guided image filtering.
  - The gradient guided image filter is able to mitigate the filtering problem of halo-artifacts.
  - The required computation in the filtering process that involves global information is dealt by a fast preprocessing stage that uses down-sampling technique.
  - The complex arithmetic operations, including division and exponentiation, are implemented by approximation circuits for high performance without jeopardizing the filtering quality.
  - Parallel processing is applied at the output stage on all the pixels of the whole image to restore the image to its full size.
  - It can achieve a high throughput of up to 150 frames per second due to the parallel output processing stage and the short datapath delay in the complex arithmetic modules.
- (ii) New non-iterative divider circuits based on three different approaches are proposed to replace the divider in the filter design that causes the performance bottleneck. The proposed divider circuits provide trade-off of accuracy and area-delay product to suit different application requirements.
- (iii) The performance of the filter, including the filtering quality, speed and hardware cost, is analyzed in details with its performance bottlenecks identified and the arithmetic modules that causes the bottlenecks replaced by the newly proposed approximation circuits. The design is improved with its operating frequency increased from $50MHz$ to $100MHz$ and the throughput from 60 to 150 frames per second while the size remains comparable.

During the design of the VLSI architecture of the GDGIF, many lessons are learnt from both the high level organization and the detailed circuit design choice. These lessons can be summarized as below.

- **Every design platform has its advantages and disadvantages and proper consideration should be made when choosing one for a specific problem.** For the case of the gradient guided filtering task, it can be conducted in more than one hardware platforms, including general purpose processors, GPUs, FPGAs and ASICs. Considering the application requirements on processing speed, power consumption and potential application on mobile devices, ASICs is selected as the proper platform to solve this problem.
- **Problems should be analyzed so as to extract common computations for hardware resource sharing.** In the proposed design, the global information computation is extracted to form the preprocessing stage, which feeds the global data that are required by each pixel in the main stage on the computation of the linear coefficients. Otherwise, the same calculation would be conducted over and over again for each and every local processing unit. Even though the image is down-sampled, the hardware resource would still be significantly large.
- **The static and dynamic elements required in operation should be properly identified.** For the implementation of the box filter, the scanning process of the filtering window is carefully studied in order to find out the relationship between the two neighboring scanning windows, which is decided by the values of only a few pixels. The identifications of the dynamic elements (the pixels that are different from the previous scanning window) result in highly efficient computation of the integrating process in the box filter.
- **Sampling can cost little, but save a lot.** Sampling process in the filter causes acceptable degradation in the quality of filtering result, but leads to a significant reduction in the hardware resources required as well as enables high throughput.

- **Develop a complete working design and then improve its performance.** The whole design process of the proposed filter architecture is complex and it involves several hardware complicated modules such as computation of means and minimum, division and exponentiation. Without a complete design, finding the bottlenecks in the critical path could not be possible. The final performance of the proposed design in this work is achieved only after the bottlenecks are located one after each other.
- **Approximation depends on identification of linearity.** For both the approximated non-iterative divider and the simplification of the exponentiation, the approximation methods are found after the computations are analyzed to identify the linearity in the operations. First, the quotient of division is modeled as a curved surface partitioned into many planes. The linearity is found in the denominator to further simplify the partition in just one dimension instead of two, i.e. stripe-based partition instead of square plane partition. Similar observation results in the linear approximation of the exponentiation. After the linearity is identified, the extent to which linearity is applied is the trade-offs between the computation accuracy required, hardware complexity and speed.
- **Parallel processing on simple computation provides boost on performance with low cost.** Parallel processing provides an approach to significantly increase the throughput of a system. It requires the processing operations to have same the steps and input data format. If the shared structure or processing data is too large, the hardware resources required will be costly. In the proposed design, parallel processing is applied only at the last step for simple linear operations, which makes the last stage small and exquisite. The up-sampling process uses the same computed linear coefficients for sixteen pixels, and operates at a higher frequency, which boosts the throughput.

6.2 Future work

Further investigation on the computation bottleneck of the proposed FGGF design can be a possible direction for the future works. Specifically, the current square root operation has the longest delay in the datapath, which could be replaced with an approximation circuit to further increase the speed and throughput. Furthermore, GPU shows another option to have the gradient guided image filtering process implemented in a hardware system for those applications where the power consumption and area is not critical. The comparison of pros and cons between the VLSI design and the GPU realization of similar applications can be interesting. Integrating the proposed FGGF design into larger hardware systems for more complicated tasks in computer vision or graphics is also a possible future work. The embedded vision system sometimes require real-time processing for stereo matching and saliency detection. Hence, the possible futures are summarized as below.

- (i) **Approximated circuits design for square root operation.** As a common and complex operation, especially one that occurs in the proposed fast gradient guided image filter, it is meaningful to develop a more efficient module to carry out the square root, with the help of approximation methods, such that the critical delay path can be further reduced. Merging the proposed approximation modules for square root into the fast gradient guided filter should further increase the operating frequency, due to the reduced longest delay in each pipelined stage.
- (ii) **Real-time processing of 4K resolution HD frame.** With the approximation modules for the complex division, the linear approximation for γ_k and potential possible solution for the square root, the operating frequency can be further boosted. The real-time filtering for 4K resolution frame could be possible and its performance could be investigated. The choices of design parameters such as the stripe width for the stripe-based method can be adjusted according to the larger frame size. The limit of frame size can also be investigated for the proposed FGGF design to determine at what frame size the real-time processing becomes impractical.

- (iii) **Compare the proposed fast gradient guided filter with an implementation in GPU.** The proposed design and the processing in GPU both deploy the parallelism in the processing. If the gradient guided filtering is implemented as a GPU mapping, the different levels of parallelism at the architecture level, and the instruction level can be investigated. The design approaches and the corresponding performance in both platforms could be compared and analyzed. The results obtained can be helpful for the selections of different implementation approaches for different application scenarios.
- (iv) **System design for the application of proposed fast gradient guided filter in stereo matching / saliency detection.** The current work focuses on the hardware design of the filter, with the performance analyzed for the application mainly on the detail enhancement. With the better performance on the edge-preserving filtering as the preprocessing for stereo matching or saliency detection, the feasibility of developing a compact system for completing the whole procedure of matching the images taken from a stereo camera and extracting the depth or finding the significant part of images/video can be studied. Such a system could incorporate the proposed fast gradient guided filter, and at the same time achieve real-time performance with the help of all the approximation circuits. It could replace software processing in some applications where real-time processing is required, e.g. an embedded vision system in driver-less autonomous vehicles.

Publications

Conference Papers

- (i) Lei Wu and Ching Chuen Jong, “A VLSI architecture for real-time gradient guided image filtering,” in *2016 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 1–6, Sept 2016.
- (ii) Lei Wu and Ching Chuen Jong, “A curve fitting approach for non-iterative divider design with accuracy and performance trade-off,” in *2015 IEEE 13th International New Circuits and Systems Conference (NEWCAS)*, pp. 1–4, June 2015.

Bibliography

- [1] K. He, J. Sun, and X. Tang, “Guided image filtering,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 6, pp. 1397–1409, 2013.
- [2] Z. Li, J. Zheng, Z. Zhu, W. Yao, and S. Wu, “Weighted guided image filtering,” *IEEE Transactions on Image Processing*, vol. 24, pp. 120–129, Jan 2015.
- [3] F. Kou, W. Chen, C. Wen, and Z. Li, “Gradient domain guided image filtering,” *IEEE Transactions on Image Processing*, vol. 24, pp. 4528–4539, Nov 2015.
- [4] C.-C. Kao, J.-H. Lai, and S.-Y. Chien, “VLSI architecture design of guided filter for 30 frames/s full-HD video,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, pp. 513–524, March 2014.
- [5] Y. C. Tseng, P. H. Hsu, and T. S. Chang, “A 124 mpixels/s VLSI design for histogram-based joint bilateral filtering,” *IEEE Transactions on Image Processing*, vol. 20, pp. 3231–3241, Nov 2011.
- [6] F. Sheikh, S. K. Mathew, M. A. Anders, H. Kaul, S. K. Hsu, A. Agarwal, R. K. Krishnamurthy, and S. Borkar, “A 2.05 Gvertices/s 151 mW lighting accelerator for 3D graphics vertex and pixel shading in 32 nm CMOS,” *IEEE Journal of Solid-State Circuits*, vol. 48, no. 1, pp. 128–139, 2013.
- [7] H. Kim, B.-G. Nam, J.-H. Sohn, J.-H. Woo, and H.-J. Yoo, “A 231-MHz, 2.18-mW 32-bit logarithmic arithmetic unit for fixed-point 3-D graphics system,” *IEEE Journal of Solid-State Circuits*, vol. 41, no. 11, pp. 2373–2381, 2006.

- [8] J. Y. L. Low and C. C. Jong, “Non-iterative high speed division computation based on Mitchell logarithmic method,” in *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2219–2222, IEEE, 2013.
- [9] B. Jahne, *Digital Image Processing*, vol. 4. Springer, 2005.
- [10] T. Acharya and A. K. Ray, *Image Processing: Principles and Applications*. John Wiley & Sons, 2005.
- [11] R. C. Gonzalez and R. E. Woods, *Digital Image Processing (3rd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006.
- [12] R. Jain, R. Kasturi, and B. G. Schunck, *Machine Vision*, vol. 5. McGraw-Hill New York, 1995.
- [13] A. M. Tekalp, *Digital Video Processing*. Prentice Hall Press, 2015.
- [14] L. K. Choi, Y. Liao, and A. C. Bovik, “Video QoE models for the compute continuum,” *IEEE Multimedia Communications Technical Committee (MMTC) E-Letters*, 2013.
- [15] A. C. Bovik, “Automatic prediction of perceptual image and video quality,” *Proceedings of the IEEE*, vol. 101, pp. 2008–2024, Sept 2013.
- [16] M. P. Ekstrom, *Digital Image Processing Techniques*, vol. 2. Academic Press, 2012.
- [17] T. Law, H. Itoh, and H. Seki, “Image filtering, edge detection, and edge tracing using fuzzy reasoning,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, pp. 481–491, May 1996.
- [18] S. Borse and P. K. Bora, “A novel approach to image edge detection using Kalman filtering,” in *2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pp. 1–5, Oct 2016.
- [19] M. Sharifi, M. Fathy, and M. T. Mahmoudi, “A classified and comparative study of edge detection algorithms,” in *Proceedings. International Conference on Information Technology: Coding and Computing*, pp. 117–120, April 2002.
- [20] K.-S. Fu, *VLSI for Pattern Recognition and Image Processing*, vol. 13. Springer Science & Business Media, 2013.

- [21] Y. Taur and T. H. Ning, *Fundamentals of Modern VLSI Devices*. Cambridge university press, 2013.
- [22] I. Kuon and J. Rose, “Measuring the gap between FPGAs and ASICs,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, 2007.
- [23] R. C. Nelson, “Finding line segments by stick growing,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 5, pp. 519–523, 1994.
- [24] M. Hollins and S. R. Risner, “Evidence for the duplex theory of tactile texture perception,” *Attention, Perception, & Psychophysics*, vol. 62, no. 4, pp. 695–705, 2000.
- [25] P. Pirsch and H. J. Stolberg, “VLSI implementations of image and video multimedia processing systems,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, pp. 878–891, Nov 1998.
- [26] H. Jiang, *Design Issues in VLSI Implementation of Image Processing Hardware Accelerators - Methodology and Implementation*. PhD thesis, Lund University, 2007.
- [27] K. He, J. Sun, and X. Tang, “Guided image filtering,” in *Computer Vision—ECCV 2010*, pp. 1–14, Springer, 2010.
- [28] I. T. Young and L. J. Van Vliet, “Recursive implementation of the gaussian filter,” *Signal processing*, vol. 44, no. 2, pp. 139–151, 1995.
- [29] G. Deng and L. Cahill, “An adaptive gaussian filter for noise reduction and edge detection,” in *Nuclear Science Symposium and Medical Imaging Conference, 1993., 1993 IEEE Conference Record.*, pp. 1615–1619, IEEE, 1993.
- [30] E. Oho, N. Baba, M. Katoh, T. Nagatani, M. Osumi, K. Amako, and K. Kanaya, “Application of the laplacian filter to high-resolution enhancement of SEM images,” *Microscopy Research and Technique*, vol. 1, no. 4, pp. 331–340, 1984.
- [31] D. Forsyth and J. Ponce, *Computer Vision: a Modern Approach*. Upper Saddle River, NJ; London: Prentice Hall, 2011.
- [32] C. Tomasi and R. Manduchi, “Bilateral filtering for gray and color images,” in *1998 Sixth International Conference on Computer Vision*, pp. 839–846, Jan 1998.

- [33] F. Durand and J. Dorsey, “Fast bilateral filtering for the display of high-dynamic-range images,” *ACM Trans. Graph.*, vol. 21, pp. 257–266, July 2002.
- [34] C. Ttofis and T. Theocharides, “High-quality real-time hardware stereo matching based on guided image filtering,” in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pp. 1–6, March 2014.
- [35] J. O. Cadenas, G. M. Megson, and R. S. Sherratt, “Median filter architecture by accumulative parallel counters,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, pp. 661–665, July 2015.
- [36] K. He and J. Sun, “Fast guided filter,” *arXiv preprint arXiv:1505.00996*, 2015.
- [37] C. Tomasi and R. Manduchi, “Bilateral filtering for gray and color images,” in *1998 Sixth International Conference on Computer Vision*, pp. 839–846, Jan 1998.
- [38] W. Gan, X. Wu, W. Wu, X. Yang, C. Ren, X. He, and K. Liu, “Infrared and visible image fusion with the use of multi-scale edge-preserving decomposition and guided image filter,” *Infrared Physics & Technology*, vol. 72, pp. 37–51, 2015.
- [39] L. Wang and G. Jeon, “Bayer pattern CFA demosaicking based on multi-directional weighted interpolation and guided filter,” *IEEE Signal Processing Letters*, vol. 22, no. 11, pp. 2083–2087, 2015.
- [40] F. Kou, W. Chen, Z. Li, and C. Wen, “Content adaptive image detail enhancement,” *IEEE Signal Processing Letters*, vol. 22, no. 2, pp. 211–215, 2015.
- [41] R. Rajendran, S. P. Rao, S. S. Agaian, and K. Panetta, “A versatile edge preserving image enhancement approach for medical images using guided filter,” in *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 002341–002346, Oct 2016.
- [42] H. Yang, K. Kim, G. E. Fakhri, K. Kang, Y. Xing, and Q. Li, “Dual-energy CT reconstruction using guided image filtering,” in *2016 IEEE Nuclear Science Symposium, Medical Imaging Conference and Room-Temperature Semiconductor Detector Workshop (NSS/MIC/RTSD)*, pp. 1–4, Oct 2016.

- [43] W. Zhao, Q. Dai, Y. Zheng, and L. Wang, “A new pansharpen method based on guided image filtering: A case study over gaofen-2 imagery,” in *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pp. 3766–3769, July 2016.
- [44] F. Kou, Z. Li, C. Wen, and W. Chen, “Multi-scale exposure fusion via gradient domain guided image filtering,” in *2017 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1105–1110, July 2017.
- [45] T. O. Aydin, N. Stefanoski, S. Croci, M. Gross, and A. Smolic, “Temporally coherent local tone mapping of HDR video,” *ACM Trans. Graph.*, vol. 33, no. 6, p. 196, 2014.
- [46] Q. Zhu, J. Mai, and L. Shao, “A fast single image haze removal algorithm using color attenuation prior,” *IEEE Transactions on Image Processing*, vol. 24, no. 11, pp. 3522–3533, 2015.
- [47] K. Tang, J. Yang, and J. Wang, “Investigating haze-relevant features in a learning framework for image dehazing,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2995–3000, 2014.
- [48] Y.-K. Wang and C.-T. Fan, “Single image defogging by multiscale depth fusion,” *IEEE Transactions on Image Processing*, vol. 23, no. 11, pp. 4826–4837, 2014.
- [49] Z. Li and J. Zheng, “Edge-preserving decomposition-based single image haze removal,” *IEEE Transactions on Image Processing*, vol. 24, no. 12, pp. 5432–5441, 2015.
- [50] A. Hosni, C. Rhemann, M. Bleyer, C. Rother, and M. Gelautz, “Fast cost-volume filtering for visual correspondence and beyond,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 2, pp. 504–511, 2013.
- [51] L. De-Maeztu, S. Mattoccia, A. Villanueva, and R. Cabeza, “Linear stereo matching,” in *2011 IEEE International Conference on Computer Vision (ICCV)*, pp. 1708–1715, IEEE, 2011.
- [52] Q. Yang, “Stereo matching using tree filtering,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 4, pp. 834–846, 2015.
- [53] K. Wang, L. Lin, J. Lu, C. Li, and K. Shi, “PISA: Pixelwise image saliency by aggregating complementary appearance contrast measures with edge-preserving coherence,” *IEEE Transactions on Image Processing*, vol. 24, no. 10, pp. 3019–3033, 2015.

- [54] Y. Ding, J. Xiao, and J. Yu, “Importance filtering for image retargeting,” in *2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 89–96, IEEE, 2011.
- [55] X. Kang, S. Li, and J. A. Benediktsson, “Spectral–spatial hyperspectral image classification with edge-preserving filtering,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 52, no. 5, pp. 2666–2677, 2014.
- [56] R. Fattal, M. Agrawala, and S. Rusinkiewicz, “Multiscale shape and detail enhancement from multi-light image collections,” *ACM Trans. Graph.*, vol. 26, no. 3, p. 51, 2007.
- [57] F. Durand and J. Dorsey, “Fast bilateral filtering for the display of high-dynamic-range images,” in *ACM Trans. Graph.*, vol. 21, pp. 257–266, ACM, 2002.
- [58] Z. Farbman, R. Fattal, D. Lischinski, and R. Szeliski, “Edge-preserving decompositions for multi-scale tone and detail manipulation,” *ACM Trans. Graph.*, vol. 27, pp. 67:1–67:10, Aug. 2008.
- [59] Z. Li, J. Zheng, and Z. Zhu, “Content adaptive guided image filtering,” in *2014 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1–6, July 2014.
- [60] B. Zhang and J. P. Allebach, “Adaptive bilateral filter for sharpness enhancement and noise removal,” *IEEE Transactions on Image Processing*, vol. 17, pp. 664–678, May 2008.
- [61] C. C. Pham and J. W. Jeon, “Efficient image sharpening and denoising using adaptive guided image filtering,” *IET Image Processing*, vol. 9, no. 1, pp. 71–79, 2015.
- [62] B. Ham, M. Cho, and J. Ponce, “Robust guided image filtering using nonconvex potentials,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, pp. 192–207, Jan 2018.
- [63] K. Lu, S. You, and N. Barnes, “Double-guided filtering: Image smoothing with structure and texture guidance,” in *2017 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, pp. 1–8, Nov 2017.
- [64] G. A. Kordelas, D. S. Alexiadis, P. Daras, and E. Izquierdo, “Content-based guided image filtering, weighted semi-global optimization, and efficient disparity refinement for fast and accurate disparity estimation,” *IEEE Transactions on Multimedia*, vol. 18, pp. 155–170, Feb 2016.

- [65] C. H. Chang, I. C. Huang, M. F. Lin, and S. R. Kuang, "Design and implementation of a low-cost guided image filter for underwater image enhancement," in *2017 IEEE Conference on Dependable and Secure Computing*, pp. 296–299, Aug 2017.
- [66] X. Zhang, H. Sun, S. Chen, and N. Zheng, "VLSI architecture exploration of guided image filtering for 1080p at 60hz video processing," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, pp. 230–241, Jan 2018.
- [67] L. Dai, M. Yuan, Z. Li, X. Zhang, and J. Tang, "Hardware-efficient guided image filtering for multi-label problem," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4905–4913, July 2017.
- [68] C. Ttofis, C. Kyrkou, and T. Theocharides, "A low-cost real-time embedded stereo vision system for accurate disparity estimation based on guided image filtering," *IEEE Transactions on Computers*, vol. 65, pp. 2678–2693, Sept 2016.
- [69] L. Chen, J. Han, W. Liu, and F. Lombardi, "On the design of approximate restoring dividers for error-tolerant applications," *IEEE Transactions on Computers*, vol. 65, pp. 2522–2533, Aug 2016.
- [70] S. F. Oberman and M. Flynn, "Division algorithms and implementations," *IEEE Transactions on Computers*, vol. 46, no. 8, pp. 833–854, 1997.
- [71] B. Pasca, "Correctly rounded floating-point division for DSP-enabled FPGAs," in *2012 22nd International Conference on Field Programmable Logic and Applications (FPL)*, pp. 249–254, IEEE, 2012.
- [72] K. Pande, A. Parkhi, S. Jaykar, and A. Peshattiwar, "Design and implementation of floating point divide-add fused architecture," in *2015 Fifth International Conference on Communication Systems and Network Technologies*, pp. 797–800, April 2015.
- [73] K. Yoshida, T. Sakamoto, and T. Hase, "A 3D graphics library for 32-bit microprocessors for embedded systems," *IEEE Transactions on Consumer Electronics*, vol. 44, no. 3, pp. 1107–1114, 1998.

- [74] L. Wu and C. C. Jong, "A VLSI architecture for real-time gradient guided image filtering," in *2016 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 1–6, Sept 2016.
- [75] R. Woo, S. Choi, J.-H. Sohn, S.-J. Song, and H.-J. Yoo, "A 210-mW graphics LSI implementing full 3-D pipeline with 264 Mtexels/s texturing for mobile multimedia applications," *IEEE Journal of Solid-State Circuits*, vol. 39, no. 2, pp. 358–367, 2004.
- [76] J. J. Cavanagh, *Digital Computer Arithmetic: Design and Implementation*. 1982.
- [77] M. D. Ercegovic and T. Lang, *Division and Square Root: Digit-Recurrence Algorithms and Implementations*. Kluwer Academic Publishers, 1994.
- [78] T. Lang and A. Nannarelli, "A radix-10 digit-recurrence division unit: algorithm and architecture," *IEEE Transactions on Computers*, vol. 56, no. 6, 2007.
- [79] E. Antelo, T. Lang, P. Montuschi, and A. Nannarelli, "Digit-recurrence dividers with reduced logical depth," *IEEE Transactions on Computers*, vol. 54, no. 7, pp. 837–851, 2005.
- [80] J. Ebergen, I. Sutherland, and A. Chakraborty, "New division algorithms by digit recurrence," in *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers, 2004.*, vol. 2, pp. 1849–1855, IEEE, 2004.
- [81] M. Baesler, S.-O. Voigt, and T. Teufel, "FPGA implementations of radix-10 digit recurrence fixed-point and floating-point dividers," in *2011 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, pp. 13–19, IEEE, 2011.
- [82] Z. Wang, L. Han, and S.-B. Ko, "Design and implementation of a radix-100 division unit," in *2012 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1239–1242, IEEE, 2012.
- [83] N. Takagi and S. Kuwahara, "Digit-recurrence algorithm for computing euclidean norm of a 3-D vector," in *14th IEEE Symposium on Computer Arithmetic, 1999. Proceedings*, pp. 86–93, IEEE, 1999.
- [84] D. L. Harris, S. F. Oberman, and M. A. Horowitz, "SRT division architectures and implementations," in *Proceedings 13th IEEE Symposium on Computer Arithmetic*, pp. 18–25, Jul 1997.

- [85] J.-A. Pineiro and J. D. Bruguera, "High-speed double-precision computation of reciprocal, division, square root, and inverse square root," *IEEE Transactions on Computers*, vol. 51, no. 12, pp. 1377–1388, 2002.
- [86] H. Nikmehr, B. Phillips, and C.-C. Lim, "Fast decimal floating-point division," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 9, pp. 951–961, 2006.
- [87] N. Sorokin, "Implementation of high-speed fixed-point dividers on FPGA," *Journal of Computer Science & Technology*, vol. 6, 2006.
- [88] M. McCann and N. Pippenger, "SRT division algorithms as dynamical systems," *SIAM Journal on Computing*, vol. 34, no. 6, pp. 1279–1301, 2005.
- [89] D. R. Kelly, B. J. Phillips, and S. Al-Sarawi, "Increasing throughput of a RISC architecture using arithmetic data value speculation," in *2009 Conference Record of the Forty-Third Asilomar Conference on Signals, Systems and Computers*, pp. 915–920, Nov 2009.
- [90] G. Gerwig, H. Wetter, E. M. Schwarz, and J. Haess, "High performance floating-point unit with 116 bit wide divider," in *16th IEEE Symposium on Computer Arithmetic, 2003. Proceedings.*, pp. 87–94, IEEE, 2003.
- [91] M. Flynn, "On division by functional iteration," *IEEE Transactions on Computers*, vol. 100, no. 8, pp. 702–706, 1970.
- [92] M. J. Schulte, J. Omar, and E. Swartzlander, "Optimal initial approximations for the Newton-Raphson division algorithm," *Computing*, vol. 53, no. 3, pp. 233–242, 1994.
- [93] L.-K. Wang and M. J. Schulte, "Decimal floating-point division using Newton-Raphson iteration," in *15th IEEE International Conference on Application-Specific Systems, Architectures and Processors, 2004. Proceedings*, pp. 84–95, IEEE, 2004.
- [94] N. M. Nenadic and S. B. Mladenovic, "Fast division on fixed-point DSP processors using Newton-Raphson method," in *The International Conference on Computer as a Tool, 2005. EUROCON 2005*, vol. 1, pp. 705–708, IEEE, 2005.

BIBLIOGRAPHY

- [95] H. Dawid and G. Fettweis, “Bit-level systolic carry-save array division,” in *Global Telecommunications Conference, 1992. Conference Record., GLOBECOM '92. Communication for Global Users., IEEE*, pp. 484–488 vol.1, Dec 1992.
- [96] IEEE Standards Committee, “IEEE standard for floating-point arithmetic,” *IEEE Computer Society Std*, vol. 2008, pp. 1–70, Aug 2008.
- [97] N. Ide, M. Hirano, Y. Endo, S. Yoshioka, H. Murakami, A. Kunitatsu, T. Sato, T. Kamei, T. Okada, and M. Suzuoki, “2.44-GFLOPS 300-MHz floating-point vector-processing unit for high-performance 3D graphics computing,” *IEEE Journal of Solid-State Circuits*, vol. 35, no. 7, pp. 1025–1033, 2000.
- [98] J. N. Mitchell, “Computer multiplication and division using binary logarithms,” *IRE Transactions on Electronic Computers*, no. 4, pp. 512–517, 1962.
- [99] J. Y. L. Low and C. C. Jong, “Unified Mitchell-based approximation for efficient logarithmic conversion circuit,” *IEEE Transactions on Computers*, 2014.
- [100] L. Wu and C. C. Jong, “A curve fitting approach for non-iterative divider design with accuracy and performance trade-off,” in *2015 IEEE 13th International New Circuits and Systems Conference (NEWCAS)*, pp. 1–4, June 2015.
- [101] R. C. Gonzalez and R. E. Woods, “Image processing,” *Digital image processing*, vol. 2, 2007.
- [102] S. Paris and F. Durand, “A fast approximation of the bilateral filter using a signal processing approach,” *Computer Vision–ECCV 2006*, pp. 568–580, 2006.
- [103] F. Porikli, “Constant time $O(1)$ bilateral filtering,” in *2008 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pp. 1–8, IEEE, 2008.
- [104] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing: Design and Analysis of Algorithms*, vol. 400. Benjamin/Cummings Redwood City, CA, 1994.
- [105] “Synopsys DesignWare Library.” <https://www.synopsys.com/designware-ip/soc-infrastructure-ip/designware-library.html>.
- [106] G. Appa and C. Smith, “On L1 and Chebyshev estimation,” *Mathematical Programming*, vol. 5, no. 1, pp. 73–87, 1973.