UNIVERSITÉ DE MONTRÉAL

FLEXIBLE HARDWARE ARCHITECTURES FOR RETINAL IMAGE ANALYSIS

HAMZA BENDAOUDI

DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION

DU DIPLÔME DE PHILOSOPHIAE DOCTOR

(GÉNIE INFORMATIQUE)

AVRIL 2017

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée:

FLEXIBLE HARDWARE ARCHITECTURES FOR RETINAL IMAGE ANALYSIS

présentée par: BENDAOUDI Hamza

en vue de l'obtention du diplôme de : Philosophiae Doctor

a été dûment acceptée par le jury d'examen constitué de :

M. BOIS Guy, Ph. D., président

M. LANGLOIS J.M. Pierre, Ph. D., membre et directeur de recherche

Mme CHERIET Farida, Ph. D., membre et codirectrice de recherche

M. BILODEAU Guillaume-Alexandre, Ph. D., membre

M. BLAQUIÈRE Yves, Ph. D., membre externe

DEDICATION

For my Mom, Dad and ‘brosters’

ACKNOWLEDGEMENTS

I would like to express my sincere and greatest gratitude to my supervisors, Dr. Pierre Langlois and Dr. Farida Cheriet. I am thankful for their constant support and encouragement and for their constructive advice and comments.

I am also thankful for my colleagues in the LASNEP and LIV4D, for the good times, mutual support and encouragement that we shared.

I will always be thankful to the faculty, staff and students of the department who helped me to expand my knowledge and expertise and appreciate my work even more. I would also like to thank them for every bit of help they provided each in his way.

My eternal gratitude to my family and friends for their unconditional encouragement and support.

RÉSUMÉ

Des millions de personnes autour du monde sont touchées par le diabète. Plusieurs complications oculaires telle que la rétinopathie diabétique sont causées par le diabète, ce qui peut conduire à une perte de vision irréversible ou même la cécité si elles ne sont pas traitées. Des examens oculaires complets et réguliers par les ophtalmologues sont nécessaires pour une détection précoce des maladies et pour permettre leur traitement. Comme solution préventive, un protocole de dépistage impliquant l'utilisation d'images numériques du fond de l'œil a été adopté. Cela permet aux ophtalmologistes de surveiller les changements sur la rétine pour détecter toute présence d'une maladie oculaire. Cette solution a permis d'obtenir des examens réguliers, même pour les populations des régions éloignées et défavorisées. Avec la grande quantité d'images rétiniennes obtenues, des techniques automatisées pour les traiter sont devenues indispensables. Les techniques automatisées de détection des maladies des yeux ont été largement abordées par la communauté scientifique. Les techniques développées ont atteint un haut niveau de maturité, ce qui a permis entre autre le déploiement de solutions en télémédecine.

Dans cette thèse, nous abordons le problème du traitement de volumes élevés d'images rétiniennes dans un temps raisonnable dans un contexte de dépistage en télémédecine. Ceci est requis pour permettre l'utilisation pratique des techniques développées dans le contexte clinique. Dans cette thèse, nous nous concentrons sur deux étapes du pipeline de traitement des images rétiniennes. La première étape est l'évaluation de la qualité de l'image rétinienne. La deuxième étape est la segmentation des vaisseaux sanguins rétiniens.

L’évaluation de la qualité des images rétinienne après acquisition est une tâche primordiale au bon fonctionnement de tout système de traitement automatique des images de la rétine. Le rôle de cette étape est de classifier les images acquises selon leurs qualités, et demander une nouvelle acquisition en cas d’image de mauvaise qualité. Plusieurs algorithmes pour évaluer la qualité des images rétiniennes ont été proposés dans la littérature. Cependant, même si l'accélération de cette tâche est requise en particulier pour permettre la création de systèmes mobiles de capture d'images rétiniennes, ce sujet n'a pas encore été abordé dans la littérature. Dans cette thèse, nous ciblons un algorithme qui calcule les caractéristiques des images pour permettre leur classification en mauvaise, moyenne ou bonne qualité. Nous avons identifié le calcul des caractéristiques de l'image comme une tâche répétitive qui nécessite une accélération. Nous nous sommes intéressés plus particulièrement à l’accélération de l’algorithme d’encodage à longueur de séquence (*Run-Length Matrix* – RLM). Nous avons proposé une première implémentation complètement logicielle mise en œuvre sous forme d’un système embarqué basé sur la technologie Zynq de Xilinx. Pour accélérer le calcul des caractéristiques, nous avons conçu un co-processeur capable de calculer les caractéristiques en parallèle implémenté sur la logique programmable du FPGA Zynq. Nous avons obtenu une accélération de 30,1 × pour la tâche de calcul des caractéristiques de l’algorithme RLM par rapport à son implémentation logicielle sur la plateforme Zynq.

La segmentation des vaisseaux sanguins rétiniens est une tâche clé dans le pipeline du traitement des images de la rétine. Les vaisseaux sanguins et leurs caractéristiques sont de bons indicateurs de la santé de la rétine. En outre, leur segmentation peut également aider à segmenter les lésions rouges, indicatrices de la rétinopathie diabétique. Plusieurs techniques de segmentation des vaisseaux sanguins rétiniens ont été proposées dans la littérature. Des architectures matérielles ont également été proposées pour accélérer certaines de ces techniques. Les architectures existantes manquent de performances et de flexibilité de programmation, notamment pour les images de haute résolution. Dans cette thèse, nous nous sommes intéressés à deux techniques de segmentation du réseau vasculaire rétinien, la technique du filtrage adapté et la technique des opérateurs de ligne. La technique de filtrage adapté a été ciblée principalement en raison de sa popularité. Pour cette technique, nous avons proposé deux architectures différentes, une architecture matérielle personnalisée mise en œuvre sur FPGA et une architecture basée sur un ASIP. L'architecture matérielle personnalisée a été optimisée en termes de surface et de débit de traitement pour obtenir des performances supérieures par rapport aux implémentations existantes dans la littérature. Cette implémentation est plus efficace que toutes les implémentations existantes en termes de débit. Pour l'architecture basée sur un processeur à jeu d’instructions spécialisé (*Application-Specific Instruction-set Processor* – ASIP), nous avons identifié deux goulets d'étranglement liés à l'accès aux données et à la complexité des calculs de l'algorithme. Nous avons conçu des instructions spécifiques ajoutées au chemin de données du processeur. L'ASIP a été rendu 7.7 × plus rapide par rapport à son architecture de base.

La deuxième technique pour la segmentation des vaisseaux sanguins est l'algorithme détecteur de ligne multi-échelle (*Multi-Scale Ligne Detector* – MSLD). L'algorithme MSLD est choisi en raison de ses performances et de son potentiel à détecter les petits vaisseaux sanguins. Cependant, l'algorithme fonctionne en multi-échelle, ce qui rend l’algorithme gourmand en mémoire. Pour résoudre ce problème et permettre l'accélération de son exécution, nous avons proposé un algorithme efficace en terme de mémoire, conçu et implémenté sur FPGA. L'architecture proposée a réduit de façon drastique les exigences de l’algorithme en terme de mémoire en réutilisant les calculs et la co-conception logicielle/matérielle.

Les deux architectures matérielles proposées pour la segmentation du réseau vasculaire rétinien ont été rendues flexibles pour pouvoir traiter des images de basse et de haute résolution. Ceci a été réalisé par le développement d'un compilateur spécifique capable de générer une description HDL de bas niveau de l'algorithme à partir d'un ensemble de paramètres. Le compilateur nous a permis d’optimiser les performances et le temps de développement. Dans cette thèse, nous avons introduit deux architectures qui sont, au meilleur de nos connaissances, les seules capables de traiter des images à la fois de basse et de haute résolution.

ABSTRACT

Millions of people all around the world are affected by diabetes. Several ocular complications such as diabetic retinopathy are caused by diabetes, which can lead to irreversible vision loss or even blindness if not treated. Regular comprehensive eye exams by eye doctors are required to detect the diseases at earlier stages and permit their treatment. As a preventing solution, a screening protocol involving the use of digital fundus images was adopted. This allows eye doctors to monitor changes in the retina to detect any presence of eye disease. This solution made regular examinations widely available, even to populations in remote and underserved areas. With the resulting large amount of retinal images, automated techniques to process them are required. Automated eye detection techniques are largely addressed by the research community, and now they reached a high level of maturity, which allows the deployment of telemedicine solutions.

In this thesis, we are addressing the problem of processing a high volume of retinal images in a reasonable time. This is mandatory to allow the practical use of the developed techniques in a clinical context. In this thesis, we focus on two steps of the retinal image pipeline. The first step is the retinal image quality assessment. The second step is the retinal blood vessel segmentation.

The evaluation of the quality of the retinal images after acquisition is a primary task for the proper functioning of any automated retinal image processing system. The role of this step is to classify the acquired images according to their quality, which will allow an automated system to request a new acquisition in case of poor quality image. Several algorithms to evaluate the quality of retinal images were proposed in the literature. However, even if the acceleration of this task is required, especially to allow the creation of mobile systems for capturing retinal images, this task has not yet been addressed in the literature. In this thesis, we target an algorithm that computes image features to allow their classification to bad, medium or good quality. We identified the computation of image features as a repetitive task that necessitates acceleration. We were particularly interested in accelerating the Run-Length Matrix (RLM) algorithm. We proposed a first fully software implementation in the form of an embedded system based on Xilinx's Zynq technology. To accelerate the features computation, we designed a co-processor able to compute the features in parallel, implemented on the programmable logic of the Zynq FPGA. We achieved an acceleration of 30.1× over its software implementation for the features computation part of the RLM algorithm.

Retinal blood vessel segmentation is a key task in the pipeline of retinal image processing. Blood vessels and their characteristics are good indicators of retina health. In addition, their segmentation can also help to segment the red lesions, indicators of diabetic retinopathy. Several techniques have been proposed in the literature to segment retinal blood vessels. Hardware architectures have also been proposed to accelerate blood vessel segmentation. The existing architectures lack in terms of performance and programming flexibility, especially for high resolution images. In this thesis, we targeted two techniques, matched filtering and line operators. The matched filtering technique was targeted mainly because of its popularity. For this technique, we proposed two different architectures, a custom hardware architecture implemented on FPGA, and an Application Specific Instruction-set Processor (ASIP) based architecture. The custom hardware architecture area and timing were optimized to achieve higher performances in comparison to existing implementations. Our custom hardware implementation outperforms all existing implementations in terms of throughput. For the ASIP based architecture, we identified two bottlenecks related to data access and computation intensity of the algorithm. We designed two specific instructions added to the processor datapath. The ASIP was made 7.7× more efficient in terms of execution time compared to its basic architecture.

The second technique for blood vessel segmentation is the Multi-Scale Line Detector (MSLD) algorithm. The MSLD algorithm is selected because of its performance and its potential to detect small blood vessels. However, the algorithm works at multiple scales which makes it memory intensive. To solve this problem and allow the acceleration of its execution, we proposed a memory-efficient algorithm designed and implemented on FPGA. The proposed architecture reduces drastically the memory requirements of the algorithm by reusing the computations and SW/HW co-design.

The two hardware architectures proposed for retinal blood vessel segmentation were made flexible to be able to process low and high resolution images. This was achieved by the development of a specific compiler able to generate low-level HDL descriptions of the algorithm from a set of the algorithm parameters. The compiler enabled us to optimize performance and development time. In this thesis, we introduce two novel architectures which are, to the best of our knowledge, the only ones able to process both low and high resolution images.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

ACC	Accuracy
ALU	Arithmetic Logic Unit
ASIC	Application-Specific Integrated Circuit
ASIP	Application-Specific Instruction-set Processor
AUC	Area Under the Curve
AXI	Advanced Extensible Interface
Brosters	Brothers and sisters
CPU	Central Processing Unit
DMA	Direct Memory Access
DSP	Digital Signal Processor
FFT	Fast Fourier Transform
FIFO	First In First Out
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
GOPs	Giga Operations Per second
GPP	General Purpose Processor
GPU	Graphics Processing Unit
HDL	Hardware Description Language
HLS	High Level Synthesis
LUT	Look Up Table
MSLD	Multi-Scale Line Detector
PL	Programmable Logic
PS	Processing System

RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
RLM	Run Length Matrix
ROI	Region Of Interest
RTL	Register Transfer Level
SE	Sensitivity
SIMD	Single Instruction Multiple Data
SoC	System on Chip
SP	Specificity
TIE	Tensilica Instruction Extension
VLIW	Very Long Instruction Word

CHAPTER 1 INTRODUCTION

1.1 Overview and motivation

An estimated 285 million people worldwide are affected by diabetes [1]. With a further 7 million people developing diabetes each year, this number is expected to reach 439 million by 2030. Today, more than 2 million Canadians live with diabetes. According to data from the National Public Health Institute of Quebec, 376,000 people had diabetes in 2003-2004 in the province. The most serious ocular complication of this disease is diabetic retinopathy, a leading cause of blindness and partial sightedness in Canadians under the age of 50. Diabetic retinopathy is an eye condition where elevated blood sugar levels cause blood vessels in the eye to swell and leak in the retina. Without treatment, the condition can result in irreversible vision loss or even blindness. With regular, comprehensive eye exams by an eye doctor, diabetic retinopathy can be detected early and treated.

Blindness and visual loss can be prevented and avoided through early detection using digital fundus imaging. This approach involves taking color images of the retina, which allows eye doctors to monitor changes in the retina and to detect the presence of diabetic retinopathy. This solution has been proposed to make regular examination widely available, even to population in remote and underserved areas. Thus, millions of retinal images would potentially require evaluation in Canada and in the world (retinal images for all people with diabetes, and at least two images per eye).

The current challenge is to make early detection using digital fundus imaging more accessible by reducing the cost and manpower required with an improved detection accuracy. Automated detection is a solution to meet this challenge. Most early detection programs use fundus images examined by human experts to detect the presence of specific lesions (microaneurysms, haemorrhages, exudates and cotton-wool spots) indicative of diabetic retinopathy. If such abnormalities are found −typically in 10% of the examined images− the patient is referred to an ophthalmologist or retinal specialist.

The major weakness of the traditional solution is the human expert, who is expensive to train. Evaluating hundreds if not thousands of images per day is a tedious task, and most images have no abnormalities. In addition, the delay between taking the images and the result of the reading can be very long. This makes it impractical to inform the patients of the test result immediately at the point of service. Thus, using an automated system to analyse retinal image abnormalities and to assist human experts to detect diabetic retinopathy is a very attractive approach that addresses the weaknesses of the traditional one.

In 1987, Baudoin et al. [2] were the first to propose an image analysis method for microaneurysms detection, and since, several algorithms and approaches have been developed to detect other disease indicators (microaneurysms, haemorrhages …). To assist the development of more accurate detection algorithms and to determine how these different algorithms rank in terms of performance, publicly available annotated image databases have been established, and quality metrics have been proposed [3] to serve in algorithms comparison. Recent works have proved that automated detection of early diabetic eye disease can be as good as highly trained human experts [4], they have proved also that the existing algorithms are mature, and significant improvements in performance are difficult to achieve [3].

Several companies have started commercializing solutions to help and assist ophthalmologists. With the aim to make eye-health care accessible in low-income countries, Peek Vision developed the Portable Eye Examination Kit (Peek). It’s a clip-on hardware adapter for smartphones with an application. Peek is designed to be affordable and portable, and to be used by non-experts with minimal training to carry out eye-health checks. CARA of Diagnos Inc is also one of the commercialized solutions. CARA is a software solution and a Tele-ophthalmology platform that uses enhanced digital images to support the early detection of diabetic retinopathy. Statistics show that 50% of people with diabetes in the USA do not have access to any form of regular examination [3]. Tele-ophthalmology platform are a good alternative to solve the care-accessibility problem. Another advantage, analysis of cost-effectiveness automated detection systems showed that their cost is lower when compared to the manual methods [3].

Tele-ophthalmology system as novel tool takes advantage of technology to offer more opportunities for people with diabetes, and to provide to the community outreach better services. A tele-ophthalmology platform is based on a centralized processing system. Figure 1-1 shows the principle of a centralized processing system. This system would receive retinal images to be processed via the Internet. The images could be sent from hospitals, clinics, and/or points-of-care. After processing, the centralized system would return back the examination results to the source.

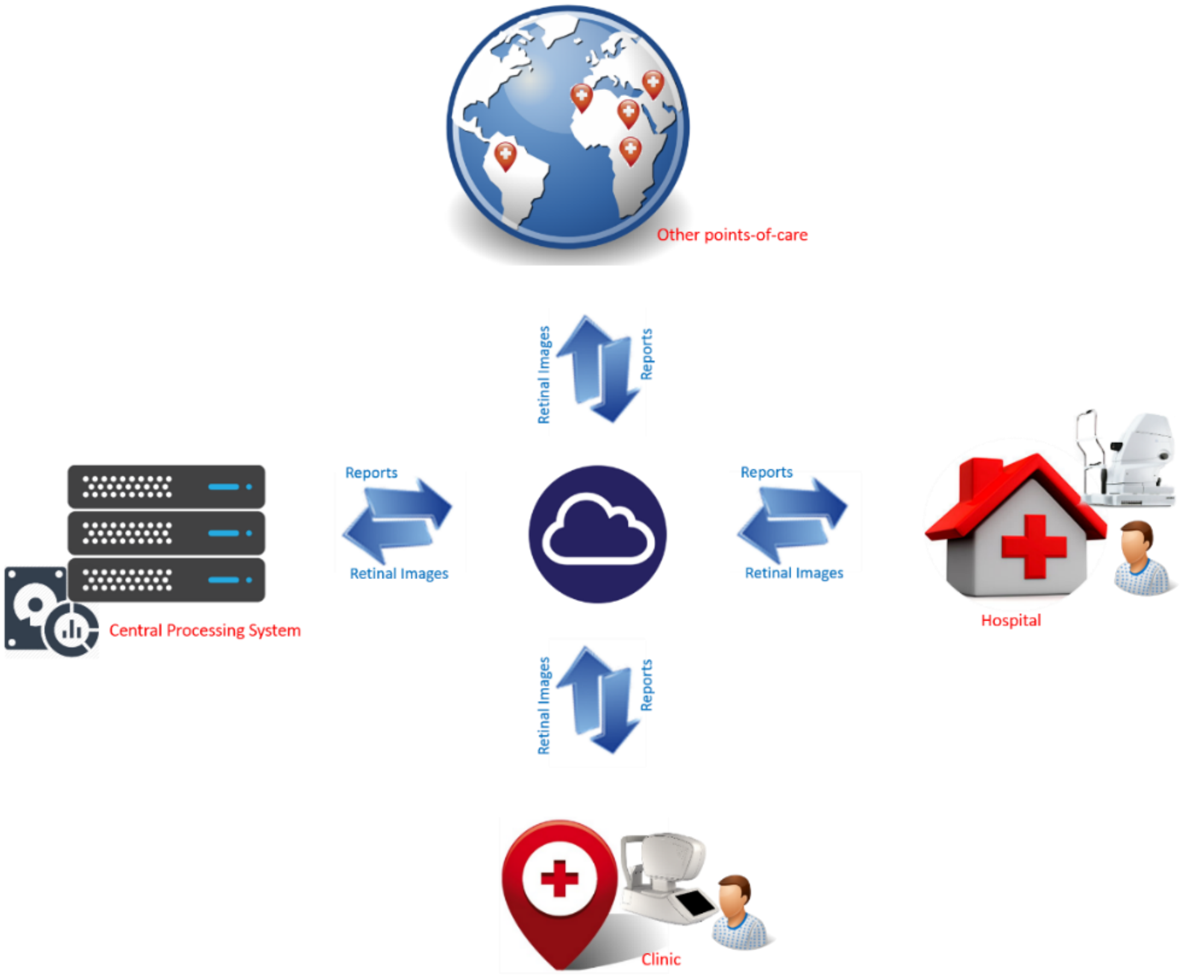


Figure 1-1. Centralized processing system overview

We can estimate that in the USA, if all underserved populations were to be provided with digital imaging, 32 million retinal images would require evaluation annually. This number assumes that 40% of the population affected by diabetes requires fundus images examination and that two images per eye are taken [3]. In other regions in the world, a greater number of images would require evaluation.

Existing algorithms for automated detection of diabetic retinopathy require several minutes to process images of public databases when using general processors. Unfortunately, the resolution of these public databases images is much smaller than the one provided by recent retinal cameras (768 × 584 pixels images). Recent cameras provide high resolution images (4288 × 2848 pixels images), which implies a higher processing time to obtain a diagnostic result.

1.2 Problem statement

In this thesis, we are addressing the problem of processing, in a reasonable time in the context of telemedicine, the large number of retinal images expected to be collected from a centralized processing system. This will require efficient processing systems. Moreover, accelerating the processing will allow the realization of embedded processing systems that can be integrated with the retinal camera to allow on-site diagnostic. Hardware platforms (FPGAs or ASICs) are very adequate for image processing algorithms acceleration. The aim of the proposed research is to introduce suitable hardware architectures to accelerate the retinal image algorithms, which presents several important challenges.

First, we consider the problem of retinal images quality assessment. This task is necessary for automated detection systems since it represents the first task of retinal image processing pipeline. This task is responsible of identifying images of good quality after acquisition and reject images of poor quality. This task require the computation of several image features which include complex algorithms difficult to parallelize by their nature. Accelerating such algorithms is very useful and favorable, which will allow the creation of mobile retinal capturing systems. Many approaches and algorithms for retinal image quality assessment have been proposed in the literature. However, there are no existing hardware implementations in this context.

Secondly, we select the problem of retinal blood vessel segmentation. This is a key and complex task in the process of eye disease detection. This task gains its importance because vessel features are good indicators of various pathologies. Detecting the blood vessels allows the evaluation of these features that subsequently allows to detect the pathologies. Indeed, in CAD systems we usually remove segmented vessels and consider other structures as lesions. Selecting the adequate algorithm is very important to get better results in terms of blood vessel detection quality.

A third problem is to propose adequate architectures for the targeted algorithms. The complexity of the architecture is very dependent to the algorithm complexity. Thus, selecting hardware friendly algorithms or making them hardware friendly is very important. A hardware friendly algorithm is an algorithm that can take advantage of the hardware platform structure. It can be parallelized easily, and that can avoid complex computations. In other words, making an algorithm hardware friendly is to make its conception suitable for hardware implementation. We can achieve this goal by proposing several simplifications and modifications of the existing algorithms while maintaining a good compromise between simplicity and accuracy. This should take in consideration the algorithm requirements in terms of data access and functional units.

Furthermore, existing retinal image datasets include images of low-resolution while recent advances in retinal imaging and cameras technology promote the use of image of higher resolution. Proposed hardware architectures should take this in consideration to solve the problem of architecture scalability. The same architecture should be able to scale according to the image resolution with less development effort and minor modifications. Generally, optimized hardware architectures require the use of very-low level programming models, which increases the amount of needed efforts. A task such as blood vessel segmentation require high computations precision to achieve the best algorithm accuracy in terms of blood vessel detection quality. Functional units and their optimizations are not easy to scale, which represent an additional challenge.

1.3 Research contributions

The main objective of this research is to design and implement retinal image processing algorithms on hardware platforms with the aim to achieve high throughput with a certain level of design flexibility. This section reviews the contributions of the different parts of this thesis.

We started our work by analyzing the context of retinal image processing and their hardware implementations in order to identify opportunities for acceleration and efficient implementation. We identified two main tasks to be addressed. The first task was the retinal image quality assessment. We thus proposed an embedded system to compute the RLM features for retinal images. The RLM features provide quantitative information describing textural properties of retinal images, which eventually allows to differentiate images according to their quality. Our first solution was completely software. In order to accelerate the execution of the algorithm, we proposed a hardware co-processor to be implemented on the programmable fabric of Zynq FPGA. We designed, implemented and tested the system with the co-processor. Our results show improvements over the software implementation. This contribution was published in a conference paper entitled "A run-length encoding co-processor for retinal image texture analysis," presented in the International Conference on ReConFigurable Computing and FPGAs (ReConFig) in 2015 [5].

The second task identified to be accelerated was the retinal image blood vessel segmentation. To that effect, we targeted two different techniques. The first technique is based on matched filtering. There exist several hardware implementations of this algorithm, however there are several limitations in terms of performances and programming flexibility. In this thesis, we designed a scalable and optimized architecture of the matched filter. The adopted low-level programming model with several optimizations allowed us to achieve better performances in terms of execution time in comparison with existing implementations. To overcome the problem of programming flexibility and to keep low level optimizations, we proposed a tool able to generate automatically a low-level hardware description of the algorithm based on a set of its parameters. With the aim of comparing two implementation strategies, we proposed a novel architecture based on an ASIP. This second architecture was improved by designing two specific instructions integrated into the Xtensa processor datapath, which allowed the improvement of the ASIP performances when executing the matched filter algorithm. These contributions were published first in a conference paper entitled "A scalable hardware architecture for retinal blood vessel in high resolution fundus images," presented in the Conference on Design & Architectures for Signal & Image Processing in 2014 [6], and in an extended version in a paper entitled "Flexible architectures for retinal blood vessel segmentation in high-resolution fundus images," published in the Journal of Real-Time Image Processing in 2016 [7].

The second targeted technique for retinal blood vessel segmentation is based on the MSLD algorithm. In this thesis, we introduced an optimized memory-efficient architecture for the MSLD algorithm. This architecture was designed and implemented in a Zynq FPGA. It benefits from HW/SW co-design to satisfy the memory requirements of the algorithm. Results in terms of blood vessel segmentation accuracy are shown in addition to acceleration results over CPU and GPU implementations. To be able to target low and high resolution images, the automatic generation of low-level HDL description of the algorithm was made possible by the development of a tool that takes as input the algorithm parameters. In addition, we provide also a full comparison table of our implementations with existing ones. These contributions were published first in a conference paper entitled "Memory Efficient Multi-Scale Line Detector Architecture for Retinal Blood Vessel Segmentation" presented in the Conference on Design & Architectures for Signal & Image Processing in 2016 [8], and have been submitted in an extended version to the IEEE Transactions on Circuits and Systems for Video Technology in a paper entitled "Memory Efficient Flexible Architecture for Retinal Blood Vessel Segmentation Using a Multi-Scale Line Detector," Jan. 2017 [9].

1.4 Thesis organization

This thesis is divided into 6 chapters. Chapter 2 reviews the important background material and related works that are used in this thesis. In Chapter 3, we present an implementation of a co-processor to accelerate a feature extraction algorithm. In chapter 4, we present flexible architectures for matched filtering technique for retinal blood vessel segmentation. Chapter 5 presents a memory-efficient architecture for a multi-scale algorithm for retinal blood vessel segmentation. Chapter 6 presents a general discussion where we present the novelty of our work and its limitations. Chapter 7 concludes the thesis by summarizing our contributions and outlining our recommendations and future research directions

CHAPTER 2 LITERATURE REVIEW

In this chapter, we examine the relevant literature for our research subject. Two topics which are directly related to our research were selected: algorithmic aspects of the automated detection of diabetic retinopathy and their hardware architectures and implementations. For the automated detection of diabetic retinopathy, we start by describing the eye anatomy and the manifestations of eye diseases. Then, we present the different parts of an automated detection system using digital fundus images and the existing approaches. For the second topic, we review the proposed hardware architectures and implementations of retinal image processing algorithms. High level synthesis of image processing algorithms is also presented.

Accordingly, this chapter is organized as follows. Section 2.1 introduces the diabetic retinopathy disease and its symptoms. In section 2.3, the retinal image processing pipeline and its hardware implementations are presented and described. Section 2.2 presents the public databases and metrics established by the scientific community to evaluate the effectiveness and the advances in disease detection quality. Section 2.4 reviews the architectural considerations for the implementation of image processing algorithms. This include the data access, hardware function evaluation and processor architectures for image processing applications. Section 2.5 reviews the existing tools for HDL description generation for image processing algorithms. Finally, section 2.6 defines our research objectives.

2.1 Diabetic retinopathy

The human eye is the most powerful sensor for the perception of its environment. By analogy, it’s often compared to a camera, and the camera sensor can be seen analogous to the retina. The retina is the inner surface of the eye and consists of transparent tissue of several layers of cells designated to absorb and convert light into neural signals [10]. Once converted to neural signals and collected to the optic nerve in the optic disc, the impulses are transmitted to the brain. The nutritional support of the retina is provided by blood vessels. Figure 2-1 highlights the different anatomical parts of the retina (Macula, fovea, optic disc and the blood vessels).

In most cases, visual disorders are consequences of vascular changes that diabete causes to the eye. Neovascular glaucoma, diabetic neuropathies, cataract and diabetic retinopathy are the most common diabetic eye diseases. In this section and in the rest of this proposal we will concentrate on diabetic retinopathy.

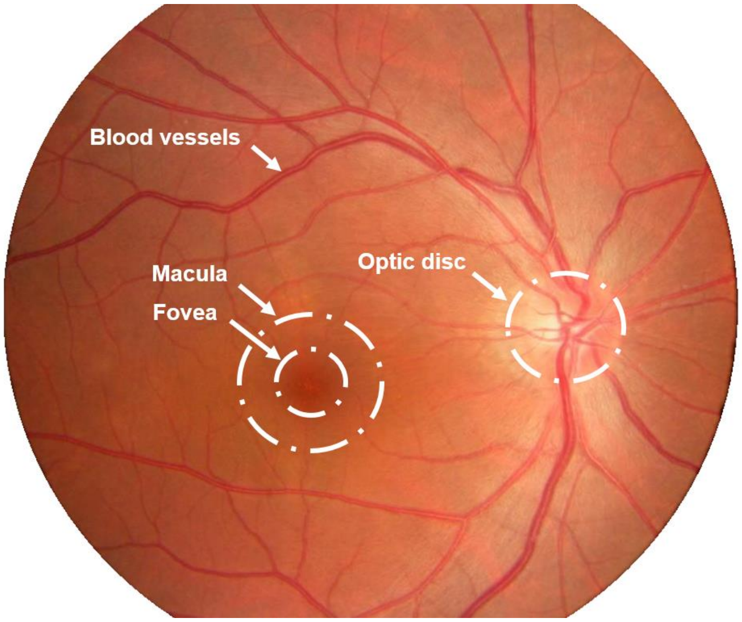
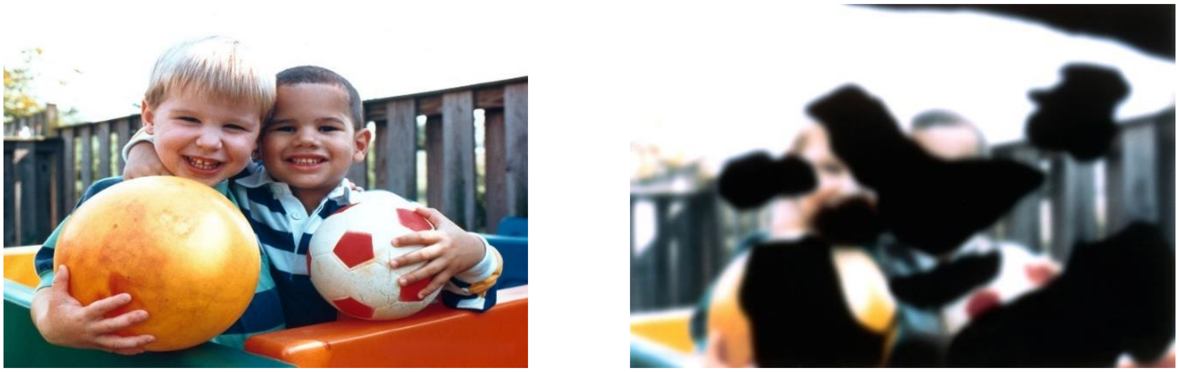


Figure 2-1. Normal retina anatomical parts

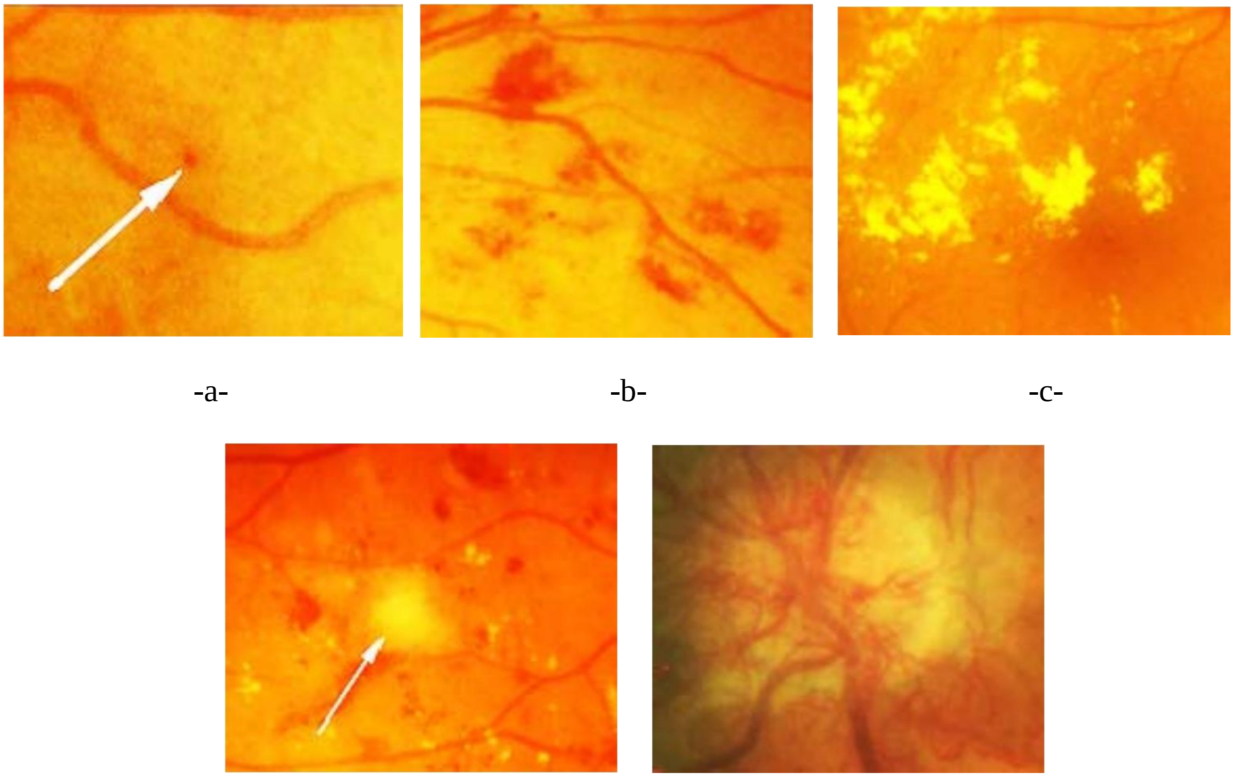
Diabetic retinopathy is the most prevalent ocular complication of diabetes, and a leading cause of blindness in American adults [3]. It occurs when high blood sugar levels provoke the abnormal growth of blood vessels. These new blood vessels are weak. They can swell and leak fluid or even close off completely and damage the cells of the retina. A healthy retina is necessary for good vision. Patients with diabetic retinopathy first start noticing changes in vision. But over time, diabetic retinopathy can get worse and cause vision loss. Usually, diabetic retinopathy affects both eyes. Figure 2-2 shows the difference between the normal vision and vision with diabetic retinopathy.

The detection of diabetic retinopathy is possible by detecting its symptoms in fundus images. These symptoms are: microaneurysms, haemorrhages, hard exudates, soft exudates and neovascularisation. Microaneurysms appear as small red dots in the retina. Due to blood vessel damage, small blood vessels may rupture and cause intraretinal haemorrhages that appear either as small red spots or larger round-shaped blots. The hard exudates are lipid formations leaking from the weakened blood vessels and appear yellowish with well-defined borders. Soft exudates are small microinfarcts pale areas with diffuse borders. Neovascularisation is the growth of new fragile vessels due to the extensive lack of oxygen and obstructed capillary in the retina [11]. The different cited symptoms are shown in Figure 2-3.



-a- -b-

Figure 2-2. a- Normal vision, b- Vision with diabetic retinopathy [12]



-a- -b- -c-

-d- -e-

Figure 2-3. Diabetic retinopathy symptoms: a- Microaneurysms, b- Haemorrhages, c- Hard exudates, d- Soft exudates, e- Neovascularisation [11].

2.2 Retinal image processing pipeline

In any complete automated system for eye diseases detection, four steps are necessary as shown in Figure 2-4. These four steps are: retinal images quality assessment, retinal images enhancement, features extraction (blood vessels, lesions, microaneurysms …) and disease detection.

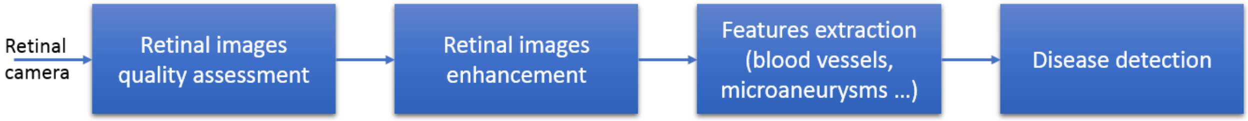

Retinal camera
Retinal images quality assessment
Retinal images enhancement
Features extraction (blood vessels, microaneurysms ...)
Disease detection

Figure 2-4. Main steps of an automated eye disease detection system

2.2.1 Quality assessment

Image quality evaluation is a common problem in image processing systems. For medical imaging and especially diabetic retinopathy detection, sufficient image quality is necessary for further processing to ensure reliable diagnosis. The quality determines the ability of a human expert or an automated system to correctly detect disease symptoms from the image. When the image quality is not sufficient, it becomes difficult or impossible to make a reliable clinical judgment on the presence or absence of any eye disease. Several published works report that 10% of the acquired mydriatic (pupil dilation) images are rejected [13] due to insufficient quality. For single field non-mydriatic (no pupil dilation), the rejection rate can be up to 20.8% [14]. Assessing the retinal image quality represents an important limiting factor for automated diabetic retinopathy detection. To avoid the costs associated with processing useless images, which must be replaced in a second patient sitting, the research community started to develop vision-based solutions to assess the quality of the retinal images [15, 16]. The quality of the acquired retinal image should be assessed as a first step, and this task should be executed immediately after the acquisition of the images. Retinal fundus images quality assessment algorithms can be grouped in three different categories: histogram based methods, retina morphology methods and “bag-of-words” methods [17].

In histogram-based methods, the quality of a given image is determined through the difference between its histogram of certain features and the mean histogram of a set of good-quality images as reference. Based on this, Lee and Wang [18] were the first to address the problem of automatic evaluation of fundus image quality. They employed the global histogram of the image intensities approximated by a Gaussian distribution. This approach was extended by Lalonde et al.

[19] using two different sets of features: the edge magnitudes distribution and the local pixel intensity distribution.

In retina morphology methods, features describing the retinal structure are used to evaluate their quality. A vessel segmentation algorithm was used by Usher et al. [20] to estimate the image blurring. This algorithm uses the area of the detected vessels as a specific feature for the retinal fundus images. Fleming et al. [15] presented a method that evaluates the image clarity using vessel area in the macula region, and the field definition using the relative position of the fovea and the length of the main vessel arcades. The image clarity and field definition are finally combined to generate a global quality metric.

In “bag-of-words” methods, a pattern recognition classifier (Support Vector Machine, Naïve Bayes, etc.) is employed to classify the occurrence of some common words automatically generated from the raw features in the test set. Niemeijer et al. [21] employed this approach to evaluate the fundus image quality. The authors use the color and second order statistics of the image as two sets of raw features to provide a compact representation of the structures found in an image. A different set of features was employed by Paulus et al. [22] : the pixel gray levels and the Haralick texture features.

The quality assessment of retinal images is still an open topic for research. There are several algorithmic challenges to be addressed such as the creation of public datasets with images of different quality. Without public datasets, it would be difficult to classify the algorithms based on their performances. Even if the quality assessment is the most critical step in terms of execution time since it’s the first task of the pipeline, the hardware implementation and acceleration of its algorithms in the context of the retina have not been addressed yet.

2.2.2 Retinal image enhancement

Acquired images through cameras may be contaminated by a variety of noise sources (e.g. photon or on chip electronic noise), distortions, shading or improper illumination. Eventually, the image quality is reduced, which affects the performance and efficiency of any subsequent processing algorithms. A pre-processing and enhancement step is necessary. Image enhancement algorithms consist of a collection of techniques which aim to improve the visual appearance of an image.

For retinal fundus image systems, image enhancement is the second step for an automated detection system. In the retina context, the aim is to reduce the noise, to correct the non-uniform illumination and to improve the contrast. The illumination can be non-uniform in retinal images due to the variation of the retina response and the imaging systems non-uniformity. To correct the non-uniform illumination, Yang et al. [23] proposed to divide the image by an over-smoothed version of it using a spatially large median filter. Many authors employ standard local or global histogram equalization techniques as Sopharak et al. [24].

In an RGB fundus image, the green channel is always considered the best to exhibit a good contrast of the structures of interest according to the background. Hence, it is a common practice in the literature to use the green channel for segmentation purposes. This has a major advantage, instead of processing the three channels, the calculation efforts are reduced to the processing of the green channel.

The background subtraction technique is one of the popular techniques for image enhancement in the literature. This technique is based on the estimation of the retina background with a large median filter applied on the green channel of the image [25]. This technique was refined by Cree et al. [26] to reduce the inter-patient color variability assuming that the background-less fundus image has colors normally distributed. The color of the new image is equalized to a reference one instead of simple histogram equalization. A hardware implementation of the background technique can be parallelized to improve the execution of the algorithm.

The contrast enhancement is a popular technique in biomedical image processing. This technique is very effective in making interesting silent parts more visible. Walter et al. [27] proposed to enhance the contrast of fundus images by applying a gray level transformation to the original grayscale image. Contrast limited adaptive histogram equalization (CLAHE) [28] is very popular. In this technique, the image is split into disjoint regions. In each region a local histogram equalization is applied. The boundaries between the regions are eliminated with a bilinear interpolation.

Several hardware architectures have been proposed to implement image enhancement algorithms. Tsutsui et al.[29] proposed hardware architecture for real-time Retinex video image enhancement. In order to efficiently reduce the enormous computational cost required for high resolution image enhancement, processing layers and repeat counts of iterations are determined according to a software evaluation result. The implemented architecture is able to support WUXGA

(1920×1200) at 60 fps. Hanumantharaju et al. [30] proposed hardware architecture for a new algorithm for adaptive color image enhancement based on Hue-saturation value (HSV) color space. This algorithm uses a saturation feedback to enhance contrast and luminance of the color image, while the saturation component is enhanced by stretching its dynamic range to get rich color display. This algorithm was implemented on Xilinx Virtex II FPGA.

Bo-Hyun et al. [31] proposed a new algorithm for image enhancement with less computational complexity. This new algorithm consists of a decimation filter, contrast enhancement and color enhancement blocks. Simulation and experimental results show better performances with lower complexity when compared to conventional algorithms. Zhang et al. [32] proposed digital color enhancement architecture based on reflectance/illumination model. The approach uses the approximation techniques for efficient estimation of log2 and its inverse to promote the log-domain computation and to eliminate multiplications, divisions and exponentiations. With effective color space conversion, the HSV-domain image enhancement architecture is able to achieve a throughput rate of 182.65 (MOPS) on Xilinx Virtex II FPGA at a clock frequency of 182.65MHz.

Image filtering is an essential part of image enhancement techniques and plays an important role in image processing. Azizabadi et al. [33] proposed hardware architectures for image filters including Gaussian, median and weighted median filters. The aim of the implemented architecture is to optimize the speed and the area. The proposed architectures are implemented and synthesized in ASIC with 65 nm technology. The authors report a maximum frequency of 1 GHz for median filter and 666.67 MHz for both Gaussian filter and weighted median filter. An FPGA implementation of median and weighted median filter for image processing was presented in [34] by Fahmy et al. The input samples are first used to construct a cumulative histogram, which is used to find the median value. The resource usage of the design is kept independent of the window size, but dependant on the number of bits of the data samples, which offers an efficient implementation of large-windowed median filtering. This method was extended to the weighted median filter. Reza et al. [35] proposed a suitable architecture for high speed VLSI and FPGA of the contrast limited adaptive histogram equalization (CLAHE) algorithm. The goal of this implementation is to minimise the latency without sacrificing precision. The maximum latency encountered in this approach is about half frame.

2.2.3 Features extraction

One goal of computer image processing is the efficient and effective visual features extraction. Deciding which visual features to extract and choosing the best way to extract them are crucial problems in many image processing tasks. Retinal image features extraction is a main step in many automated detection system. In this step, the aim is to detect retina anatomy and disease symptoms. Extracting retina anatomy facilitates the task of identifying disease symptoms. In the literature, many works target the retina anatomy, this includes the optical disk, fovea and blood vessels. Disease symptoms are also targeted, such as microaneurysms, lesions and exudates. In this section we will review two of the most important features of retinal images, blood vessels and microaneurysms.

2.2.3.1 Blood vessel extraction

The ability to distinguish retinal blood vessels from other structures is a step of great importance for diabetic retinopathy detection and in many retinal imaging applications. For diabetic retinopathy detection, the aim is to extract the retinal blood vessels to remove them and to identify red lesion candidates which are symptoms of diabetic retinopathy. A large number of algorithms have been published relating to the detection of retinal blood vessels. A complete review of the existing methods for retinal blood vessels detection can be found at [36]. There exist other methods based on tracking techniques [37] and supervised methods [38, 39] to deal with the retinal blood vessel detection problem. In this section we will present some existing algorithms and their hardware implementations.

Matched filtering is a popular technique for vessels detection in retinal images. It is based on the convolution of a 2-D kernel with the retinal image. The kernel is designed to model a feature based on its properties in the image at some unknown position and orientation. The matched filter response indicates the presence of the feature. For retinal images, the properties of the blood vessels are exploited to design the matched filter kernels. Three main properties are usually used: blood vessels can be approximated by a piece-wise linear segments since they have a limited curvature, the diameter of the vessels decrease as they move radially outward from the optic disc, the intensity profile of the cross section of the vessel approximates a Gaussian curve. The matched filtering technique can benefit greatly from a parallelized hardware implementation.

Chaudhuri et al. [40] were the first to propose a two-dimensional linear kernel with a Gaussian profile to segment the retinal blood vessels. The kernel is rotated 12 times with an increment of 15° to fit into vessels of different orientations. The highest response of the filter in each pixel of the image is selected and is thresholded to provide a binary vessel tree image. Some problems of the proposed matched filter are noticed. The kernel may be quite large and needs to be applied at several orientations especially for high resolution images, which result in a computational overhead. The kernel responds optimally to vessels that have the same form as the proposed kernel, and may not respond to those vessels with a different profile. It's also possible that the matched filter respond to some present pathologies in the image, especially with the retinal background variations, which increases the number of false detections.

Al-Rawi et al. [41] proposed an improved matched filter to detect the retinal blood vessels. They used an optimization method based on exhaustive search to find the best parameters of the filter. Then they proposed an automated method to find the best threshold to segment the retinal blood vessels based on the number of connected components and Euler number. Zhang et al. [42] proposed to use the matched filter with the first-order derivative of a Gaussian to reduce the false detections produced by the original matched filter and to detect the missed fine vessels. Dalmau et al. [43] proposed to combine the matched filter with a segmentation strategy by using a Cellular Automata, while Zolfagharnasab et al. [44] proposed a new kernel function with Cauchy distribution to improve the accuracy of the retinal vessel detection. These works introduce new segmentation methods or propose new kernel functions for the blood vessel cross section intensity approximation to improve the original matched filter.

Mathematical morphology is also used for vessels detection. The basic morphology of the retinal blood vessels is known a priori to be comprised of connected linear segments [45]. Zana et al. [46] proposed a method based on the fact that vessels are piecewise linear and connected. Hence, mathematical morphological operators are used to differentiate the vessels from the background. Jiang et al. [47] propose to threshold the image at different levels by multi-threshold probing technique, and uses a verification procedure to detect the vessels in the segmented images. The final segmentation is obtained by a combination of those segmented images returned in each step [45]. Lam et al. [48] proposed to employ the normalized gradient vector field to detect the centerlines after the use of the gradient vector field to detect vessel like objects. To reduce the falsely detected vessels, the authors proposed to use a pruning step to remove all vessel pixels that are far away from the centerlines. The same authors proposed a new method in [49]. This method deals with the bright and dark lesions. The authors proposed the use of three different concavity measures to detect the vessels and to distinguish them from the lesions.

Supervised methods have been widely proposed because of their performance [50-53]. A weakness of supervised methods is that they depend on the training set. Thus, in a system where images of different resolutions must be processed, the supervised methods must be trained with a set of representative images of each resolution. In addition, these methods are generally implemented on CPUs or GPUs because any update to the classifier may require a new implementation and major modifications, and are thus not suitable for custom processor implementations unless great efforts are paid.

Line operators detectors are an unsupervised method used to segment retinal blood vessels. These methods were previously used in mammography, were introduced by Ricci et al. [52] as a feature extractor with a classifier for retinal blood vessels detection [52]. Multi-scale line operators were also introduced and used in [54-56]. Nguyen et al. [45] proposed the MSLD algorithm for retinal blood vessel detection. By changing the length of a basic line detector, line detectors of varying scales are achieved, and their responses at different scales are linearly combined to produce a final segmentation. Multi-scale operation is achieved by changing the length of a basic line detector, which is more suitable for hardware implementation than Gaussian pyramids as in [55]. Contrary to supervised methods, a single parameter change makes this algorithm suitable for images of different resolutions.

Three main implementation strategies are followed to implement blood vessel segmentation algorithms. These strategies are based on the use of CPUs, GPUs or custom processors in FPGAs. Most of the existing approaches are implemented in software in CPUs [45-49]. Palomera-Perez et al. [44] proposed a parallel multiscale feature extraction and region growing algorithm applied for high-resolution images. The algorithm is implemented on 14 parallel CPUs. The proposed approach is able to process images of 2890 × 2308 pixels 9× faster in comparison with a serial implementation with 8% less accuracy. Becker et al. [57] proposed an approach that both maps and localizes retinal blood vessels in real-time on a video targeting an intraocular surgery application. The algorithm was implemented in an Intel i7 CPU. It processes 30-40 frames per second with an image resolution of 400 × 304. The presented CPU implementations give satisfactory results in terms of segmentation quality but they are computationally intensive due to their high complexity and they are too slow to achieve onsite, real-time processing during the patient's visit. While CPUs present the most flexible development and implementation process, they still do not have enough computation power to deal with large number of high-resolution images.

Many GPU implementations are also presented in the literature. Krause et al. [58] implemented a local Radon transform based algorithm on GPU. This implementation is able to process images of size 4288 × 2848 pixels in 1.2 s on an NVIDIA Geforce GTX680. Argüello et al. [59] proposed a hybrid strategy based on global image filtering and contour tracing for retinal blood vessel extraction. Its GPU implementation processes images of the DRIVE database in an average of 0.014 s. For high-resolution images of 4288 × 2848 pixels, the execution time is 0.753 s. Savarimuthu et al. implemented a matched filter for blood vessel detection in human forearms [60]. When implemented in GPU, the algorithm is able to process 44 images of 640 × 480 pixels each second. This implementation does not consider high-resolution images.

Several features extraction algorithms in general are implemented in FPGA such as in [61] [62] [63] [64]. Many others are proposed for retinal blood vessel segmentation. Savarimuthu et al. [60] have also implemented their algorithm as a custom processor in FPGA. For the same image size, this implementation is able to process 215 images per second. In [65], an FPGA implementation of a programmable SIMD architecture for vessel tree extraction in retinal images was presented. The implemented algorithm is based on an active contour technique called Pixel-Level Snakes (PLS) and morphological operations. This implementation requires a total of 1.349 s to extract the vessel tree from 768 × 584 pixels images. Koukounis et al. [66] presented a hardware architecture implemented in FPGA for retinal vessel segmentation targeting portable embedded systems. The implemented architecture processes images of 768 × 584 pixels in 0.0523 s. Alonso-Montes et al. [67] presented a hardware approach for an authentication system based on the retinal blood vessels extraction using local dynamic convolutions and morphological operations. This approach was implemented and tested in a fine-grained single instruction multiple data (SIMD) processor array. The execution time required to process a 768 × 584 pixels image is 0.1925 s excluding the I/O operations. These implementations only consider images of low-resolution.

The previous cited implementations achieve significant improvements over software implementations in terms of execution time. Some of them are partially able to process low resolution images of public databases in a reasonable time. However, in a telemedicine context, more improvements are necessary especially when processing high-resolution images. This is especially true for hardware implementations where the memory size and bandwidth are real challenges [68]. CPU and GPU implementations are flexible for parameters tuning when changing the image resolution. However, custom architectures are specific to a given set of algorithm parameters and image resolution, and tend to be inappropriate in situations where images of different resolutions must be processed such as in telemedicine. This is true unless the problem of ease of hardware architecture customization is resolved.

2.2.3.2 Microaneurysms detection

Microaneurysms are one of the symptoms of diabetic retinopathy, and their detection is essential in the process of diabetic retinopathy detection and grading. Detecting the microaneurysms in fundus images is a very vivid field for the automatic detection of eye diseases community. The first automated approach for the segmentation of retinal microaneurysms was described by Baudoin et al. [2]. Since then, many approaches have been proposed in the literature. The two most relevant approaches are: Morphological approach and Region growing approach [69].

Morphological processing is a collection of techniques used for image component extraction. This approach is most commonly used for microaneurysms detection. Niemeijer et al.[70] used a hybrid approach to detect the red lesions by combining the prior works by Spencer et al. [25] and Frame et al. [71]. Spencer et al. used morphological processing to detect microaneurysms. A bilinear top-hat transformation and matched filtering were used to provide an initial segmentation. Frame et al. proposed a list of features to characterise microaneurysms. These features are used by a classifier to decide which candidates are microaneurysms. Kande et al. [72] used a matched filter for contrast enhancement of red lesions, the enhanced lesions are then segmented by relative entropy based thresholding. A top-hat transformation is applied to supress the enhanced blood vessels. The candidates red lesions are classified from other dark segments using a support vector machine. The linear filtering and thresholding are employed by Matei et al. [73] to identify the presence of specific retinal lesions like microaneurysms.

Spencer et al. [25] and Frame et al. [71] have used region growing algorithm to find final candidate object set after the detection of the retinal vessel tree. Cree et al. [74] proposed a method for microaneurysms detection based on region growing algorithm to find the underlying candidate morphology. Feature classification is used to distinguish microaneurysms from other spurious objects. Usher et al. [75] used region growing algorithm to detect the microaneurysms. After preprocessing, microaneurysms are extracted using recursive region growing and adaptive intensity thresholding. The classification of the detected lesions is done by neural network. Streeter et al. [76] proposed a microaneurysms detection algorithm in color images based on region growing. After preprocessing, the blood vessel tree are extracted and removed. Thresholding and region growing are applied by taking a candidate seed image, after the region growing, the features are extracted.

Neural networks are also used and considered because they are able to detect the regions that contain the microaneurysms and reject other regions. To achieve this goal, the image is divided into several windows. According to a multi-stage training, the presence of microaneurysms is detected in these windows by the neural network. Generally, neural networks, support vector machine (SVM) and naïve Bayes techniques are used for the classification of the detected candidates.

2.3 Public datasets and metrics

Automated detection systems can significantly decrease the manual labour in diagnosing large quantities of retinal fundus images. Several algorithms have been developed in the literature. Annotated data is necessary to test and evaluate algorithms. There are some publicly available annotated databases of retinal fundus images. These databases are different in goals, characteristics and completeness level. Their main goals are vessel segmentation, diabetic retinopathy and microaneurysms detection. Each public database provides a gold standard reference which is necessary for algorithm training and testing, also called Ground Truth data. The most known public databases are: ROC [77], DRIVE [51] and STARE [78].

The ROC (Retinopathy Online Challenge) is a microaneurysms dataset. This database is part of a multi-year online competition of microaneurysms detection that was arranged by the University of Iowa in 2009. This database contain 50 training images with available annotated images and 50 test images where the annotated images were withheld by the organizers. The images are JPEG compressed and were acquired using a Topcon NW100, a Topcon NW200 and Canon CR5-45NM non-mydriatic camera at 45° field of view. There are three different image sizes present in the database: 768 × 576, 1058 × 1061 and 1389 × 1383 pixels [77]. The DRIVE (Digital Retinal Images for Vessel Extraction) consist of a total of 40 color JPEG compressed fundus images. Between the 40 images, 7 contain pathology (exudates, hemorrhages and pigment epithelium changes). The images were acquired using a Canon CR5 non-mydriatic 3-CCD camera with 45° field of view. The images are 8 bits per color with a resolution of 768 × 584 pixels [51]. The STARE database contains 20 images for blood vessel segmentation. 10 among these images contain pathology. The images were captured using a Topcon TRV-50 fundus camera with 35° field of view. The resolution of the images is 605 × 700 pixels with 8 bits per color. All the images were segmented manually by two observers [78].

Metrics have also been developed to compare and evaluate algorithms. Algorithms can be evaluated against a ground truth image dataset using sensitivity, specificity and accuracy metrics. The sensitivity is a ratio between 0 and 1, which is the number of true-positives (TP) divided by the sum of the total number of false-negatives (FN) (incorrectly missed) and true-positives as shown in Equation (2-1). Specificity is also a ratio between 0 and 1, which is the number of true-negatives (TN) divided by the sum of the total number of false-positives (FP) (incorrectly thought to have disease) and true-negatives as shown in Equation (2-2). The accuracy metric is computed as shown in Equation (2-3).

$$Sensitivity = \frac{TP}{TP + FN} \tag{2-1}$$

$$Specificity = \frac{TN}{FP + TN} \tag{2-2}$$

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{2-3}$$

An automated detection system can be better evaluated through a receiver operating characteristics (ROC) curve. This curve is obtained by setting, for example, 100 different thresholds, and obtaining sensitivity and specificity pairs of the algorithm at each of these thresholds. The resulting sensitivity/specificity pairs are plotted in a curve which represents the sensitivity on the vertical axis and the complement to 1 of the specificity on the horizontal axis. A compact representation is possible by reducing the curve to a single number, the area under the ROC curve or AUC, which is a number between 0 and 1, where 1 denotes perfect performance.

2.4 Architectural considerations for the implementation of image processing algorithms

One of the main goals of image processing is to extract useful information from images and video. This task is difficult and many different algorithms are proposed. Image processing algorithms are often applied on large volumes of data, which implies high memory bandwidth requirements, and real-time processing implies high computational loads. The insufficient performance observed in GPP (general purpose processors) and GPU (Graphical Processing Unit) implementations has led to the use of hardware architecture and reconfigurable computing. Hardware architectures are often considered to accelerate their execution, especially to meet to main requirements: needs for data, and computationally needs. In this section we review some aspects of hardware implementations of image processing algorithms. First, the aspect of data access and processor architectures for image processing is discussed. Existing hardware implementations for image quality assessment, image enhancement and features extraction are reviewed with more attention to retinal image processing algorithms.

2.4.1 Data access for image processing

Data access is an important problem in image/video processing application. Image processing applications are both computationally and data intensive. Based on data dependencies, some algorithms require data from a relativelly small neighborhood, this includes point or pixel operators (such as gamma correction, threshoding) and window operators (such as 2-D convolution, erosion and dilation). Some other algorithms depend on data from the entire image like the Fast Fourier Transform and histogram techniques, or several images like motion estimation.

To allow efficient access to structured data such as in 2D or 3D images, Larabi et al. [79] proposed a low-cost n-dimensional cache architecture for FPGA-based image and signal processing systems on chip. They developed a theoretical model for the architecture and a methodology to define the cache's practical implementation based on the application and system parameters. Using this solution, numerical results indicate that 50% improvement in run-time performance can be achieved.

Deepa et al. [80] addressed the problem of on-chip memory management. To avoid memory redesigning when data access policy changes, the authors proposed a sub-bank dual port memory architecture, which consists of a modified single port memory to function as a true dual port memory with the help of two-port memory control unit, clock and address generators. This architecture is implemented and verified for an image coding algorithm (Lapped Biorthogonal Transform based low complexity Zerotee codec (LBT-LZC)) to achieve high throughput and lower power consumption. Khalvati et al. [81] addressed the local image processing algorithms with the aim to improve the efficiency of their hardware implementations by developing a “window memorization” technique. This technique identifies similar neighbourhoods of pixels to skip them and to minimize the redundant computations. The authors applied this technique to the Kirsch edge detector and median filter. A typical speedup factor of 1.58× with 40% less hardware is reported when compared to conventional optimized techniques.

Yu et al. [82] studied the case of sliding window operations in image processing. To estimate the maximal possible speedup, they defined three upper bounds according to: the area constraints, memory bandwidth constraints and on-chip memory size constraints. Then they determined with analytical representation the three upper bounds, and they considered the tightest one to decide on the hardware implementation of the algorithm. In addition, they proposed a new buffering method to build efficient memory hierarchy for the sliding window algorithms.

2.4.2 Hardware function evaluation for image processing

Image processing algorithms are computationally intensive, and involve the use of mathematical complex functions. These functions are very hardware consuming and represent a bottleneck for several algorithms. Computing this functions quickly and accuratly is a major goal in hardware design in general.

To solve this problem, Deng et al. [83] proposed a systematic approach for automatic generation of look-up-tables (LUT) for function evaluation and minimization in hardware resource for FPGAs. The developed approach support a class of functions and include sine, cosine, exponentials, gaussians, the central B-splines and certain cylinder functions that are frequently used for signal and image processing applications. To optimize their implementations in hardware, the function evaluation is based on numerical approximation using Taylor polynomials. For design space exploration, the proposed approach involves a search in three-dimensional space (data precision, sampling density and approximation degree) to meet customer requirements (accuracy, speed, area and on-chip memory).

Sasao et al. [84] proposed to use LUTs cascade to reduce circuit complexity. This method is suitable for automatic synthesis. They developed a synthesis method to convert a MATLAB-like specification into LUT cascade design. Experimental results of this approach show its efficiency when implemented on FPGA. Zhang et al. [85] proposed a special-purpose compiler that automatically generates customized look-up-tables and implementations for elementary functions under user given constraints. The generated implementations include a C/C++ code as well as a MATLAB-like code that can be translated directly to hardware module on FPGA platforms. The experimental results in image processing applications show significant resource saving to designs on FPGAs.

Lee et al. [86] proposed an automated system for function evaluation unit generation. The system selects the best function evaluation hardware for a given function, accuracy requirements, technology mapping, and optimization metrics (area, throughput and latency). They proposed also an automated bit-width optimization technique for minimizing the sizes of the operators in the data path. They explored a vast design space for fixed-point $sin(x)$, $log(x)$ and $\sqrt{x}$ to provide optimal function evaluation results for range and precision combinations between 8 and 48 bits.

Oskar Mencer [87] proposed a parameterized module-generators for pipelined function evaluation using look-up-tables, adders, shifters, multipliers, and dividers. The author discussed the trade-offs involved between (1) full look-up-tables, (2) bipartite (look-up add) units, (3) look-up multiply units, (4) shift-and-add based CORDIC units, and (5) rational approximation. An example shows that the look-up multiply unit produces competitive designs with data width up to 20 bots when compared with shift-and-add based CORDIC units. It shows also that look-up multiply method or rational approximation can produce efficient designs for large data widths when evaluating functions not supported by CORDIC.

2.4.3 Processor architecture for image processing

Pure hardware solutions have been proposed for a long time as a good solution to accelerate image processing algorithms. To deal with the flexibility, some authors proposed generic architectures for low-level window-based image processing algorithms (sliding window functions and spatial filters). Torres-Huitzil et al. [88] proposed a compact FPGA-based systolic architecture for real-time image processing. The architecture target window-based image operators include generic image convolution, gray-level image morphology and template matching. The computational core of the architecture is a configurable 2-D systolic array of processing elements. The architecture provides a throughput of 3.16 GOPs at a 60 MHz clock frequency for a 7×7 systolic array when implemented in FPGA. Saldana et al. [89] proposed a reconfigurable systolic-based architecture for low-level image processing. The architecture is customizable to perform operations for 3×3, 5×5 and 7×7 window coefficients. The architecture consists of 2D systolic array of processing elements. The architecture is based on parallel modules with internal pipeline operation, where every processing element can be configured according to a control word. To reduce the number of access to data memory and to extend the array capabilities, the arrays are provided with image buffers. The architecture is able to achieve a throughput of around 3.6 GOPS. Ariyadoost et al. [90] proposed a 2-D systolic adaptive DLMS FIR filters for image processing. The systolic architecture consists of some cell processors in tree scheme were used for improving speed of noisy image filtering.

Increasing demands on computational power for image/video processing applications motivated the use and the development of customizable processors. Instead of designing pure hardwired architectures, customizable processors reduce the efforts and offer the possibility to accelerate the computationally demanding parts of an application. In this context, Application Specific Instruction-set Processors (ASIP) have emerged as a promising solution to provide high flexibility and high computational efficiency in order to increase design reusability and short time-to-market.

Asri et al. [91] proposed a novel ASIP methodology and architecture for image processing. The designed ASIP can handle both image scaling and image enhancement using the same processor architecture, based on common pattern extraction and resource sharing methodology. Simulation results show that the proposed ASIP overwhelmed conventional ARM and RISC Processor. Liao et al. [92] proposed two ASIPs. An ASIP with a reconfigurable multi bank memory module and an SIMD computation pipeline, designed for pixel level image processing, and a 2-D ASIP with a slide register module and reconfigurable ALU modules, designed for 2D image processing. The first ASIP can perform color conversion, Gamma correction and dithering applications 4 to 10 times faster compared to its base processor. For the second ASIP can perform color interpolation, 3x3 edge detection, median filter and 5x5 edge detection applications 5 to 43 times faster when compared to its base processor.

Instruction level parallelism and data level parallelism are employed to try to propose more efficient VLIW and SIMD processor architectures. A flexible VLIW processor is proposed by Brost et al. [93] for real-time image processing. The authors developed a VLIW VHDL processor model with a variable instruction set and a customizable architecture. The authors realized a rapid prototyping of embedded contactless palm print extraction on an FPGA and obtained a processing time of 145.6 ms per image. Wittenburg et al. [94] proposed HiPAR-DSP, a parallel VLIW RISC processor targeting real-time image processing applications. The authors tried to analyse a wide class of image processing algorithms ‘properties, to propose this architecture that gains performance from data/instruction level parallelism. This processor is designed for: 3x3 convolution, 1024 complex samples FFT and gray-level histogram applications, and is able to reache a performance of more than 2 GOPS.

Kyo et al. [95] proposed the IMAP-CE, as an SIMD linear processor arrays based on an integrated memory array processor architecture. The IMAP-CE integrates 128 VLIW processing elements with a RISC control processor to provide a single instruction stream for the processor array. The IMAP-CE can reach a peak performance up to 51.2 GOPS operating under 100 MHz. Koenig et al. [96] proposed the KHARISMA processor architecture. This architecture consists of a reconfigurable instruction set multi grained array that integrates coarse/fine grained run time reconfigurable blocks. These blocks can be combined to realize different instruction set Architectures that may execute in parallel.

Stevens et al. [97] proposed the BioThreads architecture. It consists of a VLIW-based multiprocessor that target biomedical image processing applications. In addition to instruction and data parallelism, this architecture handles efficiently the thread-level parallelism. This is possible with the aid of a novel mechanism for the dynamic creation and allocation of software threads to uncommitted processor cores by implementing key POSIX Threads primitives directly in hardware, as custom instructions.

2.5 Tools for HDL description generation for image processing

Processing of small sized images is used in several biomedical and military applications (artificial retina, UAVs and guided missiles) [98]. Traditional VGA, NTSC, PAL use medium sized images while medical imaging and several other applications need more resolution (QSXGA 2560x2048 and more) and high quality images. To overcome the real-time constraint for high resolution image processing, reconfigurable hardware and FPGAs has been proposed. FPGAs present some advantages with their reprogramability and parallelism possibilities while targeting real-time applications. However, the low-level programming model of FPGAs is their major disadvantage. With the vast range of image processing applications, traditional design methodologies (starting the system design from scratch) are inefficient and the need to new methodologies to describe modern image processing techniques becomes increasingly necessary. The goal is to meet time and performance constraints (area, energy) while offering shorter time to develop the technology and to bring it to the market.

High Level Synthesis (HLS) refers to the generation of synthesizable RTL from user defined behavioral description automatically. The research in HLS has led to the development of several tools like Vivado of Xilinx with the aim to generate hardware from a high level description. High level descriptions are usually coded in C, ANSI C, C++ and MATLAB. The HLS methods can be classified into two approaches: the annotation and constraint driven approach, and the source directed compilation approach [99]. In the first approach the source code is preserved in C or C++ as much as possible. Annotation and constraint files are used to drive the compilation process, such as SPARK, SeaCucumber, SPC, Stream-C, Catapult C and DEFACTO etc. The second approach modifies the source language to let the designer to specify, for instance, the amount of parallelism or the size of variables, such as ASC, C2Verilog, Handel-C, Handy-C, Bach-C and SpecC etc.

Desmouliers et al. [100] proposed an image and video processing platform (IVPP) based on FPGA. IVPP is a hardware/software co-design platform implemented on FPGA using high-level synthesis. It consists of a Microblaze processor with a front-end (capturing video data) and a back-end (displaying processed data). Hardware blocks synthesized from HLS language can be integrated and plugged-in to provide complete hardware solution. The authors presented also a framework that supports custom logic (user peripherals). A development tool called Synphony C High

Level Synthesis tool is used to convert C-based algorithms to hardware that can be easily incorporated into IVPP.

Andriamisaina et al. [101] developed a multimode architecture with dedicated design flow and its associated HLS tool GAUT for image processing applications. The tool generates a single RTL hardware architecture optimized in area from a unified description of a set of time-wise mutually exclusive tasks and their throughput constraints. The authors proposed a joint-scheduling algorithm to reduce the register, the steering logic and the controller complexities. This approach shows a significant area saving compared to the state-of-the art techniques. Boubekeur et al. [102] designed a high level synthesis tool targeting massively parallel image processing ASICs. From a high level description, the tool is able to generate an optimized SIMD (Single Instruction Multiple Data) mesh connected array of one-bit processing elements with minimized resources. This tool targets low-level image processing applications to generate dedicated optimized ASIC in terms of area and performance.

Hannig et al. [103] proposed an automatic synthesis of highly complex, throughput optimized architecture of an adaptive multi-resolution filter for medical image processing. The designed filter includes 16 parallel working modules, where the most computationally intensive module achieves software pipelining by a factor of 85 (computations of 85 iterations overlap each other). The implemented technique contributed to reduce the complexity and power efficiency. Also it was able to reduce the productivity gap of embedded system design by almost two orders of magnitude. Yazhuo et al. [99] designed a parameterized architecture model in high level synthesis for automated generation of hardware frames for all image processing applications. They employed data reuse to reduce the number of data memory accesses. A special control unit is designed to dominate the dataflow and to make it possible to store a small part of the data values in internal RAM and smart buffer while providing sufficient memory bandwidth for the custom data path.

Shatnawi et al. [104] developed a technique for scheduling and processor allocation during the synthesis of integrated heterogeneous pipelined processing elements for DSP applications. The new technique produces high efficiency when compared to homogeneous implementations results. The proposed technique achieved efficiency in hardware implementation at logic-level by shrinking the processing units counts used without compromising the rate and delay optimality criteria.

Improvement in high level synthesis performance during the synthesis of some image processing algorithms was detected.

Benkrid et al. [105] developed a high level description environment to bridge the gap between application design and hardware description, and to allow efficient compilation to the form of EDIF netlist. This approach is based on parameterized description of task-specific architectures. The developed system targets Xilinx XC 4000 and Virtex series FPGAs.

2.6 Summary and research objectives

In this chapter we introduced the algorithmic aspects of the automated detection of eye diseases using digital retinal images. In any automated detection system of the diabetic retinopathy, three main parts are necessary, the retinal images quality assessment, the retinal image enhancement and their features extraction. These three parts are reviewed throughout this chapter. Concerning the hardware implementations, we started first by discussing the architectural considerations for the implementation of image processing algorithms. We detailed especially three points: (1) the data access, (2) hardware function evaluation, and (3) the processors architectures for image processing. We then surveyed existing tools for HDL description generation for image processing algorithms.

After this literature review, and after analysing the context of retinal image processing and their hardware implementations, we have identified several opportunities for acceleration and efficient implementations for two algorithm classes: retinal image quality assessment, and retinal blood vessel segmentation. To the extent of our knowledge, hardware implementations targeting retinal image quality assessment have not been proposed yet. For retinal blood vessel segmentation, the existing implementation are limited in terms of performances especially in the context of telemedicine. There are no hardware implementations able to process high resolution images. The existing ones are targeting images of low resolution and their scalability issue is not discussed. Our research objectives have been defined in the light of these limitations.

The main objective of this research is to design and implement algorithms for retinal image processing on hardware platforms with the aim of achieving high throughput while maintaining level of flexibility. In order to reach our goals, the following specific objectives are identified:

- Propose adequate optimized hardware architectures for retinal image processing algorithms especially retinal image quality assessment and blood vessel segmentation algorithms. The proposed architecture should satisfy the high performance requirements in a telemedicine context.
- Develop a tool able to generate optimized low-level HDL descriptions automatically for the proposed architectures. We aim to propose a tool able to keep low level programming constructs while saving programming flexibility in same time.
- Simulate, implement, test and evaluate several architectures and designs of retinal image processing to assess the performance of the proposed solutions, and compare it with existing works.

Next chapter will focus on retinal image quality assessment algorithm and its acceleration as a first task in the retinal image processing pipeline.

CHAPTER 3 CO-PROCESSOR FOR RUN LENGTH ENCODING ALGORITHM

Retinal image quality assessment is the first task in the pipeline of automated retinal image processing systems. This task is necessary because it allows to avoid collecting images of insufficient quality for automated processing. Such a task could also be integrated with retinal cameras to allow taking rapid decisions if new acquisitions are necessary. In this chapter, we present a Zynq-based system to compute Run-Length encoding Matrix features for retinal image texture analysis. The aim is to accelerate the algorithm and allow the development of an embedded quality assessment system for retinal images. A software version of the algorithm was first implemented on PC and an ARM CPU of the Zynq platform. To improve the performance of the software implementation, we propose a co-processor architecture implemented in the programmable logic portion of the Zynq platform.

3.1 Introduction

Retinal image quality assessment as a first task in the pipeline has to be fast enough to allow fast decision on the acceptance or not of the acquired retinal image. Figure 3-1 shows examples of bad quality retinal images that should be rejected. Several research studies have shown that image texture analysis can be used for an objective assessment of the quality of the retinal images [18, 106]. In [107], an algorithm for retinal image quality assessment based on image texture analysis was developed. The proposed algorithm employed the local sharpness and texture features by applying the cumulative probability of blur detection metric and run-length encoding algorithm, respectively. The algorithm demonstrated sufficient robustness to detect relevant images for automated diagnosis using image texture analysis. Unfortunately, image texture analysis algorithms are computationally intensive due to their high complexity. For retinal image quality assessment, the images should be analysed on site after image acquisition but before transmission to a remote telemedicine processing system. This represents a challenge for embedded retinal imaging especially with high resolution retinal images.

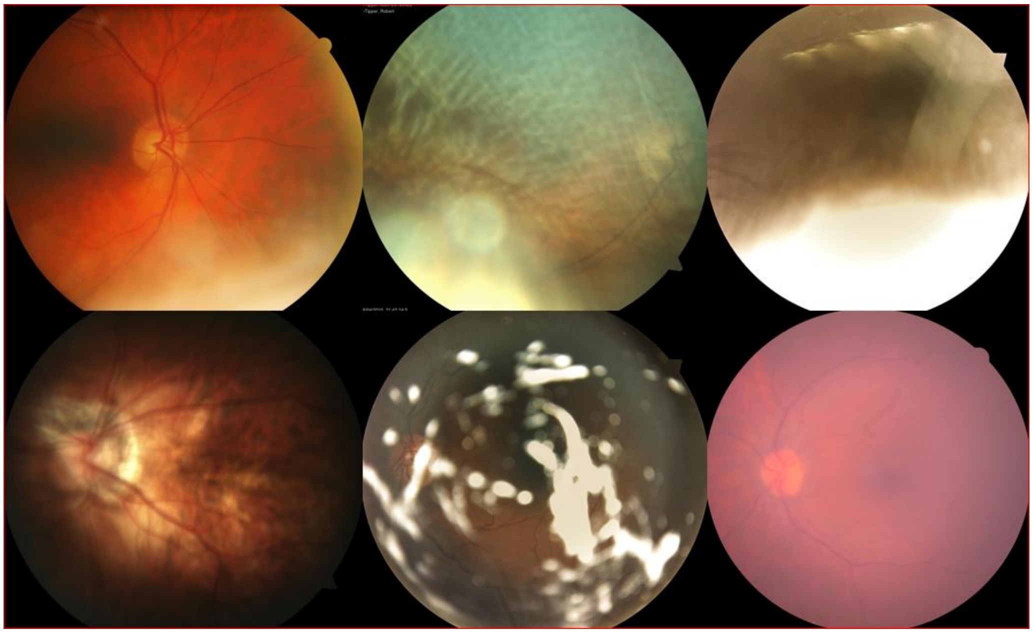


Figure 3-1. Examples of bad quality retinal images

Several hardware implementations have been developed to accelerate image texture analysis algorithms. Tahir et al. [108] proposed an FPGA implementation of the Grey-Level Co-occurrence Matrix (GLCM) and Haralick texture features to accelerate their computation. To demonstrate the efficiency of the developed architecture, they implemented a multispectral computer vision system for automatic diagnosis of prostatic cancer. The results show that the performance of an FPGA is approximately 9× faster than that of a Pentium 4 PC. Akoushideh et al. [109] proposed a hardware architecture implemented on FPGA to compute the GLCM matrix and their features. The results show that the hardware implementation is 214× faster than the software implementations.

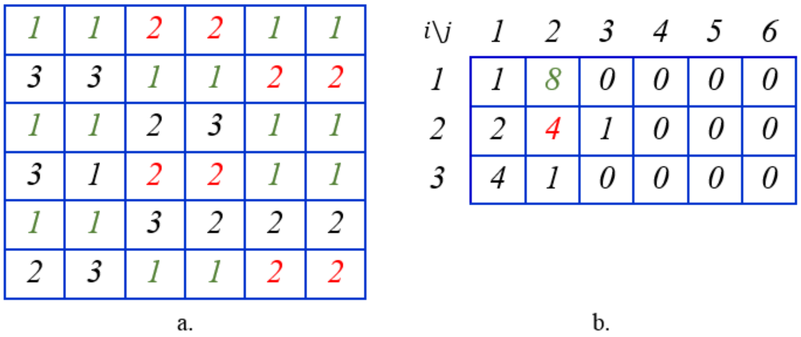
Bouris et al. [110] proposed an FPGA-based implementation of the SURF (Speeded-Up Robust Features) detector. The results show that the implemented system outperforms a state-of-the-art dual-core Intel CPU by at least 8×. Yao et al. [111] proposed an architecture of optimised SIFT (Scale Invariant Feature Transform) feature detection for an FPGA implementation of an image matcher. The proposed FPGA implementation is able to detect the features of a typical image of 640 × 480 pixels within 31 milliseconds.

The Run-length encoding algorithm constitutes a common block in numerous applications such as retinal blood vessel segmentation [112], neovascularization detection [113] and data transfer and image compression [114]. The algorithm encodes an image by counting runs of pixels with a given value in (typically four) different directions, which is useful for data compression applications. For retinal image quality assessment, we need to compute the Run-Length Matrix (RLM) and its features, which is useful for retinal image quality assessment. Several hardware implementations of the run-length encoding algorithm have been presented [115, 116], but the RLM features computing is not discussed since their computation is not necessary for the targeted applications. In this work, we are interested on the implementation of the run-length encoding algorithm for retinal images texture analysis on a Zynq platform. This chapter makes the following contributions:

- It proposes an embedded system for retinal image RLM features computation. This first solution is completely software.
- It proposes a hardware co-processor to accelerate the RLM image features computation.
- It implements the RLM features to provide quantitative information describing textural properties of images.

3.2 Run-Length Matrix and RLM Features

The use of RLM was proposed by Galloway [117] for texture feature extraction. Run-length encoding is used to represent strings of symbols in an image matrix. The gray level run is defined as a set of consecutive, collinear pixels having the same gray level. The length of the run is the number of pixels in the run. For a given image, a RLM $p(i,j)$ is defined for a specific direction as the number of runs with pixels of gray level i and run length j in that direction. The first dimension i corresponds to the gray level of the pixel. The size M of i is the maximum gray level of the pixel. The second dimension j corresponds to the run length. The size N of j is equal to the maximum run length. Figure 3.2 shows an example of an image and its corresponding run-length matrix.



1	1	2	2	1	1
3	3	1	1	2	2
1	1	2	3	1	1
3	1	2	2	1	1
1	1	3	2	2	2
2	3	1	1	2	2

a.

i\j	1	2	3	4	5	6
1	1	8	0	0	0	0
2	2	4	1	0	0	0
3	4	1	0	0	0	0

b.

Figure 3-2. a. Image and b. its corresponding Run Length Matrix

Using this computed matrix $p(i,j)$, the following three features can be computed (originally proposed by Galloway [19]):

1. Short Run Emphasis (SRE)

$$SRE\ = \frac{1}{n_r}\sum_{i=1}^{M}\sum_{j=1}^{N}\frac{p(i,j)}{j^2} = \frac{1}{n_r}\sum_{j=1}^{N}\frac{p_r(j)}{j^2} \tag{3.1}$$

2. Long Run Emphasis (LRE)

$$LRE\ = \frac{1}{n_r}\sum_{i=1}^{M}\sum_{j=1}^{N}p(i,j).j^2 = \frac{1}{n_r}\sum_{j=1}^{N}p_r(j).j^2 \tag{3.2}$$

3. Run Percentage (RP)

$$RP\ = \frac{n_r}{n_p} \tag{3.3}$$

Where n_r, n_p, M and N are the total number of runs, the number of pixels in the image, the number of RLM rows and the number of RLM columns, respectively.

The following features were later proposed by Chu et al. [118] and Dasarathy and Holder [119].

4. Low Gray-Level Run Emphasis (LGRE)

$$LGRE\ = \frac{1}{n_r}\sum_{i=1}^{M}\sum_{j=1}^{N}\frac{p(i,j)}{i^2} = \frac{1}{n_r}\sum_{i=1}^{M}\frac{p_g(i)}{i^2} \tag{3.4}$$

5. High Gray-Level Run Emphasis (HGRE)

$$HGRE\ = \frac{1}{n_r}\sum_{i=1}^{M}\sum_{j=1}^{N}p(i,j).i^2 = \frac{1}{n_r}\sum_{i=1}^{M}p_g(i).i^2 \tag{3.5}$$

6. Short Run High Gray-Level Emphasis (SRHGE)

$$SRHGE = \frac{1}{n_r}\sum_{i=1}^{M}\sum_{j=1}^{N}\frac{p(i,j).i^2}{j^2} \tag{3.6}$$

7. Long Run Low Gray-Level Emphasis (LRLGE)

$$LRLGE = \frac{1}{n_r}\sum_{i=1}^{M}\sum_{j=1}^{N}\frac{p(i,j).j^2}{i^2} \tag{3.7}$$

8. Long Run High Gray-Level Emphasis (LRHGE)

$$LRHGE = \frac{1}{n_r}\sum_{i=1}^{M}\sum_{j=1}^{N}p(i,j).i^2.j^2 \tag{3.8}$$

The proposed system is divided into four steps to compute the image features, as shown in Figure 3-3: image acquisition, mask generation, RLM computing and RLM features computing.

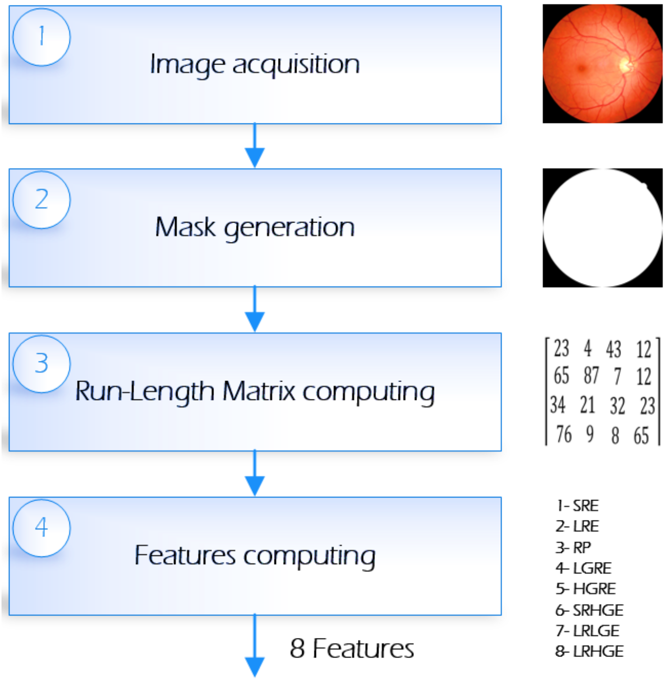


Figure 3-3. Different steps for computing the image features

3.2.1 Image acquisition

The retinal images are taken using a retinograph which is a camera with special optics capable of taking images of the retina. The image acquisition step is responsible for loading the retinal image into the processing system and transferring it onto a server.

3.2.2 Mask generation

The retinal image is square, while the retina is round. The retina does not occupy the whole image. The mask is a binary image of the same size and it identifies which pixels of the image correspond to the retina. Figure 3-4 shows an example of an original image and the corresponding generated mask.

3.2.3 Run-Length Matrix computation

In this step, we compute the RLM and its features as described in the section II. We compute four matrices corresponding to different orientations (0°, 45°, 90° and 135°) to increase the size of the features set. Only the pixels inside the mask are considered. We compute the four matrices for each channel of the image: red, green and blue channels.

3.2.4 Features computing

In this step, we compute the following 8 features since they contribute efficiently to differentiate the images based on their quality. The 8 features are described in section 3.2 by the equations (3.1) to (3.8).

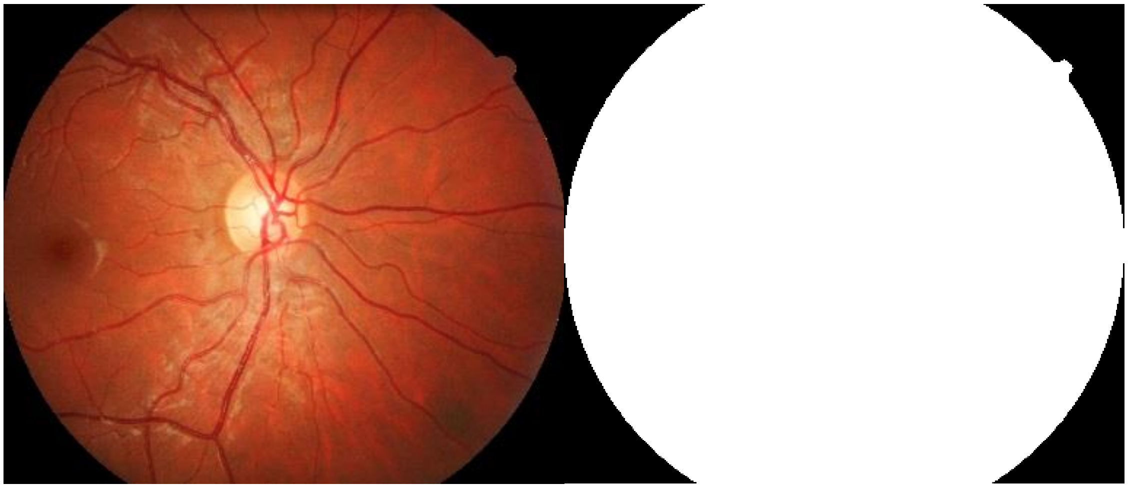


Figure 3-4. Original image and the corresponding mask

3.3 Zynq based architecture

The proposed system is implemented in a Zynq-7000 AP SoC that integrates a dual-core ARM Cortex–A9 based processing system (PS) and a 7-series Xilinx Programmable Logic fabric (PL) in a single device. The PS and PL of Zynq communicate using the Xillybus core [22]. The Xillybus core is a hardware module that communicates with the ARM processor through the AXI bus and DMA buffers to transfer data. Any additional custom logic should be connected and interfaced to the Xillybus core via FIFOs. The ARM processor is clocked at 667 MHz and runs the Linux operating system. Figure 3-5 shows an overview of the proposed Zynq-based system.

3.3.1 Data analysis

The maximum size of the RLM for 8-bit per color pixels is 256 × image width. The longest possible run is equal to the image diagonal. This situation does not occur in general for retinal images. In order to reduce RLM size we analyzed a representative image dataset. This proprietary dataset includes 213 retinal images. We computed the RLMs for these images and found that the maximum value of the run-length does not exceed 257 pixels, while the image width for high resolution images often exceeds 1000 pixels. The maximum value of the runs was found to require 18 bits for its representation. We take these values to be representative of the vast majority of retinal images.

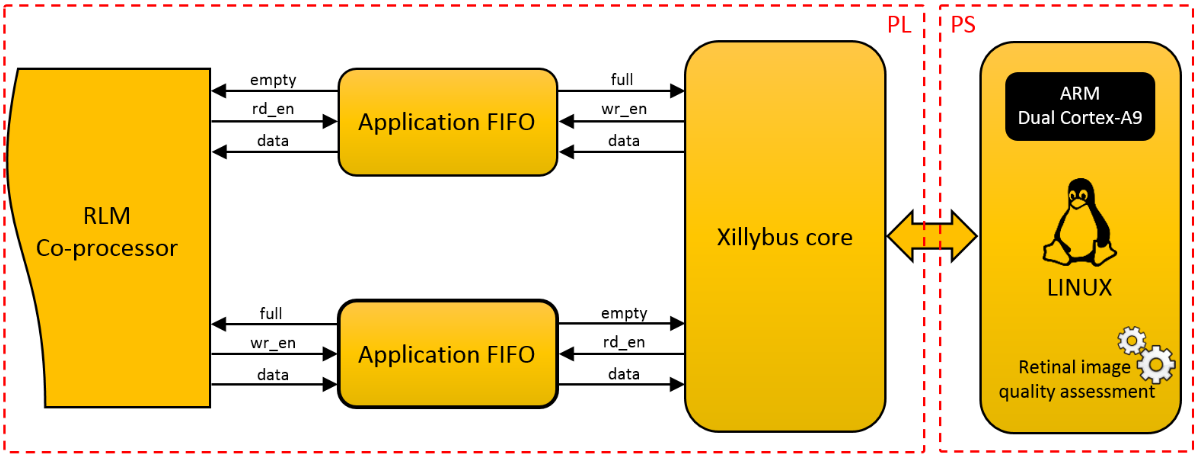


Figure 3-5. Overview of the Zynq-based system

3.3.2 RLM features co-processor

Our first implementation of the proposed system to compute the RLM features is a software implementation of all the four steps in C++ with OpenCV. We are interested on the RLM features algorithm (steps 3 and 4). To accelerate this algorithm, we propose to keep the RLM computing (step 3) in software and move its features computing (step 4) in hardware in a co-processor to get benefits from the parallelism in programmable logic. The software plays the role of host and sends the matrix to the co-processor then receives computed results. The software interfaces with the co-processor via two 32-bit × 512 FIFOs as shown in Figure 3-5. The stream between the PS and the PL is done through a generic 32-bit AXI stream interface while the PL to PS stream is through a high-performance AXI port.

The co-processor architecture is shown in Figure 3-6. The computation of the features is done in parallel. Modules F1 to F8 share the same input which is the values of the RLM.

In each clock cycle a new value is obtained and the computed features are updated until the last value of the RLM. The F8 calculator module computes the RP feature. In the first stage of F8, we accumulate the run-length values to compute the number of runs, and then the computed features are divided by the number of runs. We can see in the Figure 3-7 that the output from the F8 calculator (number of runs) is shared with the other modules as input of a divider.

All computations are done in fixed-point. The fractional part is represented by 18 bits to ensure an acceptable precision. For the integer part, we consider that the RLM values are 18 bits wide for a 256 × 257 matrix size. The division operation is performed by multiplication by the reciprocal. The divisor values are limited to the interval $[1^2, 257^2]$ and the corresponding reciprocals are stored in a look-up-table. Two division modules are necessary, to divide by i and j. The quotients are reused for other features.

A state-machine manages the communications and the different steps of the co-processor. The behavior of the state machine is described in Figure 3-7. If data is available in the FIFO, the state changes from the Idle to the Features computing state. If all data of the RLM is processed, the state changes from Features computing to the Last division state. In this state, the division by the number of runs is performed and the state changes to the Send results state. In the Send results state, the computed features are sent back to the host. Since the computed features are represented by more than 32 bits, two clock cycles are necessary to send each feature to the host 32 bits at a time.

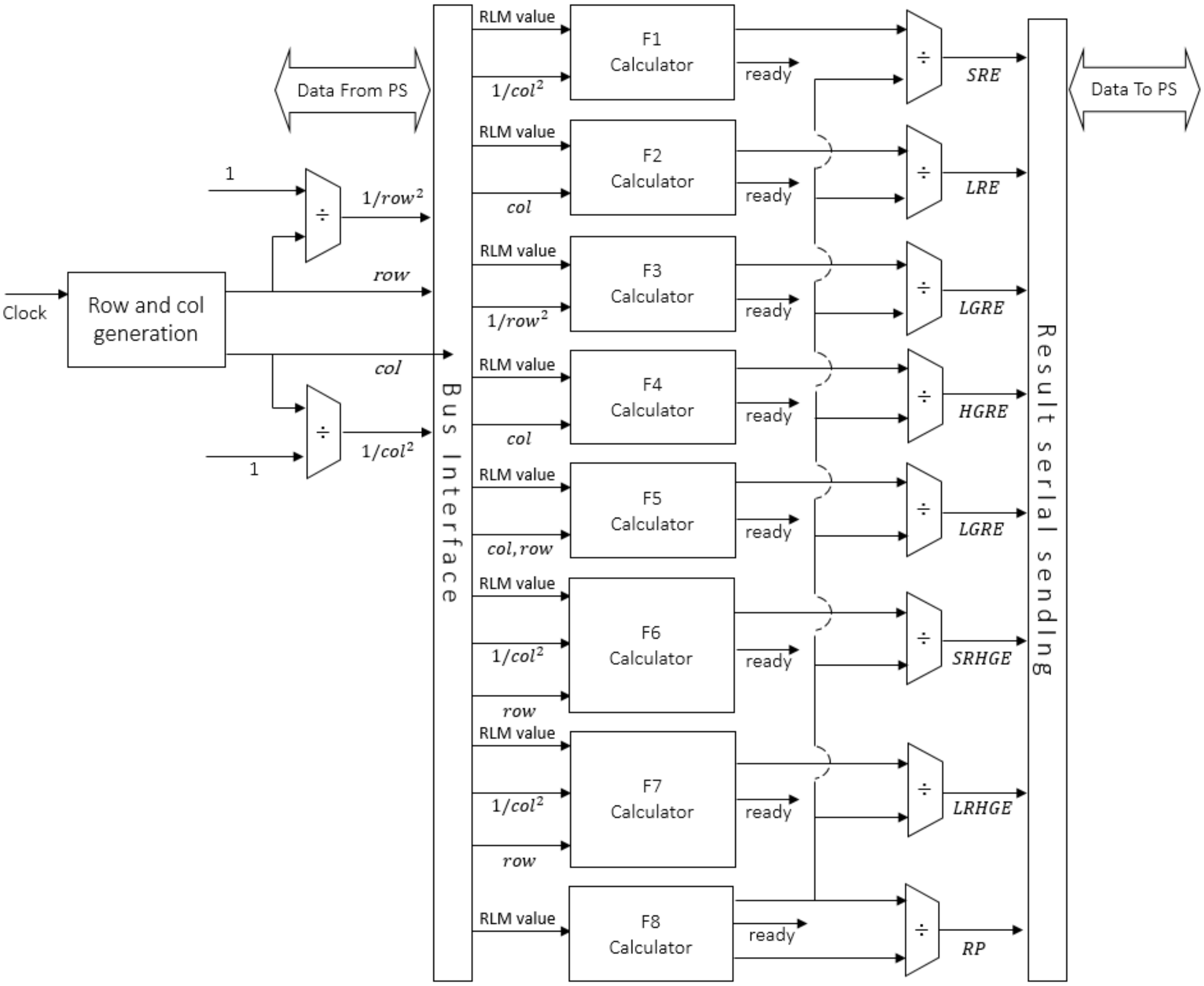


Figure 3-6. RLM features co-processor architecture

3.4 Results and discussion

This section presents and discusses the results. We implemented the system using the Zed-Board with a Zynq-7000 SoC platform which incorporates the XC7Z020-1CLG484CES device. Figure 3-8 shows the proposed system for RLM features computing for retinal images. The implemented software-based approach was first developed for a desktop computer and then ported to the Zynq platform under the Linux operating system.

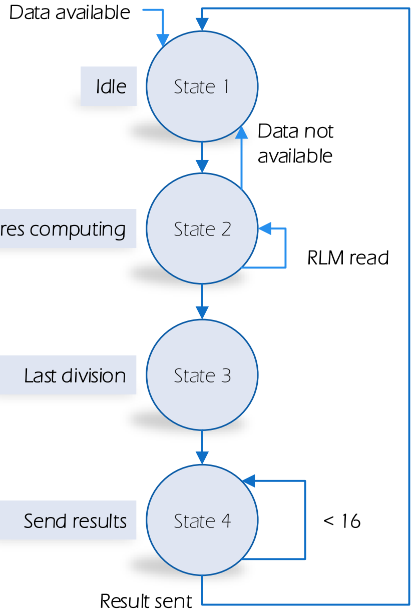


Figure 3-7. State machine sequencer of the co-processor

Table 3.1 shows the programmable logic resources utilization with and without the co-processor. Other than the co-processor, the programmable logic includes communication and user interface modules. These modules are necessary for the Linux operating system to be able to display in a VGA screen, and to communicate with the PL via the Xillybus core.

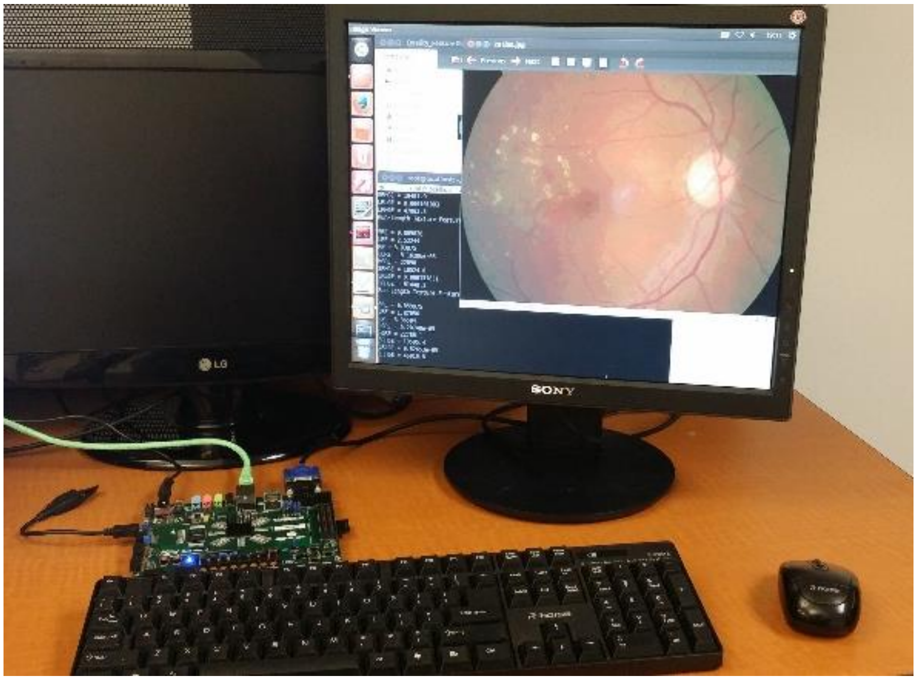


Figure 3-8. Overview of the proposed system for RLM features computing

Table 3.2 shows the execution time for several image sizes executed on the Zynq PS. As expected, the processing time grows approximately linearly with the number of pixels. Table 3.3 presents the execution time for three test cases for an image of 2496 × 1664 pixels. The first test case is the pure-software solution executed on an Intel Core i7 2620, 2.7 GHz processor.

Table 3.1. Programmable logic resources usage

Resources	**RLM co-processor**	**Communication and user interface modules**	**Total / %**
Flip-Flops	348	4177	4525 (4%)
LUT	1012	4762	5774 (10%)
BRAM/FIFO	5	5	5 (3%)
DSP48E1s	36	0	36 (13%)

Table 3.2. Execution time under Zynq PS for several image sizes

Image size	**Execution time (All software)**
1620 × 1444	1.9 s
2588 × 1958	4.3 s
3456 × 2304	6.7 s

The execution time for an image of 2496 × 1664 pixels with four color channels (Red, Green, Blue and Gray) executed in an i7 CPU is 597 ms. This one is faster than the Zynq based implementation; however, such a CPU is not suitable for an embedded system. The second test case is the pure-software solution for the RLM features computing executed on the Zynq PS running Linux. The execution time is 3.5 s. The third case shows the results of the RLM features computing when the co-processor is used. In this case, the execution time is 116 ms, which represent an acceleration by a factor of 30.1×.

Table 3.3. Execution time under Zynq PS for several image sizes

Test case	Platform	Execution time	Speedup
All software	i7-2620 CPU, 10GB RAM	597 ms	
RLM features only: software	Zynq PS (ARM-A9, 667 MHz, 512 MB RAM)	3.5 s	1
RLM features only: software + co-processor	Zynq PS + PL	116 ms	30.1×

By adding the co-processor in PL, the number of LUTs and flip-flops slightly increases. The RLM co-processor necessitates 348 flip-flops, 1012 LUTs and 36 DSP48E1s as additional area. The additional 13% of DSP48E1s slices and 2% of LUTs and flip-flops is fully justified when considering the acceleration by a factor of 30.1×.

3.5 Conclusion

This chapter presented a co-designed system implemented on the Zynq 7000 platform to compute retinal image features using run-length encoding. The whole system is implemented first in software. For high resolution retinal images of 2496 × 1664 pixels, the execution time for four color channels in an i7 CPU is 597 ms. When compared to a software implementation on the zynq platform, the i7 CPU implementation is faster. However, the Zynq platform is more suitable for an embedded system. To accelerate the processing, a hardware co-processor for RLM features computing is implemented on the programmable logic which achieves an acceleration of 30.1×. We implemented and tested the RLM features computing for four color channels of the image. Future works will integrate this system as a retinal image features generation engine. The generated features will be used with a classifier to decide if the image quality is acceptable for automated diagnosis.

CHAPTER 4 FLEXIBLE ARCHITECTURES FOR RETINAL BLOOD VESSEL SEGMENTATION USING MATCHED FILTERING

In this chapter, we address the problem of retinal blood vessel segmentation using the matched filtering approach. We present several hardware architectures targeting FPGA and ASIP implementations. Our contribution is in the flexibility and scalability of the proposed architectures to deal with images of low and high resolutions. This important issue has not yet been properly addressed in the literature of hardware implementations for retinal images processing.

4.1 Introduction

Retinal blood vessel segmentation as a main task in automated systems of several eye diseases detection is an active research topic. Several research works are presented in chapter 2. Several challenges are noticed in the literature review. The need of acceleration using hardware architectures is crucial to practical utilization of automated systems especially for large scale systems such as in telemedicine. Proposed architectures in the literature are all targeting images of low resolution. With recent advances in imaging technology, images of high resolution are more available and favorable regarding their quality and the possibility to detect small vessels. Proposing optimized hardware architectures and making them scalable is a difficult task, hence, scalability is an important factor when algorithm parameters are to be adapted to image resolution accordingly.

In this chapter, we present a scalable hardware architecture for the matched filter algorithm for retinal blood vessels segmentation. The matched filter algorithm was chosen because of its popularity and performances. However, the matched filter algorithm is computation intensive since it requires the computation of several window convolutions for each pixel in the image. This can become a serious bottleneck especially for high-resolution images after they have become the norm with recent advances in retinographs technology. The proposed architecture uses several architectural optimizations to reduce the area utilization and to accelerate processing. The architecture is optimized in terms of resources utilization and throughput. We also propose a tool for automatic HDL description that takes as input the matched filter algorithm parameters. Our tool makes the algorithm parameters selection more flexible and generates a specific HDL description based on an optimized scalable architecture template. In this chapter we also propose an Application Specific

Instruction-set Processor (ASIP) based on the Tensilica Xtensa LX extensible processor for the matched filter algorithm. In this chapter we make the following contributions:

- Design and implementation of a hardware architecture for matched filter.
- A new architecture for the matched filter based on an ASIP with two additional custom instructions.
- A compiler is introduced to automatically generate optimized low-level hardware descriptions, suitable for FPGA implementation, from any algorithm set of parameters.
- Comprehensive results are provided in terms of blood vessel segmentation quality with respect to computations accuracy.

4.2 Matched filter algorithm description

In this section we describe the implemented matched filter algorithm for retinal blood vessel segmentation. The matched filter was first proposed for eye blood vessel segmentation in [40] and is one of the most often used algorithms. It is a template matching algorithm, based on the prior knowledge of the objet to be recognized. The matched filter approximates the intensity of the vessel cross-section by a Gaussian shape curve. Figure 4-1 shows the intensity profile of a cross section of a typical retinal blood vessel for different orientations.

As proposed by Chaudhuri et al. [40], the matched filter is designed based on three main properties:

- Blood vessels usually have small local curvature and can be approximated by piecewise linear segments.
- The vessels appear darker relative to the background.
- The intensity profile varies by a small amount from one vessel to another.

Based on these properties, a Gaussian function is used as a model to fit the blood vessels and the matched filter kernel can be expressed by (4.1):

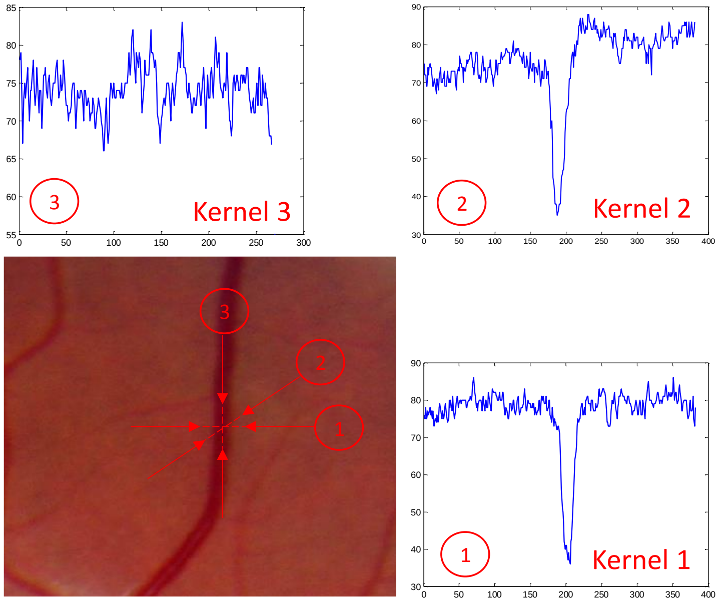


Figure 4-1. Retinal blood vessel (a) and the intensity profile of its cross section for different orientations

$$k(x,y) = \begin{cases} -\exp\left(-\frac{x^2}{2\sigma^2}\right) - m, \forall\ |y| \leq \frac{L}{2}, |x| \leq 3\sigma \\ \qquad otherwise\ 0 \end{cases} \qquad (4.1)$$

where L is the length of the vessel segment assumed as piece-wise line in the kernel, σ defines the spread of the intensity profile and m is used to normalize the filter response and to get a zero mean value. m is expressed by (4.2).

$$m = \left(\int_{-3\sigma}^{3\sigma} exp\left(\frac{-x^2}{\sigma^2}\right)\, dx \right) / 6\sigma \qquad (4.2)$$

Using equations (4.1) and (4.2), one line of the kernel is calculated. Assuming that the vessels have fixed width and orientation for a short piece-wise line, the model is extended to two dimensions by duplicating the calculated line L times to create a 2D kernel as shown in Figure 4-2. The pair (x, y) are the coordinates of each element in the kernel. Since the vessels may appear in any orientation, several rotated instances of the kernel are considered. Kernels with different orientations are calculated using (4.3):

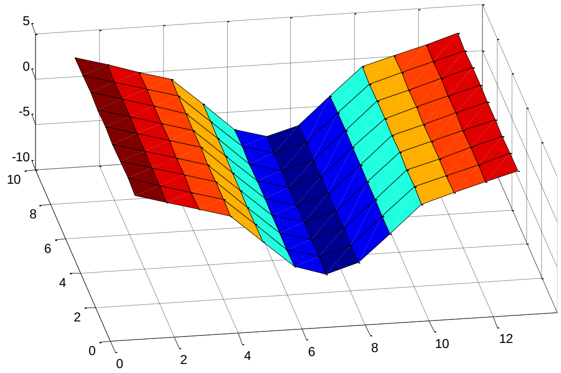


Figure 4-2. 2D kernel of the matched filter

$$[u\ v] = [x\ y]\begin{bmatrix} cos\ \theta & -sin\ \theta \\ sin\ \theta & cos\ \theta \end{bmatrix} \qquad (4.3)$$

where (u, v) are the new rotated coordinates of (x, y), and θ is the rotation angle. (u, v) are truncated at $\pm 3\sigma$ and the kernel width is $6\sigma + 1$. Each rotated kernels is convolved with the retinal image, and the maximum response for each pixel to a given kernel then is registered to indicate the presence of the vessels. Using 12 kernels separated by an angle of 15° is generally considered adequate to detect vessels with acceptable accuracy [41].

4.3 Proposed architectures for the matched filter algorithm

In this section, we give an overview of the blood vessel segmentation system and describe in more detail the two proposed architectures.

4.3.1 Scalable hardware matched filter architecture

In this section, we describe the proposed scalable hardware architecture and its HDL description generation. The Figure 4-3 gives an overview of the system that includes the proposed scalable hardware architecture for retinal blood vessel segmentation. The retinal image is stored in gray scale in the on-chip memory of the FPGA. On-chip memory is not suitable for storing high-resolution images, and the data input/output issue must be solved. However, using an image buffer allows to process the image data in streaming, and just a small part of the image must be stored. This is normally the green channel, which exhibits the best contrast for vessel segmentation purposes [41]. This memory is addressed by an address generator. The processing results are stored in a video memory and can be displayed on a VGA screen using a VGA controller. They can also be passed on for further processing and diagnosis.

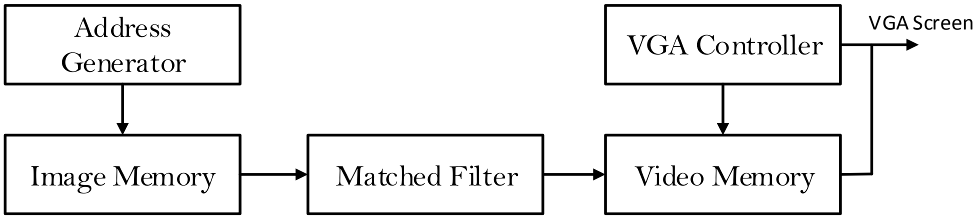


Figure 4-3. Retinal blood vessel detection system overview

The proposed matched filter architecture is shown in Figure 4-4. It is composed of three major modules: the Image Buffer, the Convolution Unit and the Max Selector Unit.

The Image Buffer receives a pixel stream from an on-chip memory. The Image Buffer uses parallelism to manage access to the pixel to be processed and to its neighborhood with no delay. It is designed using a shift register. The length of the shift register is $(N - 1) \times ImageW + N$, where N and $ImageW$ are the kernel and image width, respectively. The kernels are square matrices of $N \times N$ coefficients.

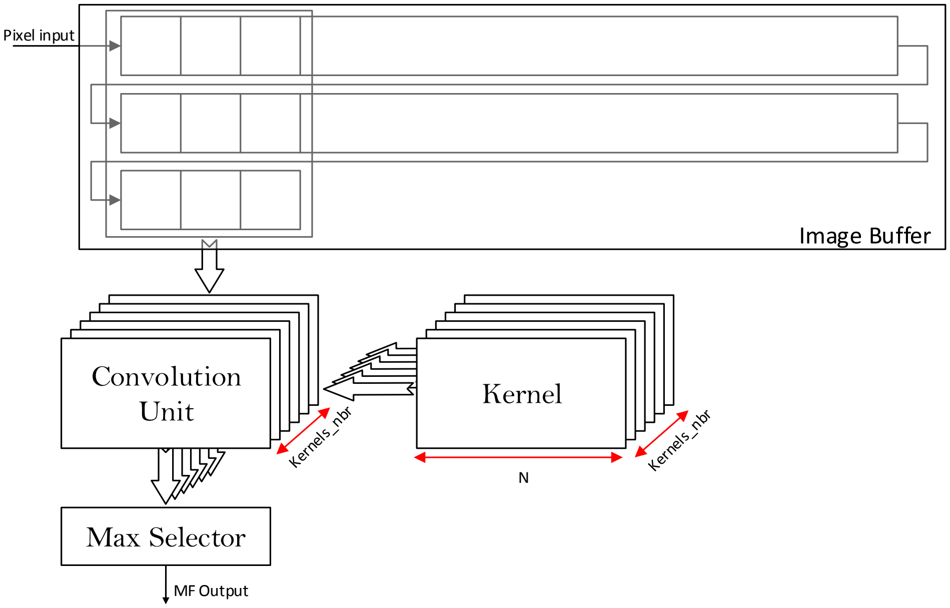


Figure 4-4. The matched filter scalable architecture overview

The main processing module is the Convolution Unit. The Convolution Unit first performs the multiplication between the kernel coefficients and the image pixels. The products are then added to calculate the filter response for a given kernel for each pixel. Since the coefficients of the kernel are constant signed integers of low value, we replace the multipliers by left shifts and additions. After the multiplication stage, a fully pipelined adder tree is used to calculate the sum of the products. A specific Convolution Unit is created for each kernel for the different angles. This allows the calculation of the convolution for all kernels in parallel. Figure 4-5 shows a simplified architecture of the Convolution Unit with eight inputs. The Convolution Unit is scalable in the number of inputs, number of pipeline stages and adder tree structure.

The Max Selector Unit finds the maximum response of the filter for all kernels. The number of inputs is equal to the number of kernels ($Kernel\_nbr$). The proposed architecture instantiates the $Kernels\_nbr$ Convolution Units with the Image Buffer and the Max Selector Unit.

The matched filter parameters are selected according to a specified blood vessels segmentation quality. Due to parameters tuning, hand coding is not an attractive solution when parameters are changed. We propose to generate a synthesizable VHDL description of the matched filter architecture automatically using a developed tool based on the proposed scalable architecture as a template.

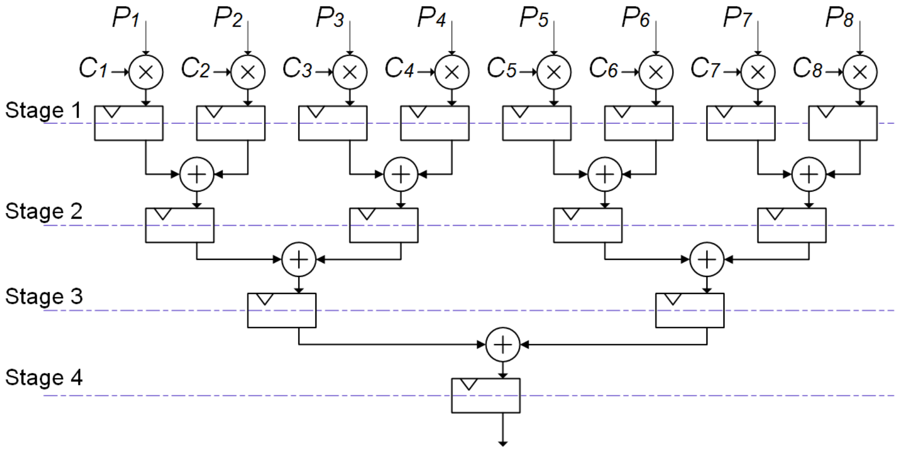


Figure 4-5. A simplified architecture of the Convolution Unit

An overview of the tool is given in Figure 4-6. It is implemented in MATLAB 7.12.0. From a set of parameters ($L, \sigma, Kernel\_nbr, ImageW, ImageH$), the tool generates the synthesizable VHDL code of the different components of the proposed architecture. In the kernel generation stage, the0° kernel is calculated using the L and σ parameters. The other kernels are then generated by rotating this kernel by steps equal to 180 / $kernel\_nbr$. The second step is the Image Buffer generation. The Image Buffer is generated based on two parameters: the image size and kernel width. The next step is the Convolution Unit generation. For each kernel, a corresponding Convolution Unit is generated. Each unit is specific to the kernels coefficients. The resulting unit area is minimized by:

- eliminating multiplication by 0 coefficients;
- replacing the multipliers by shift and add operations; and,
- minimizing the internal signal widths.

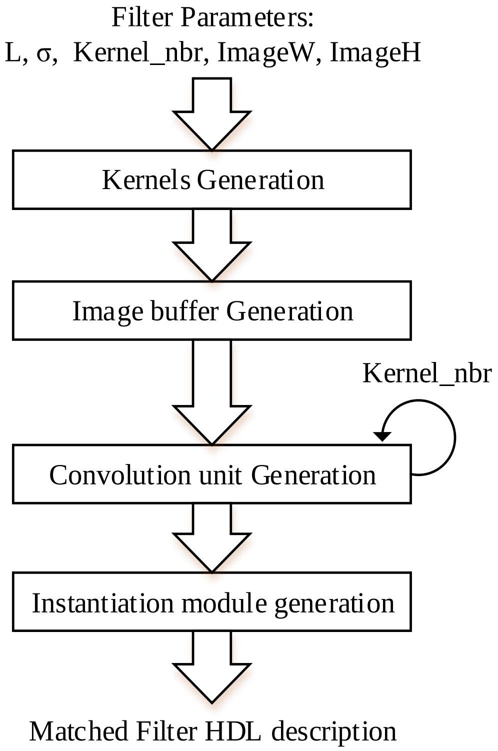

Filter Parameters:
L, σ, Kernel_nbr, ImageW, ImageH
Kernels Generation
Image buffer Generation
Kernel_nbr
Convolution unit Generation
Instantiation module generation
Matched Filter HDL description

Figure 4-6. Matched filter generation steps

The throughput of the unit is maximized by fully pipelining the adder tree as shown in Figure 4-5. Replacing the multiplication by left shifts and additions doesn't affect the segmentation performances and gives exact results because the coefficients are integers. For example, Figure 4-7 shows how a multiplication by the coefficient 6 can be done with two shifts and one addition.

The last step is the generation of the instantiation module that incorporates the different modules. File generation time is negligible, while architecture synthesis time depend on the area of the circuit. For 12 kernels of 27 × 27 pixels, synthesis time is about 80 minutes using XST synthesizer of Xilinx on a 3.4 GHz i7 processor with 16 GB of RAM.

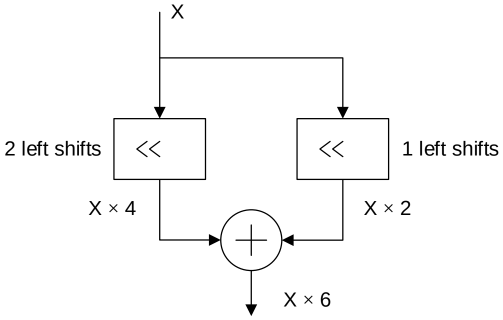


Figure 4-7. Replacing multiplication by left shifts and additions

4.3.2 Proposed ASIP for matched filter algorithm

In this section, we describe the proposed ASIP for retinal blood vessel segmentation. ASIPs offer solutions to trade-off flexibility and efficiency. They have the capability to extend the instruction-set of a processor with a set of customized instructions to gain in performance for a specific application. The proposed ASIP is based on the Xtensa extensible processor. The Xtensa LX2.0 processor is configurable in terms of pipeline length and cache. In addition, the Xtensa is extensible allowing us to extend the processor by defining application specific instructions.

Based on the basic architecture of the Xtensa processor, we proceeded to a first architectural exploration to define the best cache memory configuration. Table 4.1 shows some of the parameters and associated legal values available. As a second step, we profiled the code on the basic Xtensa architecture with the selected cache memory parameters. Based on the profiling results, we identified two major bottlenecks. The first bottleneck is the convolution operation that takes 83% of the total cycles. The second bottleneck is the data access to get the pixels of the window to be convolved with the kernel, this step takes more than 7% of the total cycles.

One of the important features of the Tensilica is the ability to profile the application code, to analyze and generate custom instructions automatically with the XPRES tool. The generated custom instructions are named Tensilica Instruction Extension (TIE). Before proceeding to the design of custom instructions, we generated the TIEs automatically using XPRES of Tensilica. The tool proposed to generate 15 TIEs.

Table 4.1. Parameters of the basic architecture of the Xtensa processor

Parameter	Value
Pipeline length	7
Core speed	312 MHz
Instruction/Data cache size	32Kb
Instruction/Data cache line size	64 bytes
Instruction/Data cache associativity	4

To accelerate the execution of the retinal blood vessel segmentation application, we designed and added two custom instructions to the basic processor as shown in Figure 4-8. The datapath of the processor was modified to include the two added instructions. These two custom instructions were designed to deal with identified bottlenecks: the problem of data access and arithmetic computations of the convolution operation. To reduce the time of access to the kernel coefficients, we decided to store them in a table near the processor.

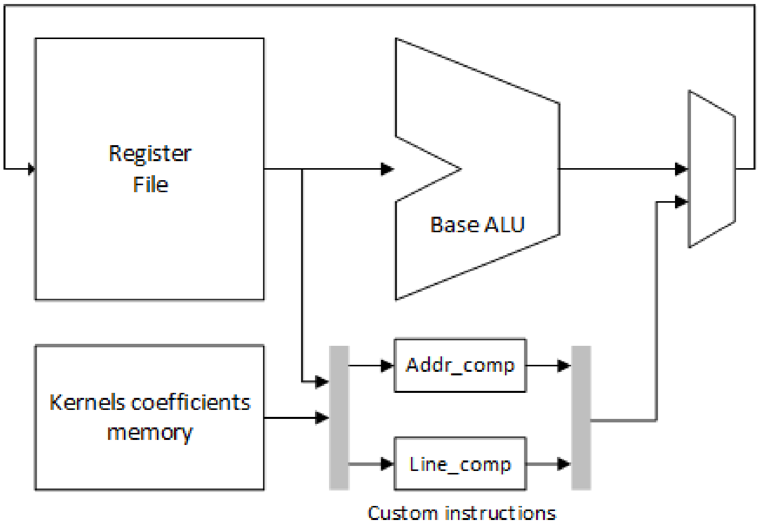


Figure 4-8. Datapath of the proposed ASIP

4.3.2.1 Custom instruction Line_comp

To accelerate the convolution operation, which constitutes the main bottleneck of our application, we added an instruction called Line_comp (Line compute). The role of this instruction is to perform a convolution between one line of the kernel coefficients and corresponding line of pixels in a window as shown in Figure 4-9. All the multiplication operations are realized by shifts and adds to minimize the area needed for the instruction. Using this instruction, computing the convolution between a kernel of N×N coefficients and an N×N pixels window is reduced to a single operation performed N times instead of N×N operations.

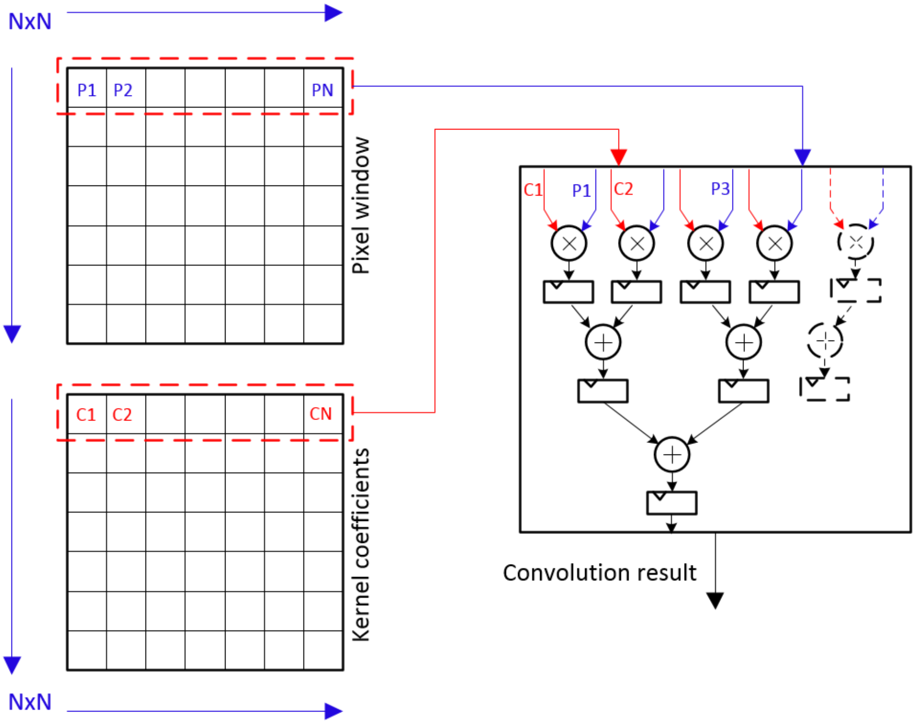


Figure 4-9. The Line_comp instruction architecture

4.3.2.2 Custom instruction Addr_comp

To deal with the problem of data access, a second custom instruction was added to the processor called Addr_comp (Address compute) instruction. At each cycle, the processor needs to compute the address of the needed pixels to realise the convolution operation with the kernel coefficients. Instead of computing the address for each pixel at each cycle, the Addr_comp instruction does it once for all N×N pixels of the window. The input of this instruction is the central pixel's coordinates, and the output are the addresses of the N×N pixels that correspond to the kernel window. This group of addresses is stored in a special table inside the instruction's datapath. To read the data from the memory, we read at each cycle one address that contains four pixel values. The number of read pixels for each kernel line is a multiple of four. When N is not a multiple of four, the last pixel is discarded and not considered for later computations. For example, when N=11, we access three addresses, each with four pixels. The 12th pixel is not considered. The number of loaded pixels is limited by the memory data bus to 32 bits.

4.4 Matched filter implementation results

Before starting the presentation of our results, we explain how the experiments are conducted and validated. The matched filter architecture was implemented on FPGA and on ASIP. For the FPGA implementation, we synthesized and implemented the architectures in a Xilinx Kintex-7 (XC7K480T-1FFG1156). For the ASIP implementation, the matched filter algorithm was described in C++ to target the Xtensa processor. Table 4.2 shows results of blood vessel segmentation quality. Area under the curve and accuracy quality measures are computed for DRIVE dataset as described in section 2.3 and presented in the table. The mean values for all images are also presented. The implemented matched filter is able to segment the blood vessels with a mean accuracy of 92.18%.

Figure 4-10 illustrates a sample of retinal image with its corresponding mask, and Figure 4-11 illustrates a sample of a green channel of a retinal image with the matched filter response as implemented in FPGA and in the ASIP. For the first solution implemented on FPGA, we studied the effect of the number of kernels on the FPGA resources utilization and the maximum frequency of the designed circuit (for the first architecture). As shown in Figure 4-12 and Figure 4-13, when the number of kernels is increased, the FPGA resources increase also. This can be explained by the fact that the added kernels require more resources to realize the implied convolution units. On the other hand, the maximum frequency is decreased each time is added additional kernel. Each added kernel means an additional input to the Max Selector unit, which contributes to make its critical path longer, thus, the frequency is reduced. This could avoided using a pipelined Max Selector unit.

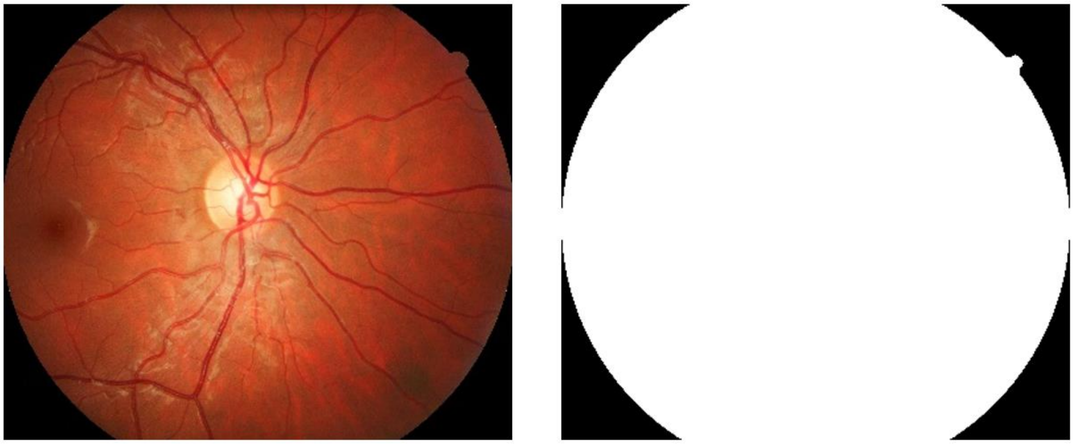


Figure 4-10. Original image and the corresponding mask

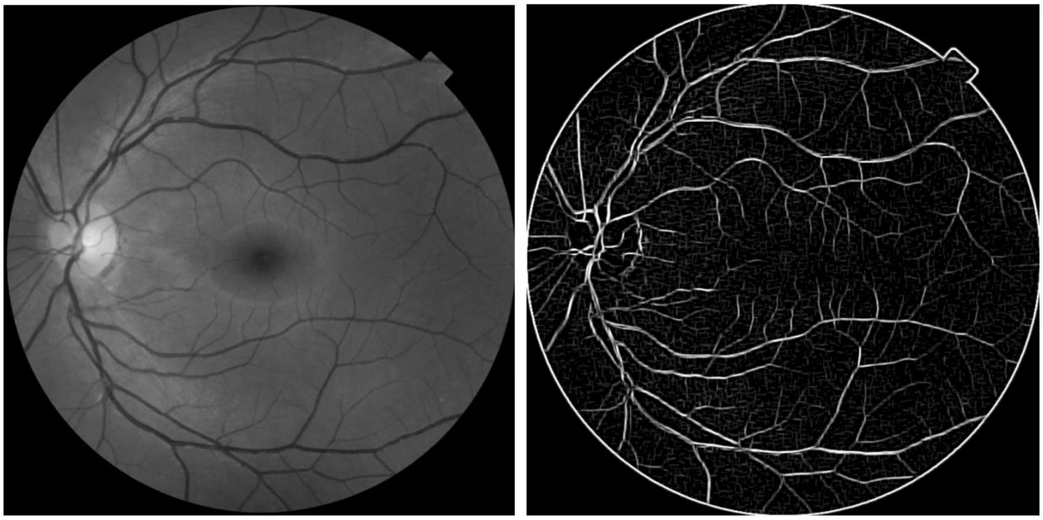


Figure 4-11. Green channel of a retinal image with the matched filter response

Table 4.2. Quality measures of blood vessel detection using DRIVE dataset.

No	ACC	AUC	No	ACC	AUC
1	0.9278	0.9362	11	0.9080	0.9123
2	0.9266	0.9153	12	0.9127	0.9127
3	0.9192	0.9132	13	0.9151	0.9149
4	0.9149	0.9152	14	0.9324	0.9187
5	0.9125	0.9143	15	0.9336	0.9322
6	0.9250	0.9167	16	0.9122	0.9174
7	0.9286	0.9246	17	0.9252	0.9187
8	0.9151	0.9232	18	0.9234	0.9305
9	0.9182	0.9273	19	0.9348	0.9242
10	0.9234	0.9262	20	0.9276	0.9212
			Mean	**0.9218**	**0.9207**

Figure 4-14 and Figure 4-15 illustrate the effect of the kernel size on resources utilization and the maximum frequency, respectively, for the same number of kernels (12 kernels). As can be seen in Figure 4-14, when the kernel size is increased, the FPGA resources are also increased. We can explain this by the fact that larger kernels require more multiplications and additions to realize the convolution operation between the pixel window and the kernel coefficients, thus, more FPGA resources are required to realize the Convolution Unit. As shown in Figure 4-15, the size of the kernel implies a slight decrease of the maximum frequency. This is expected since the Convolution Unit is pipelined, and the slight decrease may be due to the imbalanced pipeline stages.

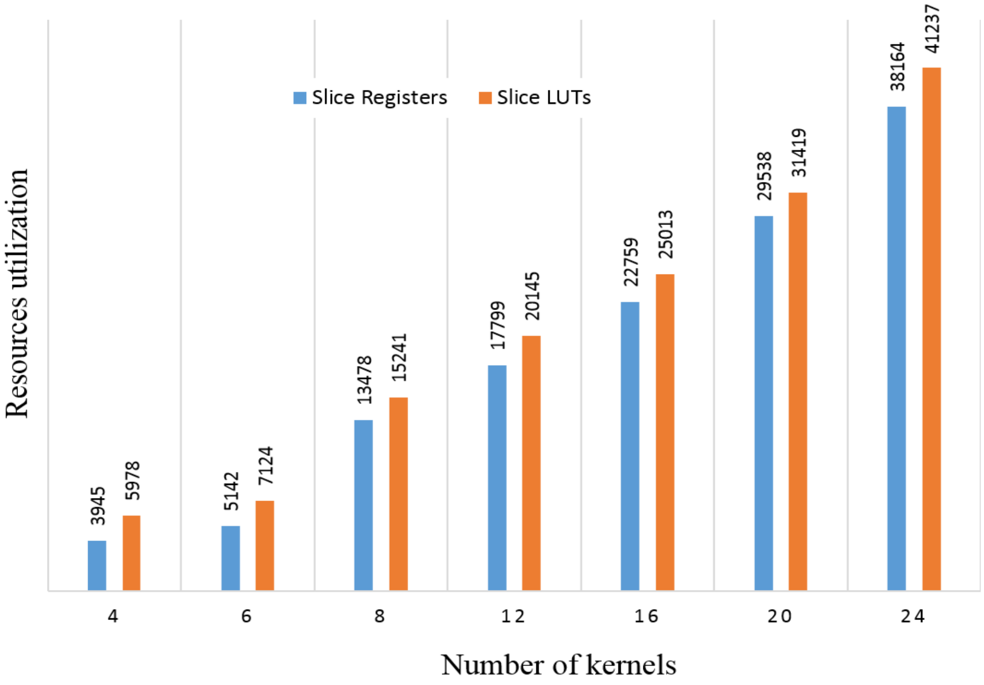


Figure 4-12. FPGA resources utilization as a function of the number of kernels of size 15 × 15

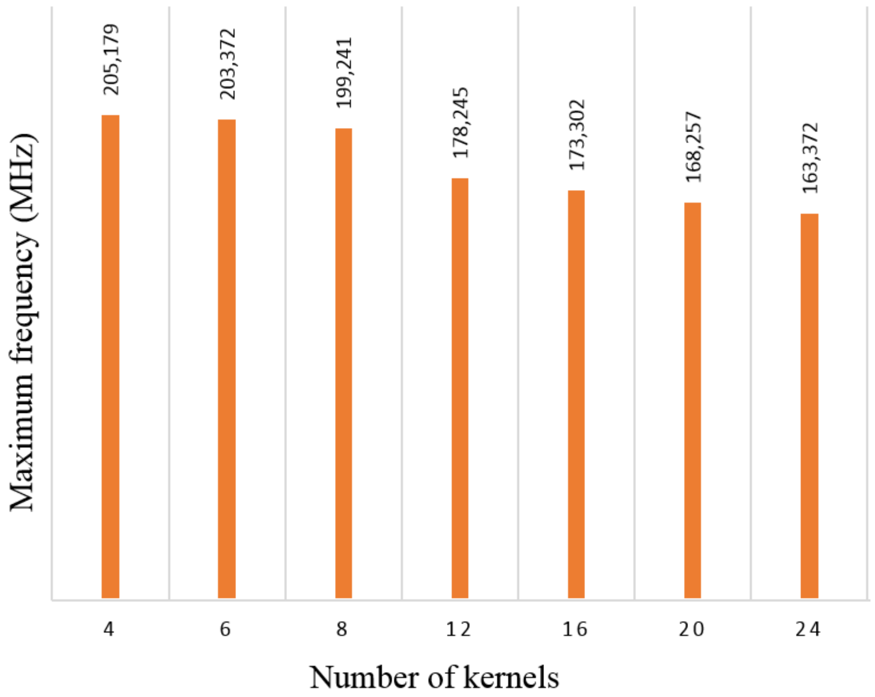


Figure 4-13. FPGA maximum frequency as a function of the number of kernels of size 15 × 15

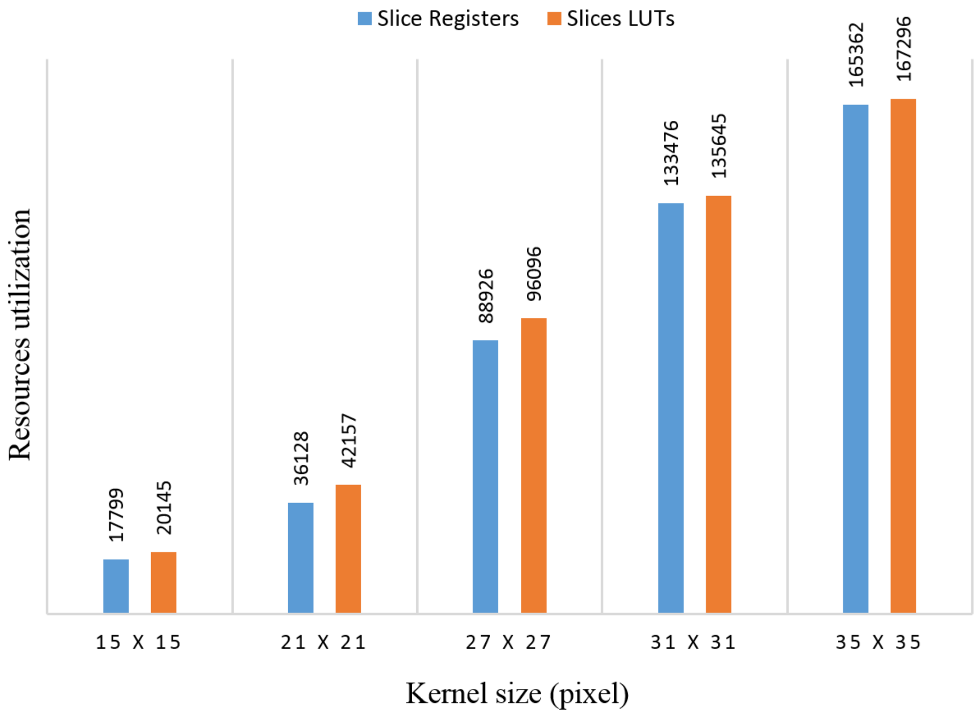


Figure 4-14. FPGA resources utilization as a function of kernel size (for 12 kernels)

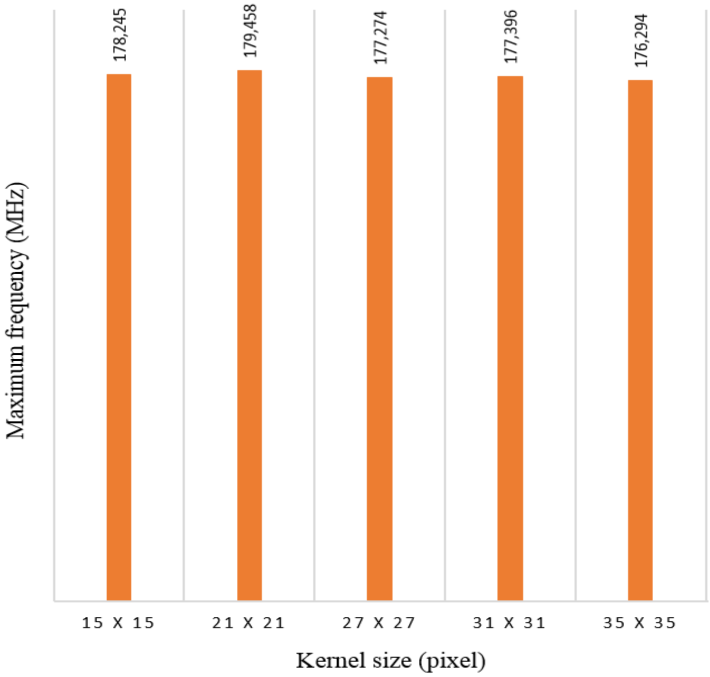


Figure 4-15. FPGA maximum frequency as a function of kernel size (for 12 kernels)

Figure 4-16 shows the estimated number of cycles needed to execute the matched filter program in the Xtensa processor for different configurations. These configurations are: base_conf (Xtensa processor without TIE), Xtensa processor with the different automatic generated TIEs and the Xtensa processor with the designed TIEs. This figure shows that the TIE_10 reduces the number of total cycles from 13.8×10^8 cycles to 4.9×10^8 cycles. This reduction represents a speedup of 2.78×. For the TIE_2, the reduction is even greater and the number of total cycles passes from 1.38 × 109 cycles to 4.8 × 108 cycles, which represents a speed-up of 2.89×. The TIE_2 represents the best solution generated automatically in terms of number of total clock cycles. Our custom TIEs achieve the best speedup overall, by a factor of 7.78× over the base configuration.

Figure 4-17 gives a comparison between all the automatically generated TIEs and the designed TIE for additional area and speedup factors. The automatically generated TIEs doesn't exceed a speedup factor of 3×, while the additional area is still relatively low (less than 0.65 KGates which represents 0.72× of the basic processor area). The designed custom TIEs consume more additional gates when compared to the other TIEs. The area of the designed ASIP with the two custom instructions is 4.3× the area of the basic processor, this represents a real gain of 1.8× area/speed.

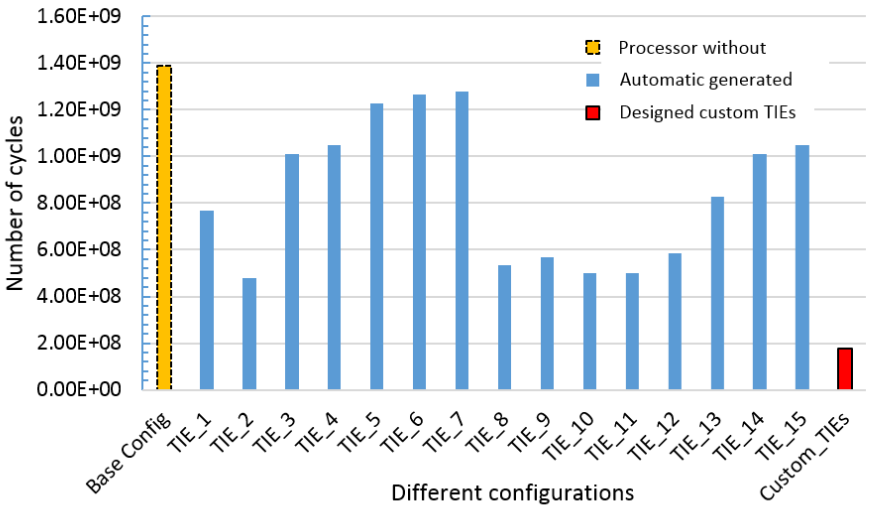


Figure 4-16. Number of cycles for the Xtensa processor for each TIE

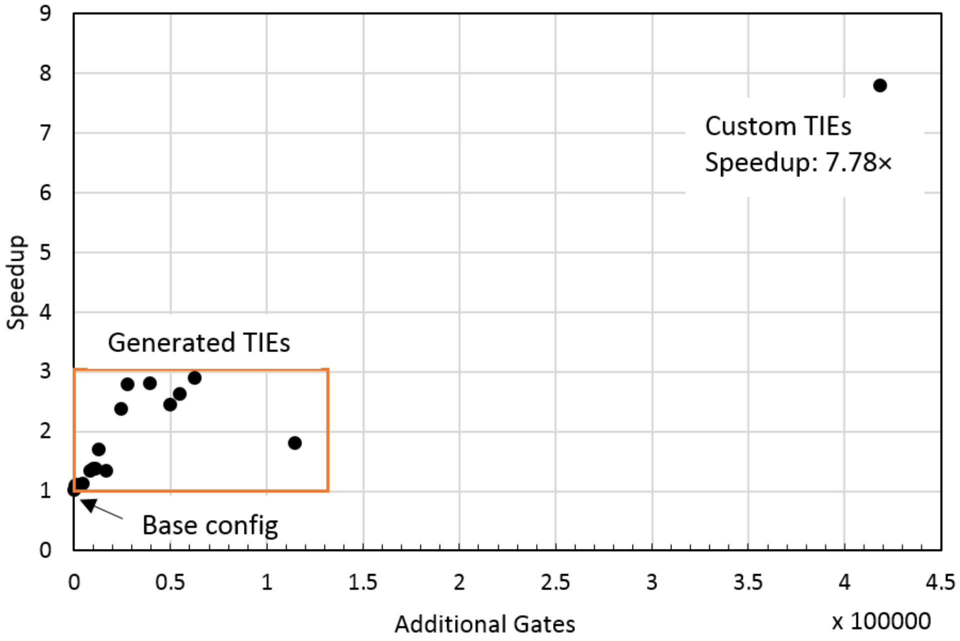


Figure 4-17. Comparison between the designed TIE and the generated TIEs in terms of Speed-up and additional gates

Table 4.3 shows the estimated efforts for the design and verification of the two architectures with three approaches. For the hand-coded hardware design, the effort is very important (105 man-hours). The ASIP design requires a significant effort (40 man-hour) with more flexibility in case of changes. This flexibility is due to the software nature of the application. The designed TIEs are described in a language very similar to Verilog. If we decide to change the parameters of the matched filter, we should modify the description of the TIEs. The most productive option however is with the automated generation tool proposed in this paper, since producing an architecture from any set of parameters requires less than one hour. The verification effort for the three approaches follows a similar pattern.

Table 4.3. Estimated design and verification efforts

Architecture	Design effort (man-hour)	Verification effort (man-hour)
Hardware (hand coding)	105	45
ASIP	40	6
Hardware (with tool)	< 1	< 1

4.5 Verification of the matched filter designs

To verify the correctness of the produced results by the matched filter architectures, we compared them with a bit-accurate reference MATLAB models. Each pixel of the output image was compared to the corresponding pixel of the model. The verification process proved the correctness of the produced results by our proposed architectures.

4.6 Conclusion

In this chapter, two architectures for the retinal blood vessel segmentation are proposed, designed and implemented. The first architecture is a scalable hardware architecture based on the matched filter algorithm. To avoid hand coding when selecting the algorithm parameters, a software tool was developed to generate an HDL description of the matched filter automatically from a set of parameters, and based on the optimized proposed architecture as template. In this work, we targeted high-resolution fundus images and the achieved improvements over the state-of-the-art implementations can reach a factor of 14 × for 4288 × 2848 image size. We evaluated the effects of the matched filter parameters on FPGA resources utilization and maximum clock frequency. The second architecture is based on an application-specific instruction-set extension for a Tensilica Xtensa LX ASIP. With only two additional custom instructions requiring an additional 4× the area of the basic processor, the ASIP achieved a significant speedup of 7.76× when compared to the basic processor, while retaining all its flexibility. An estimated man-hour efforts for the design and verification of the two architectures show that the hardware architecture necessitates more efforts than the ASIP for the design and the verification. For future works, we will focus on the integration of pre- and post-processing algorithms to evaluate the performance of the developed system.

CHAPTER 5 MEMORY-EFFICIENT ARCHITECTURE FOR RETINAL BLOOD VESSEL SEGMENTATION

In this chapter, we present a novel architecture for retinal blood vessel segmentation based on the MSLD algorithm. The proposed architecture was made memory-efficient to deal with the algorithm requirements in terms of memory requirements of the algorithm. We propose a specific compiler able to generate HDL descriptions of the MSLD algorithm automatically from a set of its parameters. We also present and discuss the results of the proposed solution. The implemented architecture was evaluated in terms of execution time, FPGA resources utilization, maximum clock frequency and blood vessels segmentation quality.

5.1 Introduction

In this chapter, we present a hardware architecture for Multi-Scale Line Detector (MSLD) algorithm for retinal blood vessels detection. The MSLD algorithm was chosen because of its popularity and performances in terms of segmentation quality. The MSLD algorithm was also chosen because of its potential and ability to detect small vessels [56]. However, the MSLD can be memory intensive because intermediate results from multiple scales must be stored. This can become a serious impediment in the case of high-resolution images which are now the norm with modern retinographs. The algorithm was made hardware friendly to achieve significant performances. The MSLD memory requirements problem was also considered [8]. Our proposed solution is based on computation reuse and parallel implementation of the computations at each scale. The architecture is optimized in terms of resources utilization and throughput. HDL description of our hardware architecture is generated automatically using a developed tool that takes as input the algorithms parameters. Our tool makes the algorithm parameters selection more flexible and generates a specific HDL description based on an optimized scalable architecture template. In this chapter we make the following contributions:

- Design of a memory-efficient architecture for the MSLD algorithm.
- A specific compiler is introduced to automatically generate optimized low-level hardware descriptions, suitable for FPGA implementation, from any algorithm set of parameters.
- A GPU implementation is also proposed and compared to the FPGA implementation.
- Comprehensive results are provided in terms of blood vessel segmentation quality with

respect to computations accuracy.

5.2 MSLD algorithm description

The MSLD algorithm was proposed by Nguyen el al. [45] as a generalized case of the line detector first proposed by Ricci et al. [52]. The basic line detector is applied to the inverted green channel where the retinal blood vessels appear brighter than the background. For each pixel of the image, we consider a window of $W \times W$ pixels and the average gray level is computed as I_{avg}^{W}. We consider also twelve lines of length W pixels centered on the pixel to process and oriented in 12 directions as shown in Figure 5-1. The average gray level is computed along each line, and the maximum value is defined as I_{max}^{W} and the line response at this pixel is then computed as:

$$R = I_{max}^{W} - I_{avg}^{W} \qquad (5.1)$$

To improve the line detector for retinal blood vessel detection, Nguyen el al. [45] proposed the generalized line detector that works at multiple scales. The MSLD is based on varying the length of the aligned lines and (5.1) is redefined as:

$$R_{W}^{L} = I_{max}^{L} - I_{avg}^{W} \qquad (5.2)$$

where $1 \leq L \leq W$, I_{max}^{L} and I_{avg}^{W} are defined as above. Line detectors at different scales are achieved by changing the value of L.

For each scale, we standardize the values of the raw response image to make them have zero mean and unit standard deviation distribution. The main purpose of the standardization is to achieve better contrast between the blood vessels and the retinal image background. The standardization is defined as:

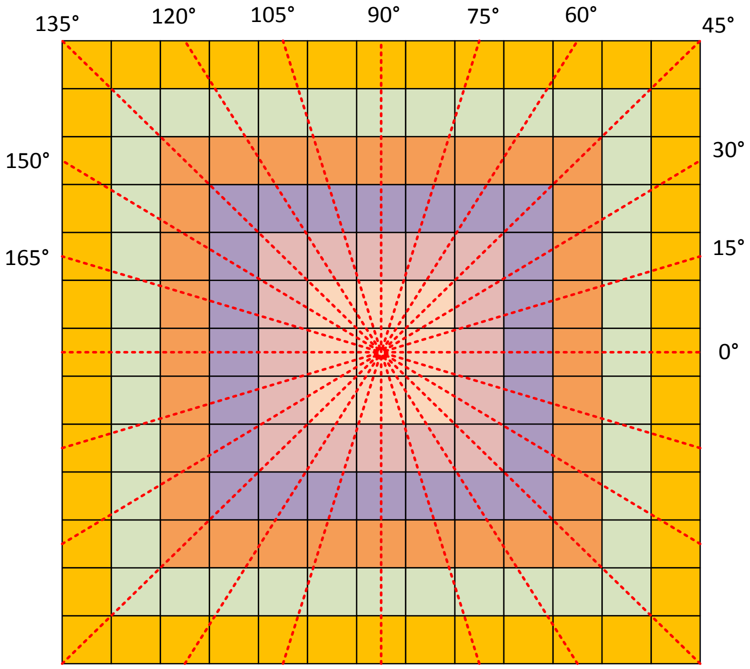


Figure 5-1. A 13 × 13 window with its different scales and orientations

$$R' = \frac{R - R_{mean}}{R_{Std}} \qquad (5.3)$$

where R' is the standardized response value, R is the raw response value, R_{mean} and R_{Std} are the mean and standard deviation of the raw response values, respectively.

The response at each image pixel is the linear combination of the line responses of different scales, defined as:

$$R_{Combined} = \frac{1}{n_L + 1}\left(\sum_{L} R_W^L + I_{igc}\right) \qquad (5.4)$$

where n_L is the number of scales, R_W^L is the response of the line detector at scale L and I_{igc} is the value of the inverted green channel at the corresponding pixel.

5.3 Proposed MSLD memory-efficient architecture

Before presenting the proposed architecture, we show how the architecture was made memory-efficient. Next, the proposed architecture is presented from pixel input to the output

MSLD response where all the different modules of the architecture are described. Figure 5-2 shows an overview of the proposed architecture.

5.3.1 Memory-efficiency

Figure 5-3 shows the CPU and custom architecture flow details. For the CPU flow, once the image and mask are acquired, the raw response computation can be started. For each value of the raw response, the computation of the mean and standard deviation values is started. The raw response for scale i is stored in memory before the standardization step. The standardization requires the value of the raw response at each pixel, the mean value and the standard deviation for the scale i. The standard response is then computed. At the same time, the combined response is computed by accumulating the standardized responses. Consequently, we store the new combined response once a new scale is processed. The final combined response is obtained once the last scale is processed.

The software implementation of the MSLD algorithm is memory intensive. CPUs typically cannot process all scales in parallel, and, thus, the processing is done in a sequential manner. For each scale, the intermediate results (the raw response and the combined response) must be stored in memory. The amount of memory necessary is thus equal to 2× the size of the original image as shown in Figure 5-3.b.

The custom parallel flow avoids the use of external memory such as in [120, 121], and does not store the intermediate responses of the different scales. Instead, it computes the raw responses twice. For the first run, the mean and standard deviation values are computed and stored. For the second run, the raw responses are computed again, which allows standardization of the raw responses on the fly using the stored mean and standard deviation values without the need to re-compute them. The custom parallel implementation takes advantage of the parallelism allowed. A fully parallel data path is a suitable solution to handle the different scales and to re-compute the raw responses twice very swiftly. The custom parallel implementation shrinks the memory requirements from 2 images to 2 × number of scales values.

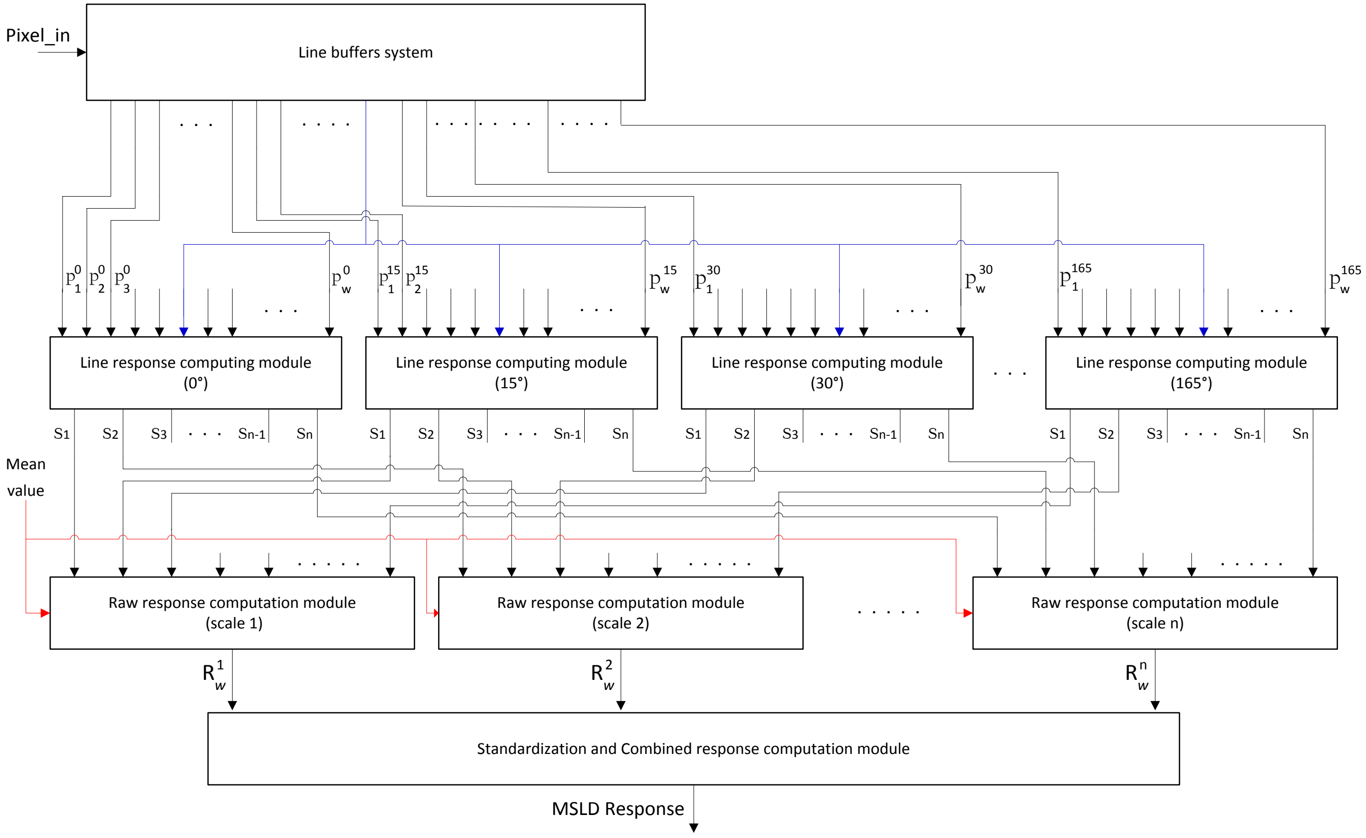
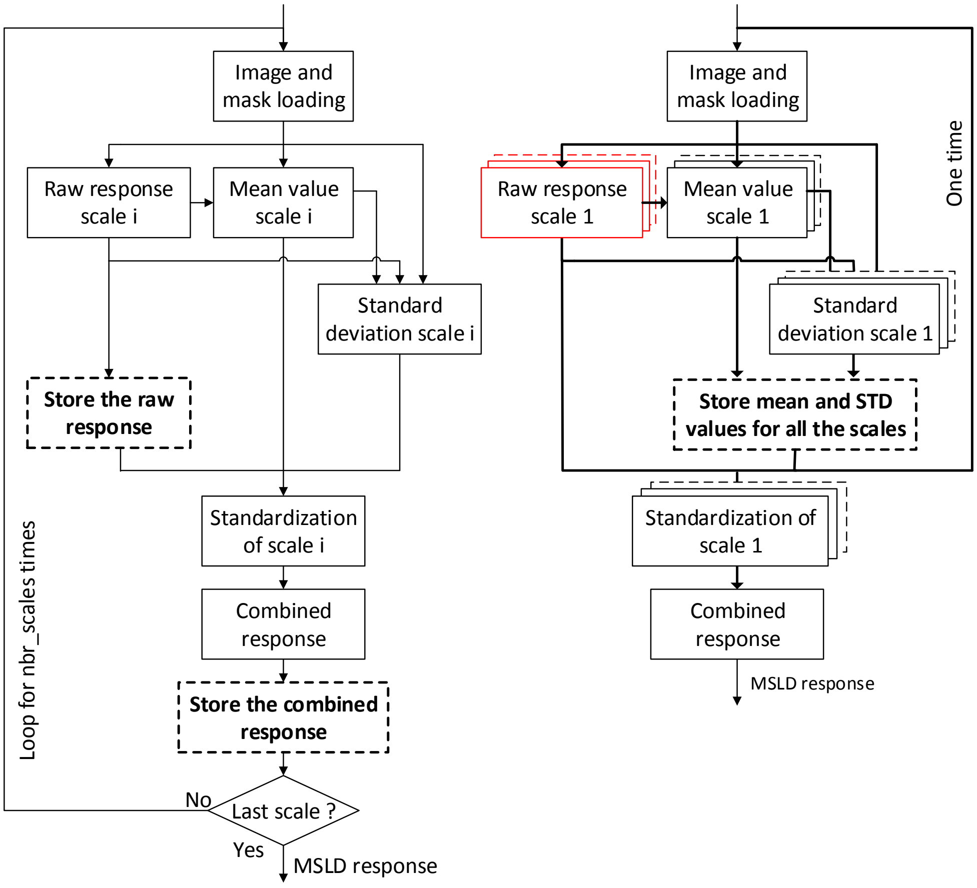

Pixel_in
Line buffers system
p_1^0
p_2^0
p_3^0
p_w^0
p_1^{15}
p_2^{15}
p_w^{15}
p_1^{30}
p_w^{30}
p_1^{165}
p_w^{165}
Line response computing module (0°)
Line response computing module (15°)
Line response computing module (30°)
Line response computing module (165°)
S1
S2
S3
Sn-1
Sn
Mean value
Raw response computation module (scale 1)
Raw response computation module (scale 2)
Raw response computation module (scale n)
R_w^1
R_w^2
R_w^n
Standardization and Combined response computation module
MSLD Response

Figure 5-2. MSLD architecture overview



a. b.

Figure 5-3. a. CPU and b. Custom parallel architecture flows

5.3.2 Line buffers

The line buffers provide data to the rest of the MSLD architecture. The modules require access to the image pixels corresponding to the window centered on the pixel to be processed. For this purpose, a shift register of length $(W-1)\times Ncols+W$ is used, where $Ncols$ is the number of columns of the retinal image. Since the MSLD works on the inverted green channel, the pixels of the image are subtracted from 255 before being pushed to the line buffer. All pixels inside the window are needed to compute the mean value of the window. Only the pixels that correspond to the line detector for the twelve different orientations are fed to the modules that compute the line response. Figure 5-4 shows the structure of the line buffers to access the pixels in parallel inside the window. This figure shows an example of a 5×5 window.

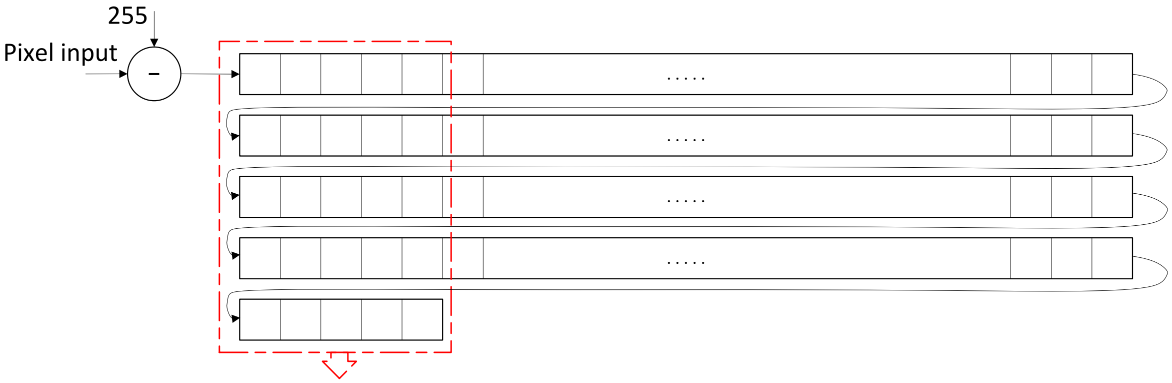


Figure 5-4. Structure of the line buffers for window pixels access

5.3.3 Line Response Computing Module (LRCM)

This module is responsible of computing one line response corresponding to one orientation. The inputs of this module are the W pixels corresponding to that orientation. Along a given line and for each scale, we compute the mean value of the intensity level of the pixels. The outputs of this module are the line responses at the different scales. As can be seen in Figure 5-2, the MSLD architecture uses twelve parallel LRCMs.

Figure 5-5 shows the architecture of the LRCM block. This figure shows how the pixels of one line (from a window of 11 × 11 pixels) are arranged to realize a fully pipelined adder tree. The output of scale 1 is used to compute the result for scale 2 and so on. In this way, the computations are reused to reduce the number of functional units. At each stage of the pipeline, the adders are tailored for the exact word-length of the outputs. The output of each scale is multiplied by the corresponding reciprocal coefficient to realize the division by the number of pixels and then compute the mean value. By reusing the computations and word-length optimization, resource consumption is reduced. Pipelining reduces the critical path and increases throughput.

5.3.4 Raw Response Computation Module (RRCM)

For each pixel of the window, the line response is computed for the twelve different orientations and at different scales. The goal behind this operation is to compute one raw response at each scale. For this purpose, the outputs of the twelve LRCM blocks, as shown in Figure 5-2 are routed to the six RRCM (the number of scales equals $(W + 1)/2$, so for an 11 × 11 window there are six scales). The RRCM blocks are responsible for computing the raw response for one scale. A

RRCM block has twelve inputs that correspond to one scale but from different lines and one input for the mean value inside the window. As can be seen in Figure 5-2, the RRCM for the scale 1 has for inputs the s_1 outputs of the twelve LRCM blocks and the mean value of the window (in red color).

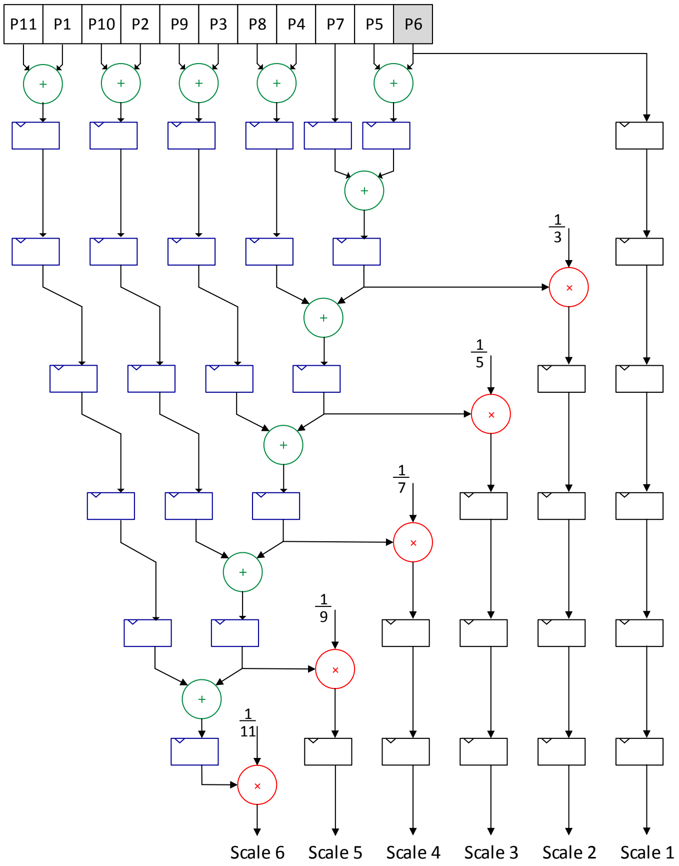


Figure 5-5. Line response computing module

Figure 5-6 shows the architecture of an RRCM block. The response of each scale is the difference between the mean value of the pixels inside the window and the maximum response of the twelve lines. The mean value computation module is designed as a fully pipelined tree with optimized word-length optimized to increase the frequency and reduce computation resources, respectively. Finding the maximum response of the twelve lines is realised by comparing the inputs two by two in parallel and in a pipelined way as shown in Figure 5-6.

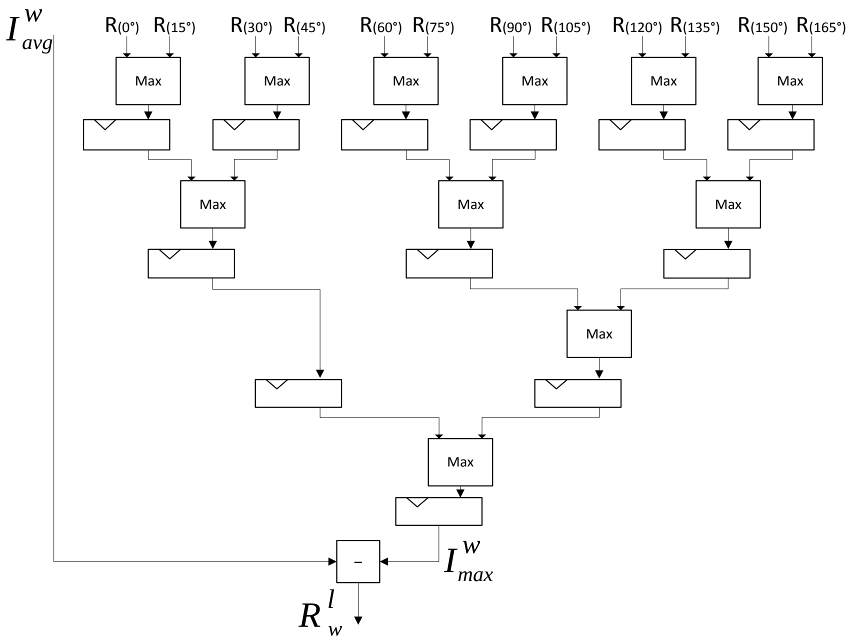


Figure 5-6. Raw response computation module architecture

5.3.5 Mean and standard deviation values computation

Mean and standard deviation values are necessary for responses standardization. They are computed in a streaming way, in parallel with the raw response computation. To compute the mean value, the intensity values of the raw responses that correspond to the ROI (Region Of Interest) are accumulated. At the same time, the number of accumulated values is counted since the ROI is circular and the number of pixels inside it is unknown. The ROI is defined by the mask by a one bit signal. The mean value is computed by computing the sum of the pixel values and at the last pixel of the image, the sum is divided by the number of pixels to get the final mean value. The variance value is also computed in a similar manner.

The square of the intensity of the raw responses is computed and then accumulated when the pixel is inside the ROI. To compute the standard deviation, the square root of the variance is computed as shown in Figure 5-7. The red part shows the computation of the number of pixels inside the ROI. The blue part shows the mean computation, and the green part shows the standard deviation computation. The mean and standard deviation are computed for each scale. Until this step, all the computations are done in a streaming manner and in parallel for all the scales without saving the raw responses. Instead, we save the mean and standard deviation for all the scales.

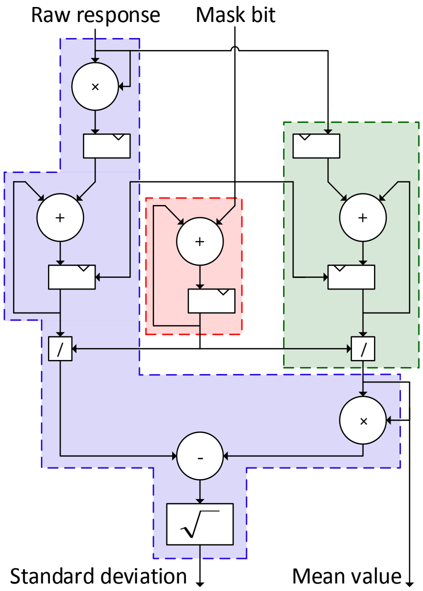


Figure 5-7. Mean and standard deviation values computation module

5.3.6 Standardized and Combined Response Computation

The mean and standard deviation values are computed to standardize the raw responses. The standardization step necessitates the raw response and the mean and standard deviation values at the same time. This can be a challenging problem because of the requirements for large on-chip memory to store the raw responses while computing the mean and standard deviation values. To overcome this problem, we propose to save the mean and standard deviation values for the different scales, which results in very low memory requirement in comparison with the raw responses saving requirements. Once the mean and standard deviation values are computed and stored, the computation of the raw responses is restarted for a second time without re-computing the mean and standard deviation values since they were already computed and stored.

For this second pass, each processed pixel for all the scales is standardized and combined on the fly according to (5.3) and (5.4). Figure 5-8 shows the principle of re-computing the raw responses to avoid saving them and to reduce the memory requirements. As can be seen in Figure 5-8, in the first pass the raw responses, the mean and standard deviation values are computed. The objective is to avoid saving the raw responses since the on-chip memory is not sufficient to save such a large amount of data. For an 11×11 pixels window, six scales are handled in parallel, and the memory requirements are 6× greater than the original image if stored.

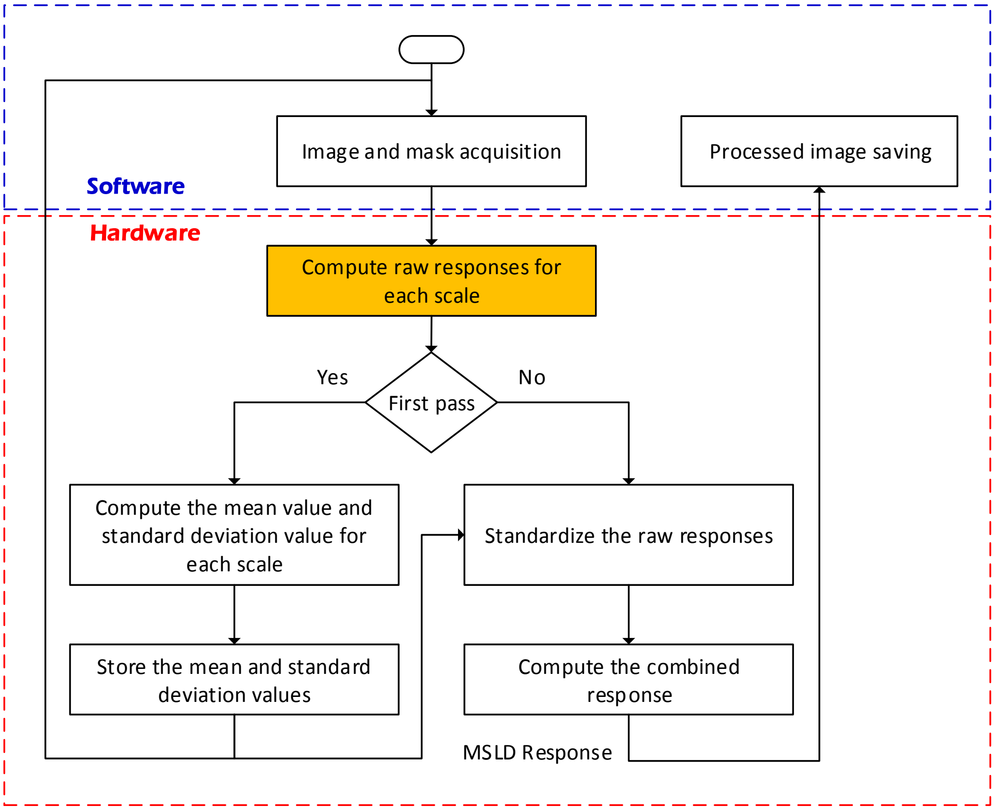

Software
Hardware
Image and mask acquisition
Processed image saving
Compute raw responses for each scale
First pass
Yes
No
Compute the mean value and standard deviation value for each scale
Standardize the raw responses
Store the mean and standard deviation values
Compute the combined response
MSLD Response

Figure 5-8. Principle of re-computing raw responses to avoid saving them

Figure 5-9 shows the architecture of the standardization and combined response computation module. For this purpose, we use six standardization modules, one for each scale. The inputs of the standardization modules are the newly computed raw response of the corresponding scale, the mean and standard deviation values.

5.3.7 Operations scheduling

Operations scheduling is a vital task for the functioning of the entire system. Since the number of inputs of each scale is variable, the number of pipeline stages is also variable. For scale 1, only the center pixel of the window is needed, while scale n needs W bits to compute the line response. As shown in Figure 5-5, the pixel values are added two by two with full pipeline. The scale 1 response is computed in the first stage of the pipeline, the scale 2 response is computed after two pipeline stages and so on for the other scales responses.

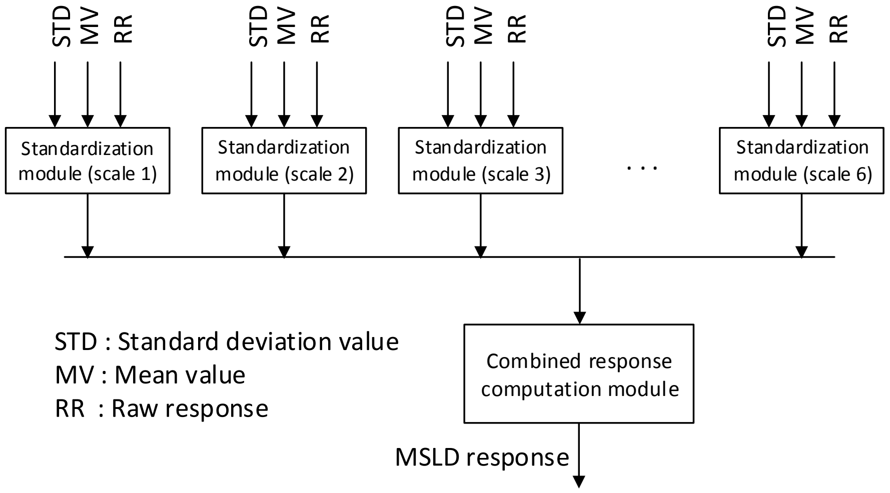

STD
MV
RR
Standardization module (scale 1)
Standardization module (scale 2)
Standardization module (scale 3)
. . .
Standardization module (scale 6)
STD : Standard deviation value
MV : Mean value
RR : Raw response
Combined response computation module
MSLD response

Figure 5-9. Combined response computation architecture

To properly compute the raw responses at each scale, the outputs of the LRCM module must be synchronized and sent at the same time to the RRCM modules. For this purpose, registers are added to the outputs of the scales to balance the pipeline and to delay the first computed responses to get them out at the same time with result of the scale n (largest scale). This can be seen in Figure 5-5, where registers are connected to the output of the multipliers. For example, the scale 1 response is delayed by six clock cycles using six registers. For scale 2, four registers are added, and the outputs are fully synchronized.

This is not the only place where registers are added to synchronize the outputs. The RRCM module needs the mean value of the window, where the window contains a large number of pixels (W^2 pixels). The number of stages of the RRCM module and the LRCM module is different, and then the first computed outputs should be delayed to meet the outputs of the second module. This is achieved by adding the necessary number of registers to balance the two pipelines.

5.3.8 MSLD architecture scalability

In all figures we considered a generic architecture except for Figure 5-4 and Figure 5-5 where we considered a window size of 5×5 and 11×11 pixels, respectively, for clarity but without loss of generality. Since the MSLD parameters (window size, the line rotation step and the number of scales) all depend on the image size, it is crucial to be able to scale the architecture easily to the new parameters when changing the image size. Writing a generic architecture of the MSLD algorithm in a hardware description language is a tedious task. In addition, hand coding of the MSLD architecture is very time consuming and hard to modify when modifying the algorithm parameters. The MSLD algorithm is very computationally intensive with complex functional units. To increase the throughput and to reduce resource consumption, computations reuse and deep optimizations using low-level programming are necessary. A trade-off between design time, deep hardware optimizations and architecture scalability must be considered. To solve these problems, a specific compiler for the MSLD algorithm was designed. The compiler is able to generate low-level hardware description of the algorithm for any set of parameters.

The compiler accepts as input the parameters of the MSLD algorithm. A high-level description describes the different functional units of the algorithm as functions:

- The line buffer function which accepts as input the image width and the size of the window.
- The adder tree with mean value computation of the window. This function accepts as parameters the number of input pixels and the precision of the mean value.
- The comparator tree function which finds the maximum value. The input parameters of this function are the number of inputs and the function to realize (max or min).
- The line response computing function. This function evaluates, for each line, which pixels must be accessed for a given orientation and then computes the response at different scales. The input parameters of this function are the orientation and the precision of the response computation. This function is called twelve times to generate the different modules for the twelve orientations.
- The raw response computing function, which computes for each scale the raw response of the MSLD. This function accepts as parameters the number of scales and the orientation. This function is called twelve times to generate the different modules for the twelve orientations.
- The standard deviation and mean value computing function. This function is called many times to generate the standard deviation and mean values computation modules for each scale.
- The sequencer function. This function generates the state-machine that synchronizes the different modules of the architecture. It computes also the number of pipeline stages of each module to be able to schedule the different modules tasks.

- The wrapper function, which generates a wrapper module to instantiate the different modules.

In addition to the parameters of the functions, each function needs the input and output flow of data. Each function is responsible for the optimization of the module to be generated. The wrapper function instantiates the different modules and generate one core of the MSLD algorithm.

5.4 MSLD implementation results

In this section we present the results of the MSLD algorithm implementation. The MSLD algorithm was implemented on three platforms, CPU, GPU and FPGA. The first implementation on CPU was realized in C++ and implemented on an Intel i7-2600 CPU @ 3.4GHz with 16 GB of RAM. The GPU implementation was described in OpenCL and implemented on an Nvidia Quadro 2000M GPU. The FPGA implementation was coded in VHDL and implemented on a Zynq®-7000 SoC XC7Z020-CLG484-1. We tested our architectures for both high and low-resolution images. Images of low resolution come from the publicly available DRIVE dataset [51]. High-resolution images come from a private database and were collected through the telemedicine platform of Diagnos Inc. In next sections we present the implementations results.

The Zynq-7000 AP SoC that includes a dual-core ARM processing system (PS) with a 7-series Xilinx Programmable Logic fabric (PL) in a single device. This makes it an adequate platform for clinical applications where the PS plays the role of a host to feed the PL with the image and its mask to process them and then receive the processed image. The communication PS-PL is established using the Xillybus core [122], which consists of an AXI bus with a DMA buffer to transfer the data between the two sides. The hardware core of the MSLD is connected to the Xillybus core via FIFOs. The ARM processor runs an embedded Linux operating system and is clocked at 667 MHz.

The first step to process the retinal image is the image and mask loading. The mask is a binary image that defines the ROI that corresponds to the retina. We use a binary image mask with the same size as the retinal image. Figure 5-10 shows an example of an original image and its corresponding mask. Once the image and its mask are available to the Linux OS, a program running on the ARM processor loads them and starts sending them to the PL at the same time. Only the green channel is processed. To ensure the synchronization of image and mask transfer to the MSLD core, we combine the pixel information (8 bits) and the mask information (1 bit) into one word of 16 bits. The traffic from the PS to PL is established for 16-bit data.

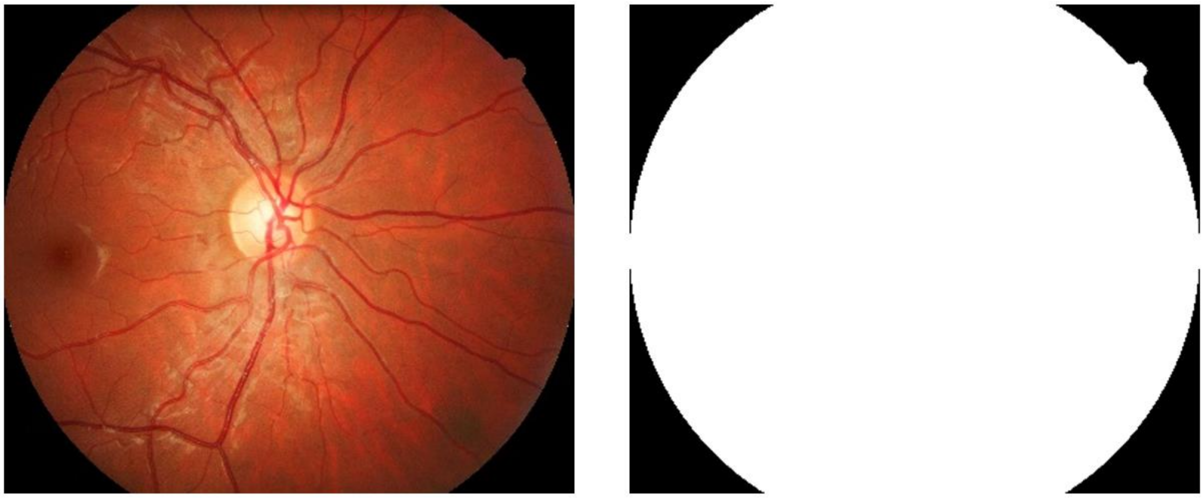


Figure 5-10. Original image and the corresponding mask

Figure 5-11 shows a sample of an original retinal image with its binarized MSLD response. To evaluate the blood vessel detection performance, we used DRIVE dataset a publically available database collected by Staal et al. [51]. To evaluate the algorithms, we use the area under the curve, sensitivity, specificity and accuracy metrics described in chapter 2. Table 5.1 shows the quality measures of blood vessel detection for the CPU and FPGA implementations for 20 images of the DRIVE database. As can be seen in Table 5.1, the quality measures of the CPU implementation are comparable. The difference in results is due to the fixed point computations precision of the FPGA implementation compared to the floating point full precision of the CPU. The FPGA implementation functional units are optimized in terms of word-length. The integer part of the signals are all set to handle the maximum value.

The number of bits of the fractional part is set based on some conducted experiments. Figure 5-12 shows the blood vessel detection quality measures as a function of the number of fractional bits. As can be seen in the figure, using more fractional bits improves the quality of blood vessel detection. However, when using 20 bits, the AUC and ACC are slightly improved while SE and SP are decreased. Using 18 bits for the fractional is an acceptable compromise in terms of precision and FPGA resources utilization. Hence, all our results are shown based on 18 bits for the fractional part.

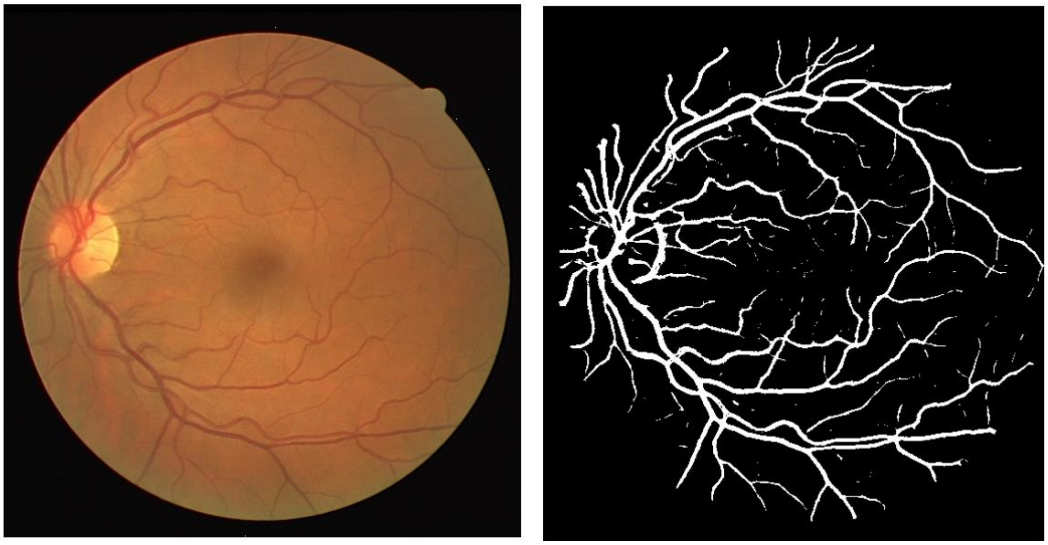


Figure 5-11. Original retinal image and binarized MSLD

Table 5.1. Quality measures of blood vessel segmentation for DRIVE database images – CPU and FPGA implementations

	CPU				**FPGA**			
Image #	**AUC**	**SE**	**SP**	**ACC**	**AUC**	**SE**	**SP**	**ACC**
1	0.9561	0.7942	0.9723	0.9489	0.9582	0.8038	0.9686	0.9470
2	0.9459	0.7408	0.9816	0.9455	0.9459	0.7483	0.9766	0.9424
3	0.9131	0.7152	0.9664	0.9298	0.9129	0.7271	0.9598	0.9259
4	0.9226	0.6654	0.9858	0.9431	0.9254	0.6806	0.9788	0.9391
5	0.9244	0.7030	0.9793	0.9418	0.9265	0.7155	0.9741	0.9390
6	0.9183	0.7007	0.9686	0.9308	0.9169	0.7107	0.9645	0.9287
7	0.9178	0.6883	0.9699	0.9326	0.9209	0.7013	0.9625	0.9279
8	0.9103	0.6678	0.9652	0.9278	0.9098	0.6813	0.9574	0.9227
9	0.9277	0.7301	0.9659	0.9382	0.9300	0.7392	0.9621	0.9360
10	0.9294	0.7345	0.9734	0.9449	0.9320	0.7515	0.9668	0.9411
11	0.9290	0.6958	0.9756	0.9393	0.9269	0.7027	0.9693	0.9347
12	0.9339	0.7667	0.9632	0.9386	0.9341	0.7766	0.9573	0.9347
13	0.9261	0.7061	0.9731	0.9352	0.9289	0.7182	0.9682	0.9327
14	0.9423	0.7884	0.9589	0.9388	0.9439	0.7983	0.9533	0.9350
15	0.9451	0.7739	0.9641	0.9443	0.9454	0.7901	0.9538	0.9368
16	0.9513	0.7470	0.9744	0.9446	0.9503	0.7538	0.9698	0.9416
17	0.9413	0.7552	0.9657	0.9397	0.9408	0.7580	0.9635	0.9382

18	0.9544	0.8026	0.9624	0.9440	0.9537	0.8055	0.9572	0.9398
19	0.9723	0.8579	0.9749	0.9608	0.9714	0.8685	0.9681	0.9561
20	0.9539	0.8161	0.9609	0.9455	0.9530	0.8237	0.9539	0.9400
Total	**0.9358**	**0.7425**	**0.9701**	**0.9407**	**0.9363**	**0.7527**	**0.9643**	**0.9370**

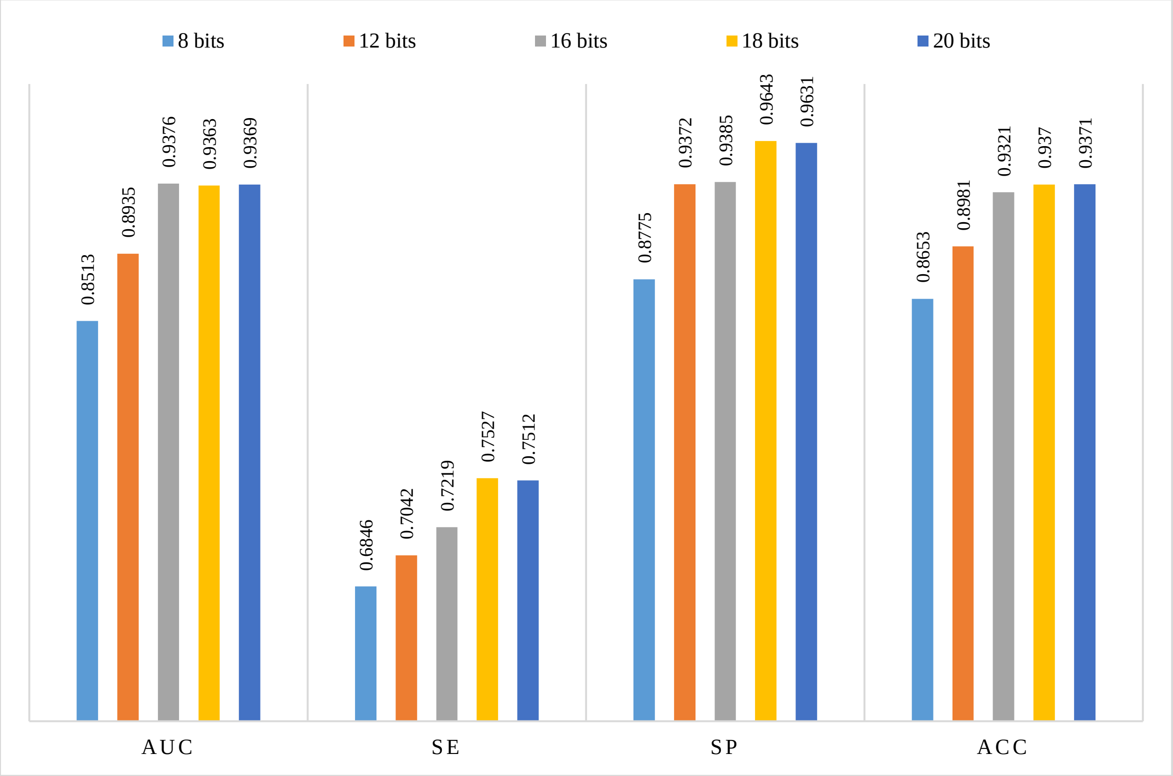


Figure 5-12. Quality measures with respect to precision of fixed point computation

For high-resolution images of 3504 × 2336 pixels, we use a window of size W = 41 pixels. The loop is thus executed 21 times and 2 images of 3504 × 2336 pixels are stored in memory. For the FPGA implementation, only 2 × 21 values are stored. For low-resolution images such as those of the DRIVE database, the parameter W of the MSLD algorithm is set to 15. Eight scales are considered (for L from 1 to 15 with a step of 2). For the CPU implementation, the loop is executed 8 times and two images of 565 × 584 pixels are stored in memory, while the FPGA implementation needs to store 2 × 8 values.

Table 5.2 shows the FPGA and CPU implementation performances for low-resolution images (565 × 584 pixels). The low-resolution images are only considered for comparison purposes. The execution time for the FPGA is 0.014 s with a throughput of 71.4 frames/s (f/s). The execution time for the CPU is 0.988 s with a throughput of 1.012 f/s. The FPGA implementation is 70.6× faster than the CPU implementation. The table also shows FPGA, GPU and CPU implementation performances for high-resolution images (3504 × 2336 pixels). For higher resolution images, the generated circuit consumes more resources than what is available in the Zynq XC7Z020-CLG484-1. For this reason, our circuit was regenerated for the Zynq 7Z045-FFG900-2 FPGA. The execution time of the FPGA is 0.447 s with a throughput of 2.2 f/s. The execution time for the GPU implementation is 50 s with a throughput of 0.02 f/s. The FPGA implementation is 111× faster than the GPU implementation. The execution time for the CPU is 144.6 s with a throughput of 0.006 f/s. The FPGA implementation is 323.5× faster than the CPU implementation. The FPGA implementation is faster than CPU and GPU implementations for both low and high-resolution images.

Table 5.2. FPGA versus CPU implementation performances

Image size	Platform	Time (s)	Throughput (f/s)	Speed Up
565×584 (W = 15)	**CPU**	0.988	1.012	1
	FPGA	0.014	71.428	**70.5 ×**
3504×2336 (W = 41)	**CPU**	144.611	0.006	1
	GPU	50	0.02	**2.9×**
	FPGA	0.447	2.232	**323.5 ×**

Table 5.3 shows the FPGA resources utilization for the MSLD architecture for DRIVE database images. The FPGA implementation uses 20% of the FPGA LUTs and 50% of its DSP Blocks. The maximum clock frequency is 60.4 MHz. This implementation achieves real-time execution of the MSLD algorithm with a throughput of 71 f/s.

Table 5.3. Logic utilisation for the MSLD algorithm for DRIVE database images

FPGA Resources	Used	Available	Utilisation
LUTs	10427	53200	20%
Flip-Flops	7498	106400	7%
DSP Blocs	110	220	50%
Maximum Frequency	60.443 MHz		

Using our designed compiler, we are able to generate an HDL description from a given set of the MSLD algorithm parameters. This represents an important gain in development time compared to manual coding. The same thing can be stated for the design verification efforts, since it requires global verification, while in manual coding, the verification should be done for each element of the design.

Figure 5-13 and Figure 5-14 show the FPGA resource consumption and frequency with respect to the MSLD window size (W). Figure 5-13 shows the FFs and LUTs necessary for the MSLD architecture for different values of W. Increasing the window size from 15 to 41 pixels implies an increase in the number of FFs by 5.8× and the LUTs by 4.8×. We can explain this by the fact that larger windows involve more pixels to process in parallel by the different functional units. Also, larger windows involve more scales (Ex: for W = 15, 8 scales are necessary while W = 41 necessitates 21 scales). Each scale requires several functional units. The number of DSP Blocks is also increased since the number of inherent computations is increased. In the same time and accordingly, the maximum clock frequency of our architecture is decreased from 60 MHz for W = 15 pixels to 40 MHz for W = 41 pixels.

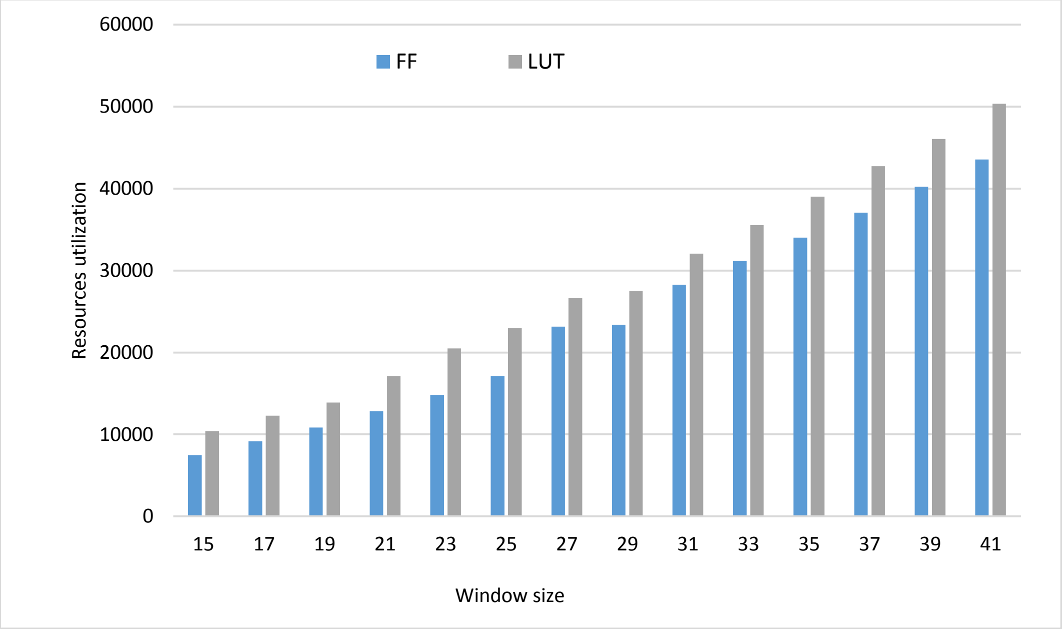


Figure 5-13. DSP blocks and circuit frequency as function of the window size

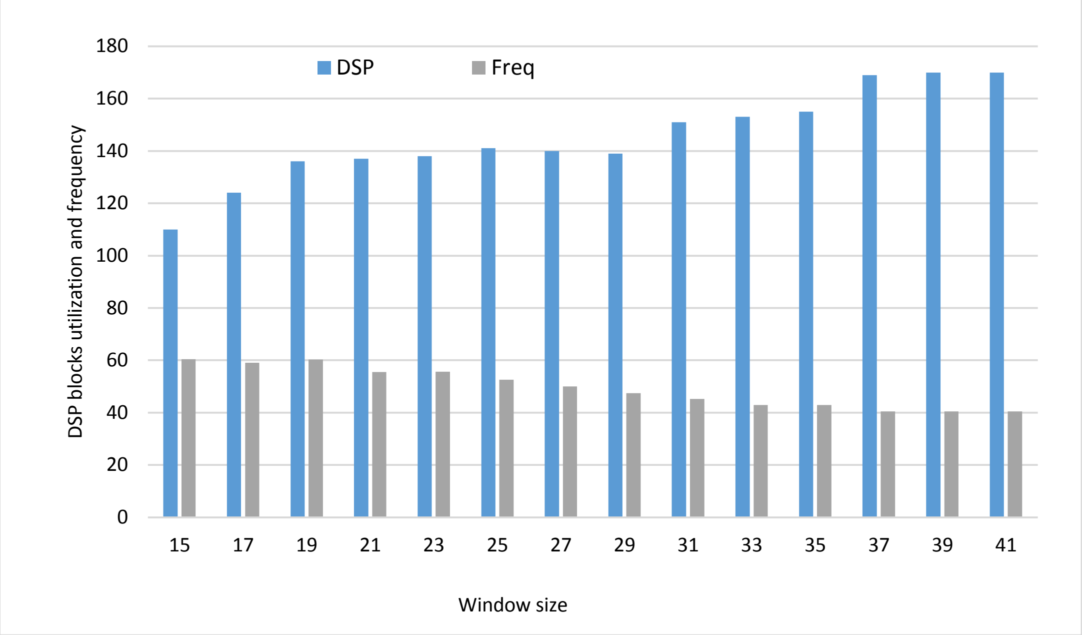


Figure 5-14. Resources utilization as function of the window size

5.5 Verification of the MSLD designs

Design and verification are complimentary tasks. As we did for the implemented matched filter in Chapter 4, we verified the correctness of the produced results by the MSLD architecture. A bit-accurate reference MATLAB model was developed, and each pixel of the output image was compared to the corresponding pixel produced by the model. The verification process proved the correctness of the MSLD architecture results.

5.6 Conclusion

In this chapter, a memory efficient custom architecture for the MSLD algorithm was proposed. It minimizes memory requirements, thanks to computations reuse and parallelism. The paper also presented a compiler able to generate low-level hardware descriptions of the architecture for a given set of parameters that include: image size, number of scales, number of line orientations and number of bits involved in the computations. The compiler generates pipelined functional units to increase the throughput. An FPGA implementation of the proposed architecture was realized on a Zynq platform for real-time retinal blood vessel segmentation from fundus images. The MSLD algorithm was also implemented on a CPU and on an Nvidia GPU. Acceleration factors of 70× and 323× of the FPGA implementation among software implementation are reported for low and high-resolution images, respectively, with comparable accuracy. In comparison with the GPU implementation for high-resolution images, the FPGA implementation is 111× faster.

CHAPTER 6 GENERAL DISCUSSION

This thesis introduced new hardware architectures for retinal image analysis. In this chapter, we discuss and compare the proposed implementations and highlight the limitations of our work.

Our first contribution is the proposal of a co-processor for the RLM features computations to analyze retinal image textures. Analyzing retinal image textures allows the description of the image and subsequently will help to classify the retinal images regarding their quality. Our aim was to accelerate the task of retinal image texture analysis by accelerating the RLM features computation. Our co-processor achieved good performances when compared to the original software implementation with an acceleration of 30.1×.

It is difficult to compare our work to other existing works in the literature, since existing works are all software implementations on CPUs. However, the proposed implementation of the RLM features computation for retinal quality assessment could be improved. The bottleneck of the RLM algorithm is the irregularity in memory access. FPGAs are able to offer a high level of parallelism, however, the limited clock frequency is a real issue. Therefore, only a serial execution (on a CPU) with high clock frequency can accelerate the access to the memory and improve the execution of the algorithm. Another limitation of our work is that the features computation task is not integrated with other tasks to realize a complete embedded system able to evaluate the quality of retinal images. A complete system would require the extension of our work to include a preprocessing step to allow the computation of more features with the integration of a classification algorithm.

Our other contributions concern the design and implementation of blood vessel segmentation architectures. In the following paragraphs, we compare our proposed implementations to existing hardware implementations described in the literature. The results are summarized in Table 6.1, which is organized by type of algorithm, segmentation accuracy, platform, image size, execution time, and throughput. The segmentation accuracy is computed for images of the DRIVE dataset. Throughput is generally considered to be the most important metric. However, an overall performance comparison is difficult to make and cannot be based on throughput alone. It must consider factors and parameters such as energy, ease of programmability, the nature and complexity of the algorithm and platform. Depending on the nature of the implementation, e.g. in a Data Center or an embedded system, some algorithms are more appropriate than others.

For our results for retinal blood vessel segmentation, and as expected, high resolution images require more time to execute the algorithm. However, we can observe that the relation between execution time and number of pixels is not always linear. It is strongly affected by the algorithm parameters and complexity. For example, in the case of the MSLD algorithm implemented on FPGA, when the image size is increased from 565 × 584 to 3504 × 2336, the number of pixels grows by 24×, but the execution time grows by 159×, from 0.014 s to 2.232 s.

Table 6.1. Summary of existing implementations of Retinal Blood Vessel Detection Algorithms

Ref	**Algorithm**		**Accuracy (DRIVE)**	**Platform**	**Image size (Pixels)**	**Time (s)**	**Throughput (pixel/sec) × 10^4**
[44]	Others methods	Features Extraction and Region Growing	0.925	Several computers	2890 × 2308	73.991	9.0
[65]		Active contour (PLS) + Morphology	0.9180	SIMD - FPGA	768 × 584	1.349	33.2
[58]		Local Radon Transform	0.9468	GPU	4288 × 2848	1.200	1017.7
[57]		Fusion + Registration	-	CPU	400 × 304	-	-
[53]		CNN	-	GPU	565 × 584	0.085	388.2
[60]	Filtering based methods	Matched Filtering	-	FPGA	640 × 480	0.005	6604.8
				GPU	640 × 480	0.023	1351.7
[66]		Matched filtering	0.9240	FPGA	768 × 584	0.052	857.6
[59]		Global Image Filtering + Contour Tracing	0.9431	GPU	4288 × 2848 565 × 584	0.753 0.014	1621.8 2356.9
This work		Matched Filtering	0.9218	Xtensa with custom TIEs	320 × 240	0.576	13.3
					768 × 584	3.340	13.4
					3504 × 2336	59.493	13.7
				FPGA	640 × 480	0.002	15360.0
					768 × 584	0.002	22425.6
					3504 × 2336	0.054	15158.0
					4288 × 2848	0.083	14714.0
[52]	Line operator based methods	Line op + SVM	0.9646	CPU	-	-	-
[54]		MSLO + K-means	0.9387	CPU	565 × 584	7.600	4.3
[55]		MSLO	-	CPU	768 × 576	120.000	0.4
[56]		MSLD + Tensor voting	0.9479	CPU	565 × 584	1099.000	< 0.1
[45]		MSLD	0.9407	CPU	565 × 584	2.500	13.2
This work		MSLD	0.9370	FPGA	3504 × 2336	2.232	366.7
			0.9407	GPU		50.000	16.4
			0.9407	CPU		144.611	5.7
			0.9370	FPGA	565 × 584	0.014	2356.9
			0.9407	CPU		0.988	33.4

For the matched filter algorithm, and as can be seen in Table 6.1, our proposed system implemented in Kintex-7 FPGA (XC7K480T-1FFG1156) outperforms all the existing matched filter implementations in terms of execution time. For a 640 × 480 image size, our system is 2.3× faster than the one proposed in [60] and implemented in Spartan-3. When synthesized for the same FPGA component, our system is 1.46× faster. For an image size of 768 × 584 pixels, the proposed system is 19× faster than the one proposed in [66] implemented in Spartan-6 FPGA. In [66], the proposed architecture includes a thresholding step in addition to the matched filter, which may affect the speed of the global architecture. When synthesized for the same FPGA component, our system is 14× faster. For high-resolution images of 4288 × 2848 pixels, our proposed system is 14× faster than the proposed GPU implementation in [58]. As mentioned, we obtained the highest performances in comparison with CPU, GPU and FPGA implementations. This is due to several hardware optimizations with parallelism and pipeline techniques utilization. Our FPGA implementation of the matched filter presents the lowest execution time an overcome all other implementations. However, the segmentation quality is limited to 92.18 % in terms of accuracy. Table 6.2 summarizes the speedups of our system when compared to the previous works, when using the same and different implementation platforms.

Table 6.2. Speedup of our matched filter FPGA implementation over similar implementations.

Ref	Platform	Image size	Speedup
[60]	Spartan-3 vs Spartan-3	640 × 480	1.46 ×
	Kintex-7 vs Spartan-3		2.3 ×
[66]	Spartan-6 vs Spartan-6	768 × 584	14 ×
	Kintex-7 vs Spartan-6		19 ×
[58]	Kintex-7 vs GPU	4288 × 2848	14 ×

For the architecture based on the ASIP, the performances are less than those of the hardware architectures targeting FPGAs. However, the ASIP implementation still better than several CPU implementations such as in [44] and [54] [56] in terms of execution time.

For the MSLD implementation, and among the line operator based methods, our FPGA and GPU implementations are the fastest in terms of execution time. The FPGA implementation achieves the highest performance in term of execution time. For the other algorithm categories, our implementation is better than many in terms of throughput and segmentation accuracy, such as the works in [44, 65, 66]. In comparison with the GPU implementation in [59], the execution time and throughput for images of DRIVE dataset are the same, while they outperform in terms of segmentation accuracy. However, the GPU implementations in [58] and [59] are performing better than our implementations in terms of execution time for high-resolution images, but we believe that our implementation has more potential if power consumption are considered since GPUs are much less power efficient [123, 124].

Taking into consideration the retinal blood vessel segmentation quality, the implementation in [52] presents the highest performances with 96.46% segmentation accuracy. However, the execution time is not reported. Complex algorithms such as in [56] present high segmentation accuracy, but they are implemented on CPUs and the execution time is very high. The work in [60] is not applied to retinal blood vessels and thus, we are not able to compare in terms of segmentation quality. By implementing the MSLD algorithm, we achieved better segmentation accuracy than our matched filter implementation with an improvement of 1.5%.

In conclusion, the ASIP implementation is flexible and allows the modification of the architecture parameters easily, but, in terms of performances, this implementation is not to be selected when execution time is an important factor and needs to be reduced. Among the cited FPGA implementations, our implemented matched filter outperforms all other implementations in terms of execution time. In addition, and to the best of our knowledge, our implementations are the only ones targeting low and high-resolution images. Our architectures are made scalable and flexible, thus, we are able to scale the architectures for any image size using our developed specific compiler. The limitations of our hardware architectures for retinal blood vessel segmentation are more concerning the segmentation accuracy. This aspect should be improved.

CHAPTER 7 CONCLUSION AND FUTURE WORK

In this thesis, we introduced several hardware architectures for retinal image processing. Firstly, we addressed the retinal image quality assessment. This is the first task in the pipeline. Texture analysis is a good solution to characterize retinal image quality and allow their classification. Run-length encoding features can help to analyze the texture of the images, however, the nature of the algorithm does not allow its parallelization. We found that the best way to accelerate the algorithm is to partition the algorithm to software and hardware parts. We get benefit from the Zynq new technology of Xilinx to implement the software part in the integrated Arm processor. The features computation was designed as a co-processor in the programmable logic of Zynq. This choice was made to allow the parallel computation of all RLM features at the same time. This solution achieved better results when compared to its all software version.

Secondly, we addressed the hardware implementation of retinal blood vessel segmentation. Countless algorithms have been proposed to solve this problem, mainly implemented on CPUs. Recently, more GPU implementations have been published and fewer hardware architectures. In this thesis, we propose several hardware architectures targeting mainly matched filtering and line operator techniques. We first designed a hardware architecture for the matched filter algorithm and implemented it on FPGA. Our optimized architecture achieved better results when compared to existing hardware architectures. The adaptation of the proposed architecture to different image sizes was challenging. The challenge is to keep the low level optimizations to get efficient implementation while allowing the adaptation of the architecture to different image sizes, especially high resolution images. To allow this flexibility, we designed a specific compiler able to generate the HDL description of the architecture from a set of algorithm parameters. This was a contribution since it was the first architecture able to address low and high resolution images. We have also proposed an ASIP-based architecture for the matched filter algorithm. The ASIP was developed based on an Xtensa processor with two additional specific instructions. The two added instruction allowed the acceleration of the matched filter execution, however, when compared to our hardware architecture implemented on FPGA, we found that the ASIP is not a favorable option when execution time need to be reduced, but, it is a nice option to consider especially when flexibility is a main factor.

For the targeted line operator technique for blood vessel segmentation, we considered the MSLD algorithm. This choice is mainly dictated by the performance of this technique in terms of blood vessel segmentation quality. This technique allows a significant improvement against other techniques in terms of segmentation accuracy, however, this technique is multi-scale which makes it computation and memory intensive. In this thesis, we proposed a memory efficient architecture able to solve the memory problem and allow the acceleration of the algorithm. This solution is based on software/hardware partitioning, thanks to Zynq technology of Xilinx. To allow the adaptation of the architecture to different image resolutions, we developed a specific compiler to generate HDL description of the algorithm from a set of its parameters.

The developed hardware architectures for matched filtering and line operators are compared in terms of resources utilization, maximum clock frequency and in terms of retinal blood vessel segmentation quality. The proposed MSLD architecture implemented on FPGA represents a good compromise between segmentation quality and execution time.

Even though the proposed architectures have presented multiple contributions in the field of hardware design for retinal imaging, several improvements could be proposed to the solutions presented and, moreover, many extensions could be provided.

To improve the proposed system for RLM features computing, we think that a better solution will require a platform with a CPU connected via a high speed bus such as PCI to the FPGA. The CPU will benefit from its high frequency to compute the run-length matrix and send it via PCI to the FPGA to allow the computation of the features in parallel. For an embedded system able to execute a full retinal image quality assessment task, we recommend to use a Zynq platform and adopt a co-designed solution based on SW/HW partitioning. We also recommend the use of new tools such as SDSoC from Xilinx or high-level synthesis tools such as Vivado-HLS.

Many authors have proposed improved matched-filter like techniques based on the modification of kernel coefficients for a better fit with vessel models such as in [44]. Based on our proposed architecture for the matched filter, and using our developed compiler, we can implement these new algorithms with fewer efforts to achieve better segmentation accuracy. One other improvement to the matched filter that can easily be mapped to the architecture is the use of multi-scales. It has already been proven that multi-scale matched filters perform better than matched filters in mono scale and allow the segmentation of vessels of different sizes [125]. Based on our architecture, the implementation of a multi scale matched filter is possible with fewer efforts.

The proposed architecture of the MSLD algorithm can be reused in different manners to improve the blood vessel segmentation quality. The line operator technique proposed first to generate features of vessels in mono scale. These features have been used to classify pixels as vessel or non-vessel pixels. A possible improvement is to use our architecture to generate blood vessel features at multiple scales to allow there classification as vessel or non-vessel pixels using a classifier such as in the original implementation of line operators by Ricci et al. [52]. The classifier has to be integrated and implemented on FPGA to improve the execution time. An interesting future work would be to address the algorithmic part of the MSLD algorithm. For instance, the combined response of the different scales is a linear combination of the different scales responses. A possible contribution is to include optimization and introduce a weighted combination. This contribution to the algorithm can be easily transferred to the hardware implementation based on the proposed architecture and the specific compiler.

In this thesis, we always preferred the use of low level programming models. To ensure an enough flexibility in case of changes to the algorithm parameters change, we developed specific compilers able to generate HDL descriptions automatically. An interesting future work would be to implement the algorithms using high-level synthesis tools such as Vivado-HLS of Xilinx and compare with the results of proposed low level programming.

Recently, deep-learning approaches have become very popular to solve computer vision problems. For retinal image processing, several authors proposed algorithms based on deep-learning for blood vessel segmentation and have achieved interesting results. However, for high-resolution images, the training and inferring are time consuming tasks. Deep-learning FPGA implementations are also an active research area. A nice contribution would be to propose hardware architectures and implementations of a deep-learning approach for retinal blood vessel segmentation on FPGA.

BIBLIOGRAPHY

[1] N. R. Brigitte Côté "Dépistage de la rétinopathie diabétique au Québec," AETMIS2008.

[2] B. J. Baudoin Ce Fau - Lay, J. C. Lay Bj Fau - Klein, and J. C. Klein, "Automatic detection of microaneurysms in diabetic fluorescein angiography," 19850308 DCOM- 19850308.

[3] M. D. Abramoff, M. Niemeijer, and S. R. Russell, "Automated detection of diabetic retinopathy: barriers to translation into clinical practice," *Expert Rev Med Devices,* vol. 7, pp. 287-96, Mar 2010.

[4] M. D. Abràmoff, J. M. Reinhardt, S. R. Russell, J. C. Folk, V. B. Mahajan, M. Niemeijer, *et al.*, "Automated Early Detection of Diabetic Retinopathy," *Ophthalmology,* vol. 117, pp. 1147-1154, 6// 2010.

[5] H. Bendaoudi, Q. Gan, F. Cheriet, H. B. Tahar, and J. M. P. Langlois, "A run-length encoding co-processor for retinal image texture analysis," in *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2015, pp. 1-6.

[6] H. Bendaoudi, F. Cheriet, H. Ben Tahar, and J. M. P. Langlois, "A Scalable Hardware Architecture for Retinal Blood Vessel Detection in High Resolution Fundus Images " presented at the Submitted to DASIP 2014. Conference on Design & Architectures for Signal & Image Processing, Madrid, Spain, 2014.

[7] H. Bendaoudi, F. Cheriet, A. Manraj, H. Ben Tahar, and J. M. P. Langlois, "Flexible architectures for retinal blood vessel segmentation in high-resolution fundus images," *Journal of Real-Time Image Processing,* pp. 1-12, 2016.

[8] H. Bendaoudi, F. Cheriet, and J. M. P. Langlois, "Memory efficient Multi-Scale Line Detector architecture for retinal blood vessel segmentation," in *2016 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, 2016, pp. 59-64.

[9] H. Bendaoudi, F. Cheriet, and J. M. P. Langlois, "Memory Efficient Flexible Architecture for Retinal Blood Vessel Segmentation Using a Multi-Scale Line Detector," *submitted to IEEE Transactions on Circuits and Systems for Video Technology,* 2017 2017.

[10] J. E. G. A. C. Hall, *Guyton and Hall textbook of medical physiology*. Philadelphia, Pa.: Saunders/Elsevier, 2011.

[11] L. Giancardo, "Automated fundus images analysis techniques to screen retinal diseases in diabetic patients," Université de Bourgogne, 2011.

[12] EyeDocs. (2017). retrived 30 April 2017, Available: http://www.eyedocsottawa.com/fr/services-et-troubles-visuels-speciaux/troubles-visuels-speciaux/diabete/

[13] S. Philip, L. M. Cowie, and J. A. Olson, "The impact of the Health Technology Board for Scotland’s grading model on referrals to ophthalmology services," *British Journal of Ophthalmology*, vol. 89, pp. 891-896, July 1, 2005 2005.

[14] P. H. Scanlon, R. Malhotra, R. H. Greenwood, S. J. Aldington, C. Foy, M. Flatman, *et al.*, "Comparison of two reference standards in validating two field mydriatic digital photography as a method of screening for diabetic retinopathy," *British Journal of Ophthalmology*, vol. 87, pp. 1258-1263, October 1, 2003 2003.

[15] A. D. Fleming, S. Philip, K. A. Goatman, J. A. Olson, and P. F. Sharp, "Automated assessment of diabetic retinal image quality based on clarity and field definition," *Investigative ophthalmology & visual science,* vol. 47, pp. 1120-1125, 2006.

[16] E. Trucco, A. Ruggeri, T. Karnowski, L. Giancardo, E. Chaum, J. P. Hubschman, *et al.*, "Validating retinal fundus image analysis algorithms: issues and a proposal," *Investigative ophthalmology & visual science,* vol. 54, pp. 3546-3559, 2013.

[17] L. Giancardo, F. Meriaudeau, T. P. Karnowski, E. Chaum, and K. Tobin, "Quality assessment of retinal fundus images using elliptical local vessel density," *New Developments in Biomedical Engineering., Campolo D., Ed,* 2010.

[18] S. C. Lee and Y. Wang, "Automatic retinal image quality assessment and enhancement," 1999, pp. 1581-1590.

[19] M. LalondeÝ, L. GagnonÝ, and M.-C. BoucherÞ, "Automatic visual quality assessment in optical fundus images," 2001.

[20] D. Usher, M. Himaga, M. Dumskyj, and J. Boyce, "Automated assessment of digital fundus image quality using detected vessel area," in *Proceedings of Medical Image Understanding and Analysis*, 2003, pp. 81-84.

[21] M. Niemeijer, M. D. Abramoff, and B. van Ginneken, "Image structure clustering for image quality verification of color retina images in diabetic retinopathy screening," *Medical image analysis,* vol. 10, pp. 888-898, 2006.

[22] J. Paulus, J. Meier, R. Bock, J. Hornegger, and G. Michelson, "Automated quality assessment of retinal fundus photos," *International journal of computer assisted radiology and surgery,* vol. 5, pp. 557-564, 2010.

[23] G. Yangف, L. Gagnonق, S. Wangف, and M. Boucher, "Algorithm for detecting micro-aneurysms in low-resolution color retinal images," 2001.

[24] A. Sopharak, B. Uyyanonvara, S. Barman, and T. H. Williamson, "Automatic detection of diabetic retinopathy exudates from non-dilated retinal images using mathematical morphology methods," *Computerized Medical Imaging and Graphics,* vol. 32, pp. 720-727, 2008.

[25] T. Spencer, J. A. Olson, K. C. McHardy, P. F. Sharp, and J. V. Forrester, "An image-processing strategy for the segmentation and quantification of microaneurysms in fluorescein angiograms of the ocular fundus," *Computers and biomedical research,* vol. 29, pp. 284-302, 1996.

[26] M. J. Cree, E. Gamble, and D. Cornforth, "Colour normalisation to reduce inter-patient and intra-patient variability in microaneurysm detection in colour retinal images," 2005.

[27] T. Walter and J.-C. Klein, "Automatic detection of microaneurysms in color fundus images of the human retina by means of the bounding box closing," in *Medical Data Analysis*, ed: Springer, 2002, pp. 210-220.

[28] K. Zuiderveld, "Contrast limited adaptive histogram equalization," in *Graphics gems IV*, 1994, pp. 474-485.

[29] H. Tsutsui, H. Nakamura, R. Hashimoto, H. Okuhata, and T. Onoye, "An fpga implementation of real-time retinex video image enhancement," in *World Automation Congress (WAC), 2010*, 2010, pp. 1-6.

[30] M. C. Hanumantharaju, M. Ravishankar, D. R. Rameshbabu, and S. Ramachandran, "A novel FPGA implementation of adaptive color image enhancement based on HSV color space," in *Electronics Computer Technology (ICECT), 2011 3rd International Conference on*, 2011, pp. 160-163.

[31] B.-H. Hwang, J.-H. Yun, and M.-R. Choi, "Implementation of image enhancement algorithm with less computational complexity for real-time processing," in *10th International Meeting on Information Display and International Display Manufacturing Conference and Asia Display 2010*, Seoul, Korea, Republic of, pp. 30-31.

[32] M. Z. Zhang, L. Tao, M.-J. Seow, and V. K. Asari, "Design of an efficient flexible architecture for color image enhancement," in *Advances in Computer Systems Architecture*, ed: Springer, 2006, pp. 323-336.

[33] M. Azizabadi and A. Behrad, "Design and VLSI implementation of new hardware architectures for image filtering," in *Machine Vision and Image Processing (MVIP), 2013 8th Iranian Conference on*, 2013, pp. 110-115.

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

[35] A. Reza, "Realization of the Contrast Limited Adaptive Histogram Equalization (CLAHE) for Real-Time Image Enhancement," *Journal of VLSI signal processing systems for signal, image and video technology,* vol. 38, pp. 35-44, 2004/08/01 2004.

[36] M. M. Fraz, P. Remagnino, A. Hoppe, B. Uyyanonvara, A. R. Rudnicka, C. G. Owen, *et al.*, "Blood vessel segmentation methodologies in retinal images – A survey," *Computer Methods and Programs in Biomedicine,* vol. 108, pp. 407-433, 10// 2012.

[37] Y. Yi, M. Adel, M. Guillaume, and S. Bourennane, "A probabilistic based method for tracking vessels in retinal images," in *Image Processing (ICIP), 2010 17th IEEE International Conference on*, 2010, pp. 4081-4084.

[38] X. You, Q. Peng, Y. Yuan, Y.-m. Cheung, and J. Lei, "Segmentation of retinal blood vessels using the radial projection and semi-supervised approach," *Pattern Recognition,* vol. 44, pp. 2314-2324, 10// 2011.

[39] D. Marín, A. Aquino, M. E. Gegúndez-Arias, and J. M. Bravo, "A new supervised method for blood vessel segmentation in retinal images by using gray-level and moment invariants-based features," *Medical Imaging, IEEE Transactions on,* vol. 30, pp. 146-158, 2011.

[40] S. Chaudhuri, S. Chatterjee, N. Katz, M. Nelson, and M. Goldbaum, "Detection of blood vessels in retinal images using two-dimensional matched filters," *Medical Imaging, IEEE Transactions on,* vol. 8, pp. 263-269, 1989.

[41] M. Al-Rawi, M. Qutaishat, and M. Arrar, "An improved matched filter for blood vessel detection of digital retinal images," *Computers in Biology and Medicine,* vol. 37, pp. 262-267, 2// 2007.

[42] B. Zhang, L. Zhang, L. Zhang, and F. Karray, "Retinal vessel extraction by matched filter with first-order derivative of Gaussian," *Comput. Biol. Med.,* vol. 40, pp. 438-445, 2010.

[43] O. Dalmau and T. Alarcon, "MFCA: Matched Filters with Cellular Automata for Retinal Vessel Detection," in *Advances in Artificial Intelligence*. vol. 7094, I. Batyrshin and G. Sidorov, Eds., ed: Springer Berlin Heidelberg, 2011, pp. 504-514.

[44] M. A. Palomera-Perez, M. E. Martinez-Perez, H. Benitez-Perez, and J. L. Ortega-Arjona, "Parallel Multiscale Feature Extraction and Region Growing: Application in Retinal Blood Vessel Detection," *Information Technology in Biomedicine, IEEE Transactions on,* vol. 14, pp. 500-506, 2010.

[45] U. T. Nguyen, A. Bhuiyan, L. A. Park, and K. Ramamohanarao, "An effective retinal blood vessel segmentation method using multi-scale line detection," *Pattern recognition,* vol. 46, pp. 703-715, 2013.

[46] F. Zana and J. C. Klein, "Segmentation of vessel-like patterns using mathematical morphology and curvature evaluation," *IEEE Trans Image Process,* vol. 10, pp. 1010-9, 2001.

[47] J. Xiaoyi and D. Mojon, "Adaptive local thresholding by verification-based multithreshold probing with application to vessel detection in retinal images," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 25, pp. 131-137, 2003.

[48] B. S. Y. Lam and Y. Hong, "A Novel Vessel Segmentation Algorithm for Pathological Retina Images Based on the Divergence of Vector Fields," *Medical Imaging, IEEE Transactions on,* vol. 27, pp. 237-246, 2008.

[49] B. S. Lam, Y. Gao, and A.-C. Liew, "General retinal vessel segmentation using regularization-based multiconcavity modeling," *Medical Imaging, IEEE Transactions on,* vol. 29, pp. 1369-1381, 2010.

[50] J. V. B. Soares, J. J. G. Leandro, R. M. Cesar, H. F. Jelinek, and M. J. Cree, "Retinal vessel segmentation using the 2-D Gabor wavelet and supervised classification," *IEEE Transactions on Medical Imaging,* vol. 25, pp. 1214-1222, 2006.

[51] J. Staal, M. D. Abramoff, M. Niemeijer, M. A. Viergever, and B. v. Ginneken, "Ridge-based vessel segmentation in color images of the retina," *IEEE Transactions on Medical Imaging,* vol. 23, pp. 501-509, 2004.

[52] E. Ricci and R. Perfetti, "Retinal Blood Vessel Segmentation Using Line Operators and Support Vector Classification," *IEEE Transactions on Medical Imaging*, vol. 26, pp. 1357-1365, 2007.

[53] K. K. Maninis, J. Pont-Tuset, P. Arbeláez, and L. V. Gool, "Deep Retinal Image Understanding," presented at the Medical Image Computing and Computer-Assisted Intervention (MICCAI), 2016.

[54] V. M. Saffarzadeh, A. Osareh, and B. Shadgar, "Vessel Segmentation in Retinal Images Using Multi-scale Line Operator and K-Means Clustering," *Journal of Medical Signals and Sensors*, vol. 4, pp. 122-129, 2014.

[55] D. J. J. Farnell, F. N. Hatfield, P. Knox, M. Reakes, S. Spencer, D. Parry, *et al.*, "Enhancement of blood vessels in digital fundus photographs via the application of multiscale line operators," *Journal of the Franklin Institute*, vol. 345, pp. 748-765, 10/15/ 2008.

[56] A. Christodoulidis, T. Hurtut, H. B. Tahar, and F. Cheriet, "A Multi-scale Tensor Voting Approach for Small Retinal Vessel Segmentation in High Resolution Fundus Images," *Computerized Medical Imaging and Graphics*.

[57] B. C. Becker and C. N. Riviere, "Real-time retinal vessel mapping and localization for intraocular surgery," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, 2013, pp. 5360-5365.

[58] M. Krause, R. M. Alles, B. Burgeth, and J. Weickert, "Fast retinal vessel analysis," *Journal of Real-Time Image Processing*, vol. 11, pp. 413-422, 2013.

[59] F. Argüello, D. L. Vilariño, D. B. Heras, and A. Nieto, "GPU-based segmentation of retinal blood vessels," *Journal of Real-Time Image Processing*, pp. 1-10, 2014// 2014.

[60] T. Savarimuthu, A. Kjær-Nielsen, and A. Sørensen, "Real-time medical video processing, enabled by hardware accelerated correlations," *Journal of Real-Time Image Processing*, vol. 6, pp. 187-197, 2011/09/01 2011.

[61] A. Hematian, S. Chuprat, A. Manaf, S. Yazdani, and N. Parsazadeh, "Real-Time FPGA-Based Human Iris Recognition Embedded System: Zero-Delay Human Iris Feature Extraction," in *The 9th International Conference on Computing and InformationTechnology (IC2IT2013)*. vol. 209, P. Meesad, H. Unger, and S. Boonkrong, Eds., ed: Springer Berlin Heidelberg, 2013, pp. 195-204.

[62] F. Xitian, W. Chenlu, C. Wei, Z. Xuegong, W. Shengye, and W. Lingli, "Implementation of high performance hardware architecture of OpenSURF algorithm on FPGA," in *Field-Programmable Technology (FPT), 2013 International Conference on*, 2013, pp. 152-159.

[63] H. Feng-Cheng, H. Shi-Yu, K. Ji-Wei, and C. Yung-Chang, "High-Performance SIFT Hardware Accelerator for Real-Time Image Feature Extraction," *Circuits and Systems for Video Technology, IEEE Transactions on,* vol. 22, pp. 340-351, 2012.

[64] W. Deng and Y. Zhu, "A memory-efficient hardware architecture for real-time feature detection of the SIFT algorithm (abstract only)," presented at the Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays, Monterey, California, USA, 2013.

[65] A. Nieto, V. M. Brea, and D. L. Vilarino, "FPGA-accelerated retinal vessel-tree extraction," in *2009 International Conference on Field Programmable Logic and Applications*, 2009, pp. 485-488.

[66] D. Koukounis, C. Ttofis, A. Papadopoulos, and T. Theocharides, "A high performance hardware architecture for portable, low-power retinal vessel segmentation," *Integration, the VLSI Journal,* 2013.

[67] C. Alonso-Montes, D. L. Vilariño, P. Dudek, and M. G. Penedo, "Fast retinal vessel tree extraction: A pixel parallel approach," *International Journal of Circuit Theory and Applications,* vol. 36, pp. 641-651, 2008.

[68] H. Jiang, H. Ardo, and V. Owall, "A Hardware Architecture for Real-Time Video Segmentation Utilizing Memory Reduction Techniques," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, pp. 226-236, 2009.

[69] A. Agrawal, C. Bhatnagar, and A. S. Jalal, "A survey on automated microaneurysm detection in Diabetic Retinopathy retinal images," in *Information Systems and Computer Networks (ISCON), 2013 International Conference on*, 2013, pp. 24-29.

[70] M. Niemeijer, B. Van Ginneken, J. Staal, M. S. A. Suttorp-Schulten, and M. D. Abramoff, "Automatic detection of red lesions in digital color fundus photographs," *Medical Imaging, IEEE Transactions on,* vol. 24, pp. 584-592, 2005.

[71] A. J. Frame, P. E. Undrill, M. J. Cree, J. A. Olson, K. C. McHardy, P. F. Sharp, *et al.*, "A comparison of computer based classification methods applied to the detection of

microaneurysms in ophthalmic fluorescein angiograms," *Computers in biology and medicine,* vol. 28, pp. 225-238, 1998.

[72] G. B. Kande, T. S. Savithri, and P. V. Subbaiah, "Automatic detection of microaneurysms and hemorrhages in digital fundus images," *Journal of digital imaging,* vol. 23, pp. 430-437, 2010.

[73] D. Matei and R. Matei, "Detection of diabetic symptoms in retina images using analog algorithms," in *Proceedings of World academy of Science, Engineering and Technology*, 2008, pp. 409-412.

[74] M. J. Cree, J. A. Olson, K. C. McHardy, J. V. Forrester, and P. F. Sharp, "Automated microaneurysm detection," in *Image Processing, 1996. Proceedings., International Conference on*, 1996, pp. 699-702.

[75] D. Usher, M. Dumskyj, M. Himaga, T. Williamson, S. Nussey, and J. Boyce, "Automated detection of diabetic retinopathy in digital retinal images: a tool for diabetic retinopathy screening," *Diabetic Medicine,* vol. 21, pp. 84-90, 2004.

[76] L. Streeter and M. J. Cree, "Microaneurysm detection in colour fundus images," *Image Vision Comput. New Zealand,* pp. 280-284, 2003.

[77] M. Niemeijer, B. Van Ginneken, M. J. Cree, A. Mizutani, G. Quellec, C. I. Sánchez*, et al.*, "Retinopathy online challenge: automatic detection of microaneurysms in digital color fundus photographs," *Medical Imaging, IEEE Transactions on,* vol. 29, pp. 185-195, 2010.

[78] A. Hoover, V. Kouznetsova, and M. Goldbaum, "Locating blood vessels in retinal images by piecewise threshold probing of a matched filter response," *Medical Imaging, IEEE Transactions on,* vol. 19, pp. 203-210, 2000.

[79] Z. Larabi, Y. Mathieu, and S. Mancini, "Efficient Data Access Management for FPGA-Based Image Processing SoCs," in *Rapid System Prototyping, 2009. RSP '09. IEEE/IFIP International Symposium on*, 2009, pp. 159-165.

[80] P. Deepa and C. Vasanthanayaki, "FPGA based efficient on-chip memory for image processing algorithms," *Microelectronics Journal,* vol. 43, pp. 916-928, 11// 2012.

[81] F. Khalvati and M. Aagaard, "Window memoization: an efficient hardware architecture for high-performance image processing," *Journal of Real-Time Image Processing,* vol. 5, pp. 195-212, 2010/09/01 2010.

[82] Y. Haiqian and M. Leeser, "Optimizing data intensive window-based image processing on reconfigurable hardware boards," in *Signal Processing Systems Design and Implementation, 2005. IEEE Workshop on*, 2005, pp. 491-496.

[83] L. Deng, C. Chakrabarti, N. Pitsianis, and X. Sun, "Automated optimization of look-up table implementation for function evaluation on FPGAs," 2009, pp. 744413-744413-9.

[84] T. Sasao, S. Nagayama, and J. T. Butler, "Numerical Function Generators Using LUT Cascades," *Computers, IEEE Transactions on*, vol. 56, pp. 826-838, 2007.

[85] Y. Zhang, L. Deng, P. Yedlapalli, S. P. Muralidhara, H. Zhao, M. Kandemir, *et al.*, "A special-purpose compiler for look-up table and code generation for function evaluation," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2010, pp. 1130-1135.

[86] D. U. Lee, A. A. Gaffar, O. Mencer, and W. Luk, "Optimizing hardware function evaluation," *Computers, IEEE Transactions on*, vol. 54, pp. 1520-1531, 2005.

[87] O. Mencer and W. Luk, "Parameterized High Throughput Function Evaluation for FPGAs," *J. VLSI Signal Process. Syst.*, vol. 36, pp. 17-25, 2004.

[88] C. Torres-Huitzil and M. Arias-Estrada, "Real-time image processing with a compact FPGA-based systolic architecture," *Real-Time Imaging*, vol. 10, pp. 177-187, 6// 2004.

[89] G. Saldana and M. Arias-Estrada, "FPGA-based customizable systolic architecture for image processing applications," in *Reconfigurable Computing and FPGAs, 2005. ReConFig 2005. International Conference on*, 2005, pp. 8 pp.-3.

[90] H. Ariyadoost, Y. S. Kavian, and K. Ansari-Asl, "Two dimensional systolic adaptive DLMS FIR filters for image processing on FPGA," in *Electrical Engineering (ICEE), 2012 20th Iranian Conference on*, 2012, pp. 243-248.

[91] M. Asri, L. Hsuan-Chun, T. Isshiki, L. Dongju, and H. Kunieda, "A reconfigurable ASIP-based approach for high performance image signal processing," in *Circuits and Systems (APCCAS), 2012 IEEE Asia Pacific Conference on*, 2012, pp. 611-614.

[92] H.-C. Liao, M. Asri, T. Isshiki, D. Li, and H. Kunieda, "Flexible and high performance ASIPs for pixel level image processing and two dimensional image processing," *Journal of Information Processing*, vol. 21, pp. 552-562, 2013.

[93] V. Brost, F. Yang, and C. Meunier, "Flexible VLIW processor based on FPGA for efficient embedded real-time image processing," *Journal of real-time image processing*, vol. 9, pp. 47-59, 2014.

[94] J. P. Wittenburg, M. Ohmacht, J. Kneip, W. Hinrichs, and P. Pirsch, "HiPAR-DSP: a parallel VLIW RISC processor for real time image processing applications," in *Algorithms and Architectures for Parallel Processing, 1997. ICAPP 97., 1997 3rd International Conference on*, 1997, pp. 155-162.

[95] S. Kyo, T. Koga, and S. Okazaki, "IMAP-CE: a 51.2 GOPS video rate image processor with 128 VLIW processing elements," in *Image Processing, 2001. Proceedings. 2001 International Conference on*, 2001, pp. 294-297 vol.3.

[96] R. Koenig, L. Bauer, T. Stripf, M. Shafique, W. Ahmed, J. Becker, *et al.*, "KAHRISMA: A Novel Hypermorphic Reconfigurable-Instruction-Set Multi-grained-Array Architecture," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010*, 2010, pp. 819-824.

[97] D. Stevens, V. Chouliaras, V. Azorin-Peris, Z. Jia, A. Echiadis, and H. Sijung, "BioThreads: A Novel VLIW-Based Chip Multiprocessor for Accelerating Biomedical Image Processing Applications," *Biomedical Circuits and Systems, IEEE Transactions on,* vol. 6, pp. 257-268, 2012.

[98] K. N. Plataniotis and A. N. Venetsanopoulos, *Color image processing and applications*: Springer, 2000.

[99] D. Yazhuo and D. Yong, "A Parameterized Architecture Model in High Level Synthesis for Image Processing Applications," in *Design Automation Conference, 2007. ASP-DAC '07. Asia and South Pacific*, 2007, pp. 523-528.

[100] C. Desmouliers, E. Oruklu, S. Aslan, J. Saniie, and F. M. Vallina, "Image and video processing platform for field programmable gate arrays using a high-level synthesis," *Computers & Digital Techniques, IET,* vol. 6, pp. 414-425, 2012.

[101] C. Andriamisaina, P. Coussy, E. Casseau, and C. Chavet, "High-Level Synthesis for Designing Multimode Architectures," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on,* vol. 29, pp. 1736-1749, 2010.

[102] A. Boubekeur and G. Saucier, "A high level synthesis tool for massively parallel image processing ASICs," in *CompEuro '91. Advanced Computer Technology, Reliable Systems*

and Applications. 5th Annual European Computer Conference. Proceedings., 1991, pp. 53-57.

[103] F. Hannig, M. Schmid, J. Teich, and H. Hornegger, "A deeply pipelined and parallel architecture for denoising medical images," in *Field-Programmable Technology (FPT), 2010 International Conference on*, 2010, pp. 485-490.

[104] A. Shatnawi, J. Ghanim, and M. O. Ahmad, "High level synthesis of integrated heterogeneous pipelined processing elements for DSP applications," *Computers & Electrical Engineering*, vol. 30, pp. 543-562, 11// 2004.

[105] K. Benkrid and D. Crookes, "From application descriptions to hardware in seconds: a logic-based approach to bridging the gap," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 12, pp. 420-436, 2004.

[106] H. Bartling, P. Wanger, and L. Martin, "Automated quality evaluation of digital fundus photographs," *Acta Ophthalmologica*, vol. 87, pp. 643-647, 2009.

[107] M. Fasih, J. P. Langlois, H. B. Tahar, and F. Cheriet, "Retinal image quality assessment using generic features," in *SPIE Medical Imaging*, 2014, pp. 90352Z-90352Z-7.

[108] M. A. Tahir, A. Bouridane, and F. Kurugollu, "An FPGA Based Coprocessor for GLCM and Haralick Texture Features and their Application in Prostate Cancer Classification," *Analog Integrated Circuits and Signal Processing*, vol. 43, pp. 205-215, 2005.

[109] A. R. Akoushideh and A. Shahbahrami, "Accelerating Texture Features Extraction Algorithms Using FPGA Architecture," in *2010 International Conference on Reconfigurable Computing and FPGAs*, 2010, pp. 232-237.

[110] D. Bouris, A. Nikitakis, and I. Papaefstathiou, "Fast and Efficient FPGA-Based Feature Detection Employing the SURF Algorithm," in *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, 2010, pp. 3-10.

[111] L. Yao, H. Feng, Y. Zhu, Z. Jiang, D. Zhao, and W. Feng, "An architecture of optimised SIFT feature detection for an FPGA implementation of an image matcher," in *2009 International Conference on Field-Programmable Technology*, 2009, pp. 30-37.

[112] S. H. Rezatofighi, A. Roodaki, A. Pourmorteza, and H. Soltanian-Zadeh, "Polar Run-Length Features in Segmentation of Retinal Blood Vessels," in *2009 International Conference on Digital Image Processing*, 2009, pp. 72-75.

[113] J. Lee, B. C. Y. Zee, and Q. Li, "Detection of Neovascularization Based on Fractal and Texture Analysis with Interaction Effects in Diabetic Retinopathy," *PLOS ONE*, vol. 8, p. e75699, 2013.

[114] K. Sayood, *Lossless compression handbook*: Academic Press, 2002.

[115] K. Appiah, A. Hunter, P. Dickinson, and J. Owens, "A run-length based connected component algorithm for FPGA implementation," in *2008 International Conference on Field-Programmable Technology*, 2008, pp. 177-184.

[116] J. Trein, A. T. Schwarzbacher, and B. Hoppe, "FPGA implementation of a single pass real-time blob analysis using run length encoding," in *MPC-Workshop, February*, 2008.

[117] M. M. Galloway, "Texture analysis using gray level run lengths," *Computer Graphics and Image Processing*, vol. 4, pp. 172-179, 6// 1975.

[118] A. Chu, C. M. Sehgal, and J. F. Greenleaf, "Use of gray value distribution of run lengths for texture analysis," *Pattern Recognition Letters*, vol. 11, pp. 415-419, 6// 1990.

[119] B. V. Dasarathy and E. B. Holder, "Image characterizations based on joint gray level—run length distributions," *Pattern Recognition Letters*, vol. 12, pp. 497-502, 8// 1991.

[120] H. Yin, H. Jia, H. Qi, X. Ji, X. Xie, and W. Gao, "A Hardware-Efficient Multi-Resolution Block Matching Algorithm and its VLSI Architecture for High Definition MPEG-Like Video Encoders," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, pp. 1242-1254, 2010.

[121] M. Tomasi, M. Vanegas, F. Barranco, J. Diaz, and E. Ros, "High-Performance Optical-Flow Architecture Based on a Multi-Scale, Multi-Orientation Phase-Based Model," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, pp. 1797-1807, 2010.

[122] T. B. Preuber and R. G. Spallek, "Ready PCIe data streaming solutions for FPGAs," in *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*, 2014, pp. 1-4.

[123] J. Fowers, G. Brown, P. Cooke, and G. Stitt, "A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications," presented at the Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays, Monterey, California, USA, 2012.

[124] H. Giefers, P. Staar, C. Bekas, and C. Hagleitner, "Analyzing the energy-efficiency of sparse matrix multiplication on heterogeneous systems: A comparative study of GPU, Xeon

Phi and FPGA," in *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2016, pp. 46-56.

[125] K. S. Sreejini and V. K. Govindan, "Improved multiscale matched filter for retina vessel segmentation using PSO algorithm," *Egyptian Informatics Journal*, vol. 16, pp. 253-260, 11// 2015.