Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Asbjørn Magnus Midtbø

Fast Correlation Computations on Large Image Data Sets

Master's thesis in Electronic Systems Design
Supervisor: Per Gunnar Kjeldsberg
Co-supervisor: Tormod Njølstad
June 2023

NTNU
Norwegian University of
Science and Technology

Asbjørn Magnus Midtbø

Fast Correlation Computations on Large Image Data Sets

Master's thesis in Electronic Systems Design
Supervisor: Per Gunnar Kjeldsberg
Co-supervisor: Tormod Njølstad
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

NTNU
Norwegian University of
Science and Technology

NTNU

Norwegian University of Science and Technology

Master thesis for the degree
MSc in Engineering, Electronics

Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Task Description

Supervisor: Per Gunnar Kjeldsberg

Co-Supervisor: Tormod Njølstad

Fast Correlation Computations on Large Image Data Sets

Fast correlation computations are usually needed in medical image processing, video analysis, tracking, radar systems, and in many other applications. This assignment will include a literature review and will explore the possibilities and implementation aspects related to performing fast correlation computations using a system-on-a-chip field-programmable gate array (SoC FPGA) on real-time camera images and large image data sets.

A development board, ZCU104 from Xilinx for a "multiprocessor system-on-a-chip" (MPSoC) Zynq Ultrascale+ FPGA is the target platform in this project. In such MPSoC, one or more parallel correlation computations can be executed very fast in the on-chip programmable logic (PL), while efficient memory handling can be controlled by the on-chip microcontroller system (PS) connected to 4GB DDR4 memory and optionally to a 2TB-4TB SATA solid state disk (SSD). By using Jupyter and the PYNQ framework, a testbench will be developed suitable for experimenting with different solutions for correlation computations and memory handling, including solutions of different sizes and with different degrees of parallelism.

Abstract

The aim of this thesis is to optimize the cross-correlation of a large number of speckle images and implement it in hardware. Cross-correlation may be used to find the displacement of images with overlapping content. Speckle images are obtained by shining a laser on a surface. The laser will scatter in different directions depending on the roughness of the surface. This may then be used to measure a location very accurately. However, even small changes to the observing location will make the speckle pattern appear differently. Consequently, there is a need for a substantial amount of images to represent an object or area. Cross-correlation is computationally heavy and therefore a substantial amount of time may be used if there is a large number of images.

Cross-correlation may be performed in the spatial domain or the frequency domain. In this thesis, the frequency domain is used to capitalize on the characteristics of speckle images in the frequency domain. The frequencies of a speckle image will be distributed in an even cone shape in the frequency spectrum. This characteristic was then used to create the suggested solution of partial cross-correlation. Partial cross-correlation performs cross-correlation on parts of the frequency spectrum. This allows for a significant increase in performance, and the method still gives an exact result.

Partial cross-correlation is tested with the high-level programming language Python. This allows for a fast implementation. The results show that Partial cross-correlation is viable for use on speckle images. The use of partial cross-correlation on normal images may be possible, however, without pixel accuracy. Partial cross-correlation shows a performance improvement over the normal method of between 300 and 500 times depending on the system used and when the image size is 512 by 512.

Partial cross-correlation is very resource friendly. The data that is required to store the image frequency spectrums is reduced by 256 when the image size is 512 by 512. The hardware needed to implement partial cross-correlation is also greatly reduced compared to the normal method and no resource optimization is necessary. The performance improvements of using partial cross-correlation in hardware rather than software are between 14 and 190, depending on the system used.

Samandrag

Målet med denne masteroppgåva er å optimalisere krysskorrelasjon for ei stor mengd bilete med flekkemønster(speckle patterns). Dette skal også implementerast i maskinvare. Krysskorrelasjon kan brukas til å finne forskyving av to bildete med overlapp. Flekkemønstre får ein av å lyse ein laser på ei overflate. Denne overflata vil ha ein ruleik slik at laserlyset vil sprette i ulike rettingar basert på ruleiken. Ein kan då observera eit flekkemønster. Desse flekkemønstera kan bli brukast å måle lokasjon på eit objekt eller ei overflate svært nøyaktig. Men viss ein skal representera eit objekt eller ei overflate med flekkemønster må ein ha mange bilete. Dette er naudsynt sidan laserlyset vil observerast ulikt sjølv med små flyttingar. Dette førar då til at krysskorrelasjonsutrekningane, som alt tek tid, vil ta enno lenger tid.

Krysskorrelasjon kan gjerast i det romlege planet eller i frekvensplanet. I denne masteroppgåva vil vi bruka frekvensplanet for å utnytte at flekkemønster vil fordele seg i ein jamn kjegleform i frekvensspekteret. Dette kjenneteiknet er då brukt til å utvikle delvis krysskorrelasjon. Delvis krysskorrelasjon utfører berre korrelasjonsberekningane på delar av frekvensspekteret. Dette gjer at ein får ei særdels rask utrenking. Sjølv om det er berre delar av frekvensspekteret som blir brukt, så er delvis krysskorrelasjon framleis nøyaktig.

Delvis krysskorrelasjon har vorte implementert og testa med høgnivåspåket Python. Python er eit programmeringsspråk som eignar seg godt for rask utvikling og implementering av algoritmar. Resultata av testinga viser at det er mogleg å bruka delvis krysskorrelasjon for å finne forskyving i flekkemøsterbilete. Resultata viser også at det er eit potensiale for bruk på andre bilete, men med lågare presisjon. Testing med ulike system viser ei kraftig forbetring i programvareyting smamalikna med den vanlege metoden på mellom 300 og 500 gonger for bilete som har ein storleik på 512 gonger 512.

Delvis krysskorrelasjon eignar seg godt for ein maskinvareimplementasjon. Resursbruken er kraftig redusert både i form av logikk og lagringsplass. Delvis krysskorrelasjon bruker 256 gonger mindre plass på å lagra frekvensspektera enn den vanlege metoden når bileta er 512 gonger 512. Trongen for maskinvareresursar er også kraftig redusert, og det er enklare å implementera delvis krysskorrelasjon. Ytelseforbetringane i maskinvare over programvare er mellom 13 og 190 gonger.

Preface

My journey with cross-correlation started in the fall of last year during my specialization project. It was in this project that the basic concept of partial cross-correlation was born. When I started the work on my thesis I realized that some choices were either wrong or inefficient. So during the work on my thesis, the implementation of partial cross-correlation has changed completely and only the basic concept has stayed the same. The specialization project has had some influence on how my thesis is structured, however, no part of the project, except some figures, is reused.

This last semester has been great and I will look back on today with fond memories. I would like to thank my supervisor *Per Gunnar Kjeldsberg*. He has been a great teacher and has provided valuable academic guidance. I would also like to thank my co-supervisor *Tormod Njølstad*. He proposed the topic of my thesis and has delivered technical guidance and always asked questions that made me reflect. I also want to thank Gudmund Slettemoen for sharing his deep insight into optical analysis with me. Lastly, I would like to thank my fellow students, friends, and family for the support they have given. You all have been great listeners to my ramblings.

I have really enjoyed working on my thesis. Conceptualizing an original solution that delivers the scale of improvement that partial cross-correlation does was not something I envisioned when I started. Partial cross-correlation is probably one of my greatest achievements so far in life. The method is not the most complex and has drawbacks, however, producing an original method that delivers great improvement is something I am very proud of.

Asbjørn Magnus Midtbø

Contents

List of Tables

List of Figures

List of Abbreviations

AXI Advanced eXtensible Interface

AXIS AXI-Stream

CC Cross-Correlation

CORDIC Coordinate Rotation Digital Computer

DC Direct Current

DFT Discrete Fourier Transform

DMA Direct Memory Access

DSP Digital Signal Processor

FFT Fast Fourier Transform

FIFO First In First Out

FPGA Field Programmable Gate Array

GPIO General Purpose Input Output

GPU Graphics Processing Unit

HW Hardware

IO Input Output

IP Intellectual Property Blocks

NCC Normalized Cross Correlation

PCC Partial Cross-Correlation

PL Programmable Logic

PS Processing System

PYNQ Python productivity for Zynq

RAM Random Access Memory

RTL Register Transfer Level

SNR Signal to Noise Ratio

SoC System on a Chip

SW Software

UVVM Universal VHDL Verification Methodology

Chapter 1

Introduction

Astronomers may want to track and analyze the trajectory of celestial bodies. This may be done by taking pictures over time and measuring the displacement of the celestial body. Measuring displacement of images, or object location within an image, may be done with Cross-Correlation (CC). CC is computationally heavy, and when performed on large data sets this will eventually present a significant problem. If an effective method could be utilized there would be an increase in both performance and energy efficiency. Therefore this thesis will present a solution for fast CC on a large number of images. The proposed solution is implemented on a Field Programmable Gate Array (FPGA) and preliminary results will be presented.

1.1 Motivation

In this project, we want to measure the displacement of speckle images. Speckle images are obtained by shining a laser upon a surface. This surface will have a certain roughness and this roughness will make the laser scatter in different directions giving an observable speckle pattern [1]. In Figure 1.1 a visual example of what we want to measure may be observed. There are two different speckle images that spatially overlap. We want to find the displacement vector visualized in the image to the right. This displacement vector may be found with the help of CC. Speckle images may represent a specific location on an object or the location of the camera in space.

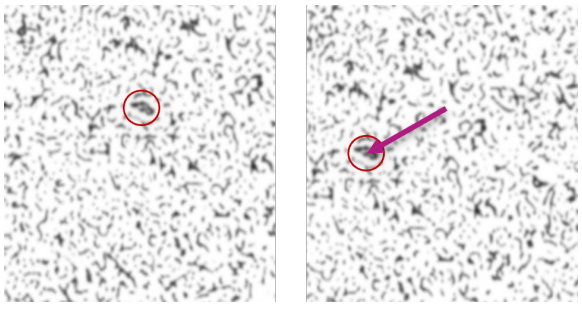


Figure 1.1: Visual example of displacement vector

CC is a common tool to analyze data. In the two-dimensional space, it may be used to measure displacement, location, deformation, and more. However, CC is mainly used in post-analysis because of the required computation. If CC is used in a real-time system, i.e. in a feedback loop, it will need expensive hardware and optimization.

There are several ways to optimize CC. Optimizing the calculations and selecting the appropriate hardware and programming language are viable solutions. However, it may be more desirable to implement CC with dedicated hardware. Using dedicated hardware will be faster if implemented correctly [2]. It depends if it is cost-effective to use dedicated hardware. However, if performance is of utmost importance, dedicated hardware is desirable. Therefore a CC implementation is to be realized in hardware for this thesis.

The Hubble space telescope uses mosaic imaging to represent larger areas [3]. The same mosaic representation may be applied to an area or object where details are important. This mosaic representation of areas or objects is used due to the physical limits of the resolution in cameras. If a higher level of detail is required, one will need to zoom in. This means that the number of images will increase for a given area. The Hubble legacy field covers roughly the same area of the sky as the moon and consists of almost 7500 images [4]. Thus if CC were to be used on these images it would require some time to process. The time needed to process is not ideal for use in real-time systems. Consequently, this thesis will explore an optimization of CC that may be used on a large number of speckle images.

1.2 Research Questions

This report will seek to answer these questions.

- Is it possible to optimize for a significant performance increase for CC on large data sets of images? What will the drawbacks of these optimizations be?
- How will a selected solution affect a hardware implementation and performance on an FPGA?
- Optimizing for a single use case often comes with limits for every other use case. How does the selected solution affect another use case?

1.3 Scope

CC may be performed either in the spatial domain or in the frequency domain. In this thesis, we will be performing the CC with the frequency domain. This limitation of scope is selected because there are multiple ways to perform CC and even more ways to perform possible optimizations. This makes it impractical to find a good solution in the given time frame, and therefore the scope is narrowed to the frequency domain.

The type of images we will be performing CC with are speckle images of equal sizes. Speckle images are images containing unique shapes referred to as speckles and may be used to represent an area. See details in Section 2.5. Based on the characteristics of these speckle images it is likely that appropriate simplification and optimization may be performed. Additionally, the speckle images will be monochromatic grayscale images. This will simplify the implementation. The difference in images to be correlated is translation, i.e. displacement. Rotation or scaling will not be present.

The FPGA that is the target for the implementation is the Xilinx Zynq Ultrascale+ MPSoC [5]. When hardware is to be uploaded, tested, and controlled on the FPGA, Python productivity for Zynq (PYNQ) will be used. This allows for rapid prototyping and testing. For details, see Section 2.7.

The hardware that is supposed to be implemented on the FPGA will consist of a mix of self-designed and Xilinx-provided Intellectual Property Blocks (IP). Xilinx IPs will be used to shorten the time for development and provide a stable implementation of complex features. The hardware description language that will be used for the rest of the implementation is VHDL. This is a personal preference. The hardware modules will be simulated and tested in Modelsim.

The focus of this project is to optimize CC for a large number of images. This means that more effort will be put into this than the hardware implementation. Consequently, a complete system for cross-correlating large data sets of images will not be implemented. The hardware implementation will consist of the primary parts of the solution, however, the timeframe allocated will not allow for the implementation and testing of a complete system.

1.4 Main Contributions

In this thesis, the following has been achieved.

- The development and implementation of partial cross-correlation. An optimized CC method for speckle images.
- Partial cross-correlation delivers significant improvements over normal CC when a large number of images is to be correlated. Both in terms of performance, energy efficiency, and resource usage.
- Both software and hardware implementations of partial cross-correlation have been realized.
- Due to the fact that partial cross-correlation needs only parts of the frequency spectrum a novel two-dimensional Fast Fourier Transform (FFT) has been developed. The novel two-dimensional FFT optimizes the transformation to the frequency domain.

1.5 Structure

This thesis will consist of the following chapters.

Chapter 2 contains the theory and required background information needed to understand the suggested solution. This will include both solution-specific theory and hardware-related info.

Chapter 3 introduces partial cross-correlation as a solution to our problem. The reasoning behind and how to implement partial cross-correlation will be shown.

Chapter 4 presents the results of a Python implementation of partial cross-correlation. The difference between the normal and partial cross-correlation will be shown as well as general performance and accuracy.

Chapter 5 outlines the necessary hardware needed for an FPGA implementation.

Chapter 6 presents the results of the hardware implementation of partial cross-correlation.

Chapter 7 discusses the partial cross-correlation and its hardware implementation. This includes weaknesses and possible avenues of new research.

Chapter 8 concludes the thesis.

Chapter 2

Background

To be able to measure the displacement of images some background information is needed. The basic theory needed to understand CC will be introduced. This chapter will define the speckle images to be used in this thesis. Basic Fourier transforms between points and interference patterns will be shown. There will also be presented hardware-specific background information.

2.1 Digital Image Processing

The earliest application of digital image processing was used to encode images for transfers in submarine cables between London and New York [6, p.25]. Since then, the field has grown massively to encompass everything one may do with an image. From hospital equipment to image restoration and enhancement, and more. Everything one may do to an image in order to achieve a specific goal.

In this thesis, we are performing digital image processing with the goal of being able to measure the displacement of images. This type of digital image processing is often referred to as template matching [6, p.891]. Template matching uses CC to search through an image for the best match of the given template. In Figure 2.1 an example of template matching may be seen where the template is the image of the letter S while the image is a collection of words. To the right in the figure, one may see the result of the calculation in the form of the two dots that correspond to the locations of the letter S.

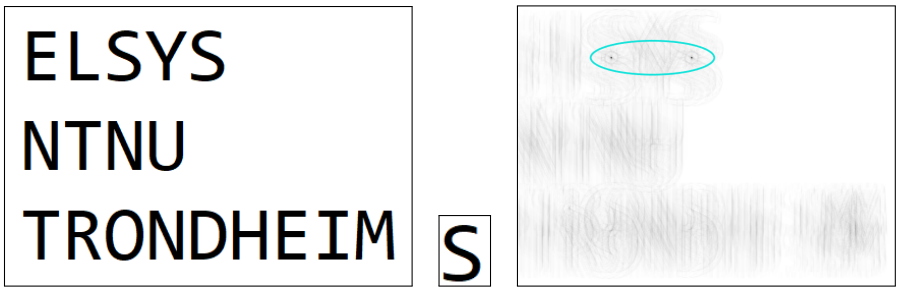


Figure 2.1: Template matching example. The result has inverted colors.

This thesis uses images of equal sizes, and no specific object is to be identified. Therefore it is referred to purely as CC instead of template matching. Regardless, both use CC to compute a result.

2.2 Cross-Correlation

In statistical analysis, correlation is used to evaluate if there is a linear relationship between two random series variables [7]. Cross-Correlation (CC) is very much the same, however, with the added benefit of looking through a series, e.g. a signal, for the best match [8, p.116]. It is common that this series is represented in either time or space. CC is a mathematical tool with many uses. It is for example used in radar, digital communication, sonar, geology, and of course, digital image processing [8, p.116].

A basic CC example may be seen in Figure 2.2. This is not an accurate representation, however, it conveys the same principle. The figure shows a situation where we have the signal $f(t)$ and want to know if and when another signal $g(t)$ is present. The calculated result of this CC is the $CC(t)$ signal. As seen, when $f(t)$ and $g(t)$ are completely overlapping we will have the maximum correlation strength. This is a one-dimensional signal, however, CC may be used with several dimensions, i.e. with images. The result in Figure 2.1 is essentially the same as the $CC(t)$ signal in two dimensions.

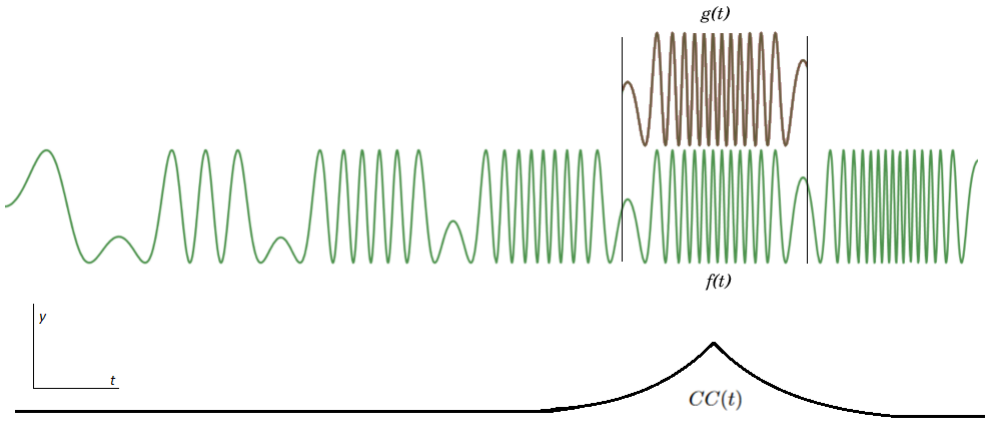


Figure 2.2: CC of the two signals f(t) and g(t).

CC is not perfect, and as seen in the results in Figure 2.1 there are other areas of high intensities other than the two correct dots. These ambiguous results are due to the image used and that there are no precautionary methods applied in this example. This ambiguity is a common problem with CC and is very dependent on the image used[9]. In the example in Figure 2.1 the ambiguity arises because of the characteristics of the font used by letters. The letters have the same basic size and thickness of lines. Which makes it poor for CC where letters have the same shapes. In the example, one may observe that the results are less ambiguous where the letters are purely horizontal and vertical. This is due to that the

letter S has very few vertical lines and even fewer horizontal. Ambiguity may also arise when there is a change in background intensity. However, this may not be observed in the example due to no change in background intensity[10].

This ambiguity is not a problem if the result is reviewed by a human, however, this ambiguity is a major problem for computers. This is because the nature of the ambiguity changes with the type of image used [9]. Therefore there is often the need for normalization [6, p.891].

2.3 Cross-Correlation in the Frequency Domain

There are two common ways of performing CC. One may do it in the spatial domain or in the frequency domain. For images, the spatial approach calculates the correlation for each pixel while the frequency approach calculates the correlation of the entire image [11, p.9]. The convolution theorem states that correlation in the spatial domain is equal to elementwise multiplication in the frequency domain[6, p.272].

CC with the help of the frequency domain may be seen in Equation 2.1. The images f and g are first transformed to the frequency domain. This is done with the help of a Discrete Fourier Transform (DFT) like the Fast Fourier Transform (FFT) and is shown with $\mathcal{F}$. More information about the Fourier transformation of images will be presented in Section 2.4. $\mathcal{F}^*$ denotes that it is the complex conjugate of the transform. One may not need to use the complex conjugate if the image is mirrored before the Fourier transform [12]. After the Fourier transforms, the elementwise matrix multiplication (Hadamard product), shown by $\circ$, is performed. Lastly, the inverse Fourier transform is computed. This is because the results need to be in the spatial domain [6, p.272]. The location of the offset will be the argument of maxima in the spatial domain($\Delta x, \Delta y = argmax(|CC|)$).

$$CC_{x,y} = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \circ \mathcal{F}^*\{g\}\} \tag{2.1}$$

In Figure 2.3 a flowchart of the CC method in the frequency domain is shown. In this example, we are using the same signals as in Figure 2.2. As may be observed the first step is to perform a Fourier transformation on both signals. The next step is to perform the elementwise multiplication. Last we need to do an inverse Fourier transform to be able to interpret the result.

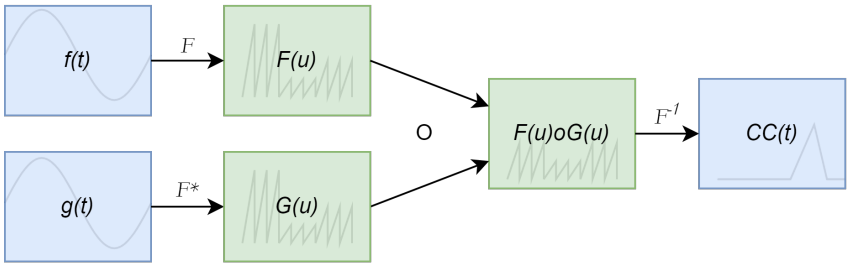


Figure 2.3: Flowchart of the CC of the two signals f(t) and g(t) in the frequency domain.

When performing CC in the frequency domain it is not the correlation multiplication that is compute-intensive. It is the Fourier transforms that require the most computing. The number of computations required for a single two-dimensional Fourier transform is $2N^2Log_2(N)$ [13]. Where N is the height and width of the image. This means that a full CC needs $6N^2Log_2(N) + N^2$ computations. This is because of the need for three transforms and one elementwise multiplication. This is a substantial amount of computing that is needed.

2.3.1 Normalization

The method in Equation 2.1 does not have any form for normalization. Therefore, depending on the image, there may be a need to add normalization. When performing CC in the frequency domain there are two common ways to normalize the result. Normalized CC and Phase Correlation[10]. The two methods are one of the most common types of normalization. There are other methods as well [14], however, they will not be presented.

Normalized Cross-Correlation

Normalized Cross Correlation (NCC) is a method that is used to negate the difference in background intensity between images and normalize the output[10]. This is done by subtracting the cumulative sum of one image multiplied by the mean from the other image and then dividing it by the standard deviation[15]. The NCC method is shown in Equation 2.2, where $\bar{g}$ denotes the mean of the image.

$$NCC_{x,y} = \frac{\mathcal{F}^{-1}\{\mathcal{F}\{f\} \circ \mathcal{F}^*\{g\}\} - \bar{g}\sum_{u,v} f(u,v)}{\sqrt{\sum_{u,v}(f(u,v) - \bar{f}_{x,y})^2 \sum_{u,v}(g(x+u, y+v) - \bar{g})^2}} \tag{2.2}$$

The NCC method solves the ambiguity problem we encountered in the example in Figure 2.1. NCC performs poorly when rotation or scaling is present. Nevertheless, this does not affect the viability if NCC is to be used in this thesis since rotation and scaling are outside the limitation of scope [10]. There is a substantial increase in computation needed to perform NCC. However, in [10] J.P. Lewis presents Fast NCC. Fast NCC utilizes running sums in the calculation in the denominator in Equation 2.2 to reduce the number of calculations at the cost of using more memory. If the calculation of the denominator is performed in parallel to the CC, it will be completed before the CC calculation.

Phase Correlation

When calculating the CC in the frequency domain one may perform phase correlation. Phase correlation is shown in Equation 2.3 and is a relatively simple operation. Before performing the inverse Fourier transform one may divide the values of the frequency spectrum by their absolute value. This will make the CC based on only phase and not weak for background intensity and with the added benefit of normalizing the output[16].

$$PC_{x,y} = \mathcal{F}^{-1}\left\{ \frac{\mathcal{F}\{f\} \circ \mathcal{F}^*\{g\}}{\|\mathcal{F}\{f\} \circ \mathcal{F}^*\{g\}\|} \right\} \tag{2.3}$$

Phase correlation is like NCC weak for rotation and scaling [17]. One might overcome this with the Fourier Mellin transform[17]. However, like NCC these problems are outside the scope of the thesis. Unlike NCC, phase correlation has the added drawback of using only phase information. This will make every component in the image weighted equally. This might make the method more receptive to noise. In order to make the method more robust against noise, pre-filtering should be applied to the image. J.P. Lewis suggests that the best pre-filter is approximately Laplacian [10].

2.4 The Fourier Transform of Images

In this section, the basic theory about the Fourier transform of images will be presented. There will also be shown some of the properties images have in the frequency spectrum. The contents of Subsection 2.4.4 are especially important for the reasoning of the implementation in the next chapter.

2.4.1 The Two-Dimensional Fourier Transform

When an image is to be represented in the frequency spectrum one will first need to transform the image with a Fourier transform. This may be done with the DFT method, however, this is very slow and the FFT method is almost always used. FFTs greatly decreases the computation needed for the Fourier transform [18]. To perform a Fourier transform of an image, a series of regular one-dimensional Fourier transforms are needed [6, p.321]. The two-dimensional transformation is performed by taking a Fourier transform of every row of the image and then using that result, taking another Fourier transform of every column [6, p.321]. If it is row first or column first does not matter and it will provide the same result. This operation is visualized in Figure 2.4 and the mathematical definition is shown in Equation 2.4. Note that this is the DFT definition of the two-dimensional Fourier transform.

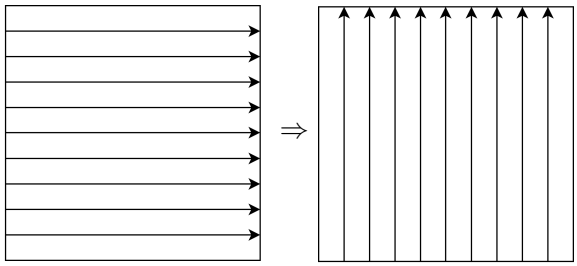


Figure 2.4: Fourier transform of an image. A series of Fourier transforms.

$$F_{x,y} = \sum_{c=0}^{C-1} \sum_{r=0}^{R-1} f(c,r) e^{-j2\pi(x\frac{c}{C} + y\frac{r}{R})} \tag{2.4}$$

2.4.2 The Frequency Spectrum and the FFT Shift

After the Fourier transform has been performed we may be able to observe how an image is represented in the frequency spectrum. Often an FFT shift is performed when doing so [6, p.260]. This is because the frequency spectrum is not in the best human observable form because the lower frequencies are located in the corners of the spectrum. Figure 2.5 shows the frequency distribution before and after an FFT shift. The FFT shift moves the zero-frequency, often referred to as the Direct Current (DC) frequency, to the center of the frequency spectrum. It does not come in this form since negative frequencies do not exist and by doing an FFT shift we are assigning parts of the spectrum negative frequencies. However, by performing an FFT shift we may be able to group the low frequencies together. In turn, this makes the frequency spectrum more readable. Lower frequencies represent the average intensity of, for example, an object where there is little intensity change. The higher frequencies represent, for example, the edges of an object, where there is a large intensity change [6, p.278].

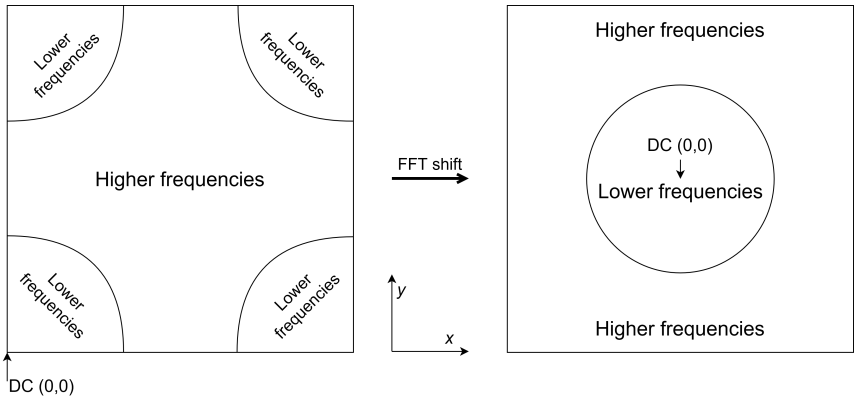


Figure 2.5: The frequency spectrum of an image before and after FFT shifting.

2.4.3 Coarse and Fine Details in the Frequency Spectrum

When analyzing and utilizing the frequency spectrum, one must take precautions. The frequency spectrum of images is often dominated by the lower frequencies [6, p.278]. This makes it relatively safe, i.e. no faulty calculations, to perform operations with the lower frequencies of the spectrum due to a higher Signal to Noise Ratio (SNR). Operations involving higher frequencies are unsafe in comparison. Higher frequencies will to a higher degree be affected by noise. Figure 2.6 shows an example of a cross-section of a frequency spectrum of an image. It may be observed that the signal will cross the noise floor when the frequencies

are higher. When selecting the level of detail, one must evaluate this curve. If finer details are required the SNR will be lowered.

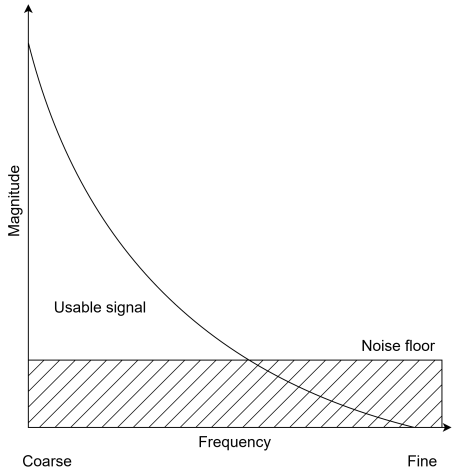


Figure 2.6: Cross-section of a frequency spectrum with an exaggerated noise floor.

2.4.4 Points and Interference Patterns

The result of CC comes in the form of a point in an image. The offset is indicated by the location of this point. Due to this, we may assume there is an interference pattern in the frequency spectrum. This is because the relationship between points and interference patterns is one of the basic phenomena in the introduction of Fourier transformation of images[19]. Figure 2.7 shows a point, with its weaker subpoints, and an interference pattern. An FFT shift has been performed. If a Fourier transform is performed on either one of them, the other will be the result of that transformation [20]. The distance between the point and subpoints dictates the frequency of the interference pattern. Or wise versa. Less distance between the points gives a lower frequency, i.e. further between wave tops [11, p.8].

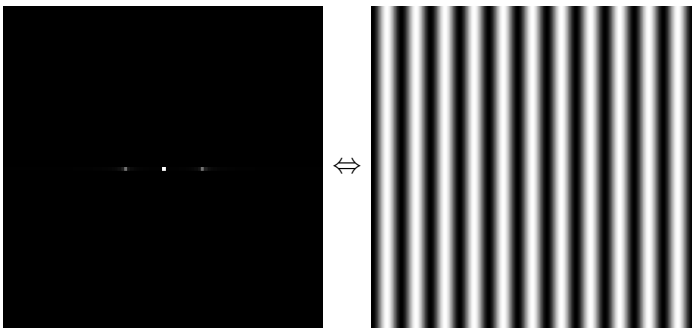


Figure 2.7: Fourier transforms of a point and an interference pattern

If the point in Figure 2.7 is a result of a CC, it would indicate a correlation with no offset, i.e. the images that are cross-correlated are completely overlapping. However, when performing CC it is not likely. Therefore the point has to be able to be moved around. This may

also be represented with interference patterns as seen in Figure 2.8. In the figure, one may observe that the offset may indeed be represented as an interference pattern with the Fourier transform. However, this figure is misleading. This is because the interference pattern is represented using its absolute values as the frequency spectrum uses complex numbers when the point is moved. If a Fourier transform were to be performed on this image of the interference pattern, the point would appear at the center. The information needed for the offset location is lost when applying the absolute value to the frequency spectrum.

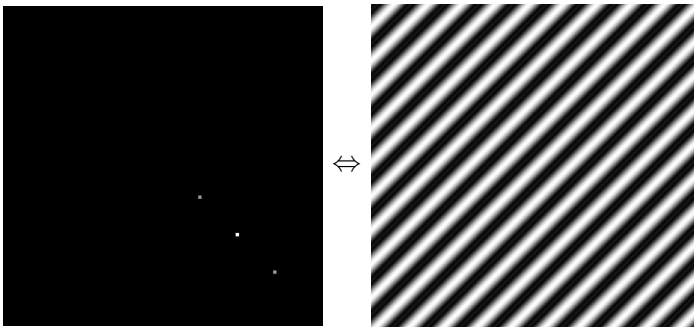


Figure 2.8: Fourier transform of a moved point and a changed interference pattern

Figure 2.9 shows the decomposed imaginary and real values of the changed interference pattern. We are still able to observe the interference pattern, however, other interference patterns have been introduced. More interference patterns will also be introduced when performing an actual CC. This will make the primary interference pattern in the frequency spectrum very hard to spot. The frequency spectrum may look very much like noise. However, the correct interference pattern is still present and provides the correct information needed for a point.

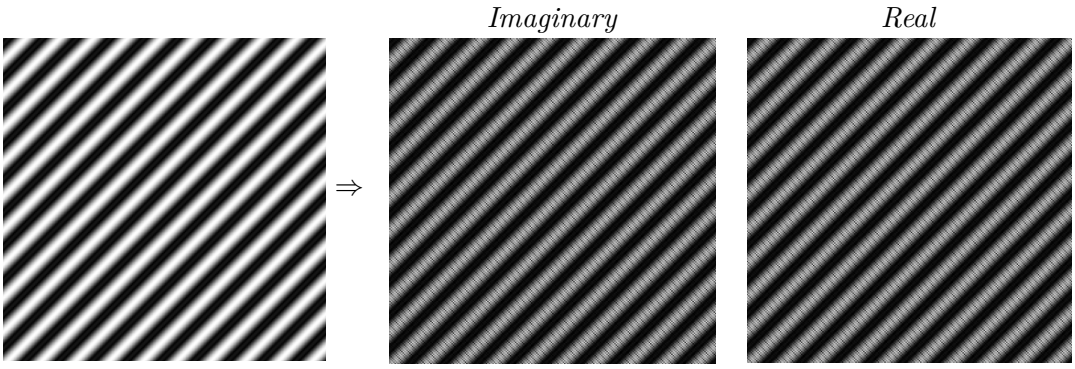


Figure 2.9: A changed interference pattern's imaginary and real components

2.4.5 Cross-Correlation Area

When performing a CC in the frequency domain it is customary to pad the images with their own length in each dimension. This is due to the circular behavior of the Fourier transform and is described in the convolution theorem [21]. Without this padding, one may encounter a

wraparound error. As explained in the Subsection 2.4.4, a perfect correlation will give a result in the form of a point at the center when FFT shifted. This means that this point is only able to move by half the image length in each dimension. If this is surpassed, a wraparound error like the one in Figure 2.10 will occur. As may be observed, the images that are cross-correlated are not sharing enough overlapping area. The image to be cross-correlated is offset too far to the southeast. The result will occur as a point, however, it will be ambiguous. In Figure 2.10 the calculated ambiguous offset is the red dot while the green dot represents the actual offset. This ambiguity is highly unwanted. The regular solution to avoid wraparound is to pad the images with their own length. The padding often consists of zeros, however, a duplicate or a mean of the image may be chosen. However, by applying padding, there will be an increase in computing by four times.

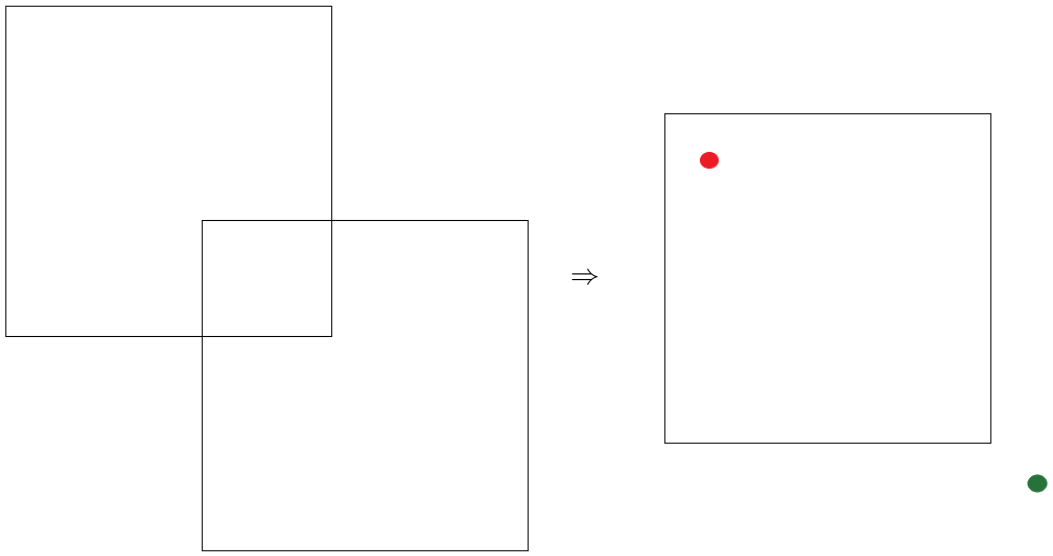


Figure 2.10: Wraparound error.

2.5 Speckle Images

If we want to measure accurately, fine details have to be present. To observe the fine details of an object we may use a laser. We may use this to observe the roughness of the object's material. The laser beam will scatter upon the surface and we will observe a speckle pattern[22]. Since the roughness of an object is relatively constant, we may use these speckles to make an accurate reference system. Hence the speckle pattern may be used to measure very accurately.

In Figure 2.11 a synthetic and captured speckle pattern may be seen. The difference in the patterns is due to the calculated contrast when creating the synthetic speckle pattern. It is possible to adjust the speckle pattern manually. However, this has to be done each time either the image or speckle sizes change. The difference is addressed in Chapter 4

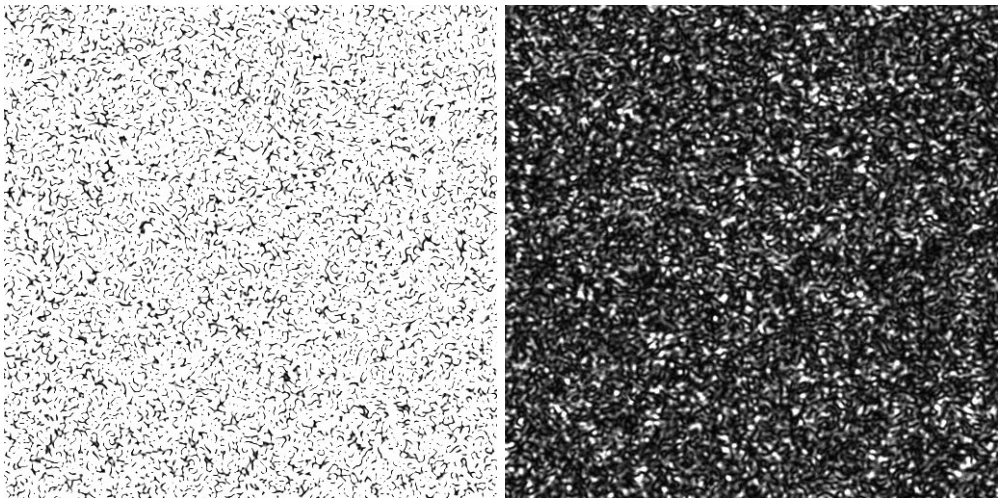


Figure 2.11: Synthetic(L) and captured(R) speckle pattern courtesy of [23].

The synthetic speckle pattern was generated using the method outlined in [24] and with the Matlab script(Appendix B.2) courtesy of Gudmund Slettemoen [23]. These synthetic speckle patterns are useful for testing since a large number of images may be generated with relatively little effort. The speckle images are also very distinct [24]. However, with this method, one must be aware that there is a symmetry in the generated images. This means that only a quarter of the image may be used. If the symmetry is not accounted for, the correlation strength will be equal for mirrored offsets. However, other than this it does not affect the results.

In this thesis, we will perform the CC in the frequency domain. So it has to be noted that the frequencies of speckle images are evenly divided in a cone shape in the frequency spectrum [23]. This may be seen in Figure 2.12 where there are frequency spectrums of two speckle images. The size of the speckles decides the width of the cone in the frequency spectrum. Smaller speckles give a wide cone while larger speckles give a narrow cone.

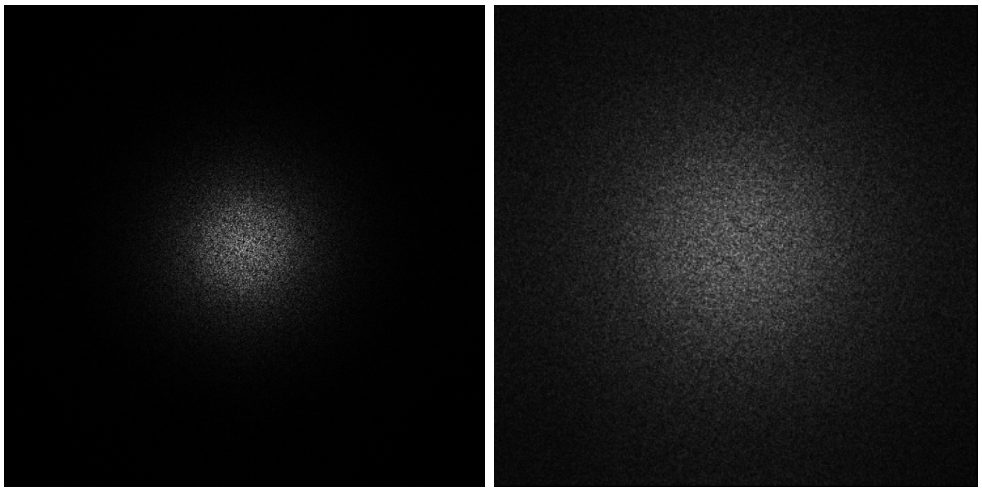


Figure 2.12: The frequency spectrum of speckle images with large(L) and small(R) speckles.

2.6 Zynq Ultrascale+ MPSoC ZCU104

When designing an implementation to be run on specific hardware one must always design with that in mind. The Zynq Ultrascale+ MPSoC ZCU104 is an evaluation board geared towards embedded vision applications [25]. Processing images and video is quite demanding, meaning that the FPGA is well equipped and has many features [5]. In Figure 2.13 a simplified block diagram of the Zynq Ultrascale+ is shown.

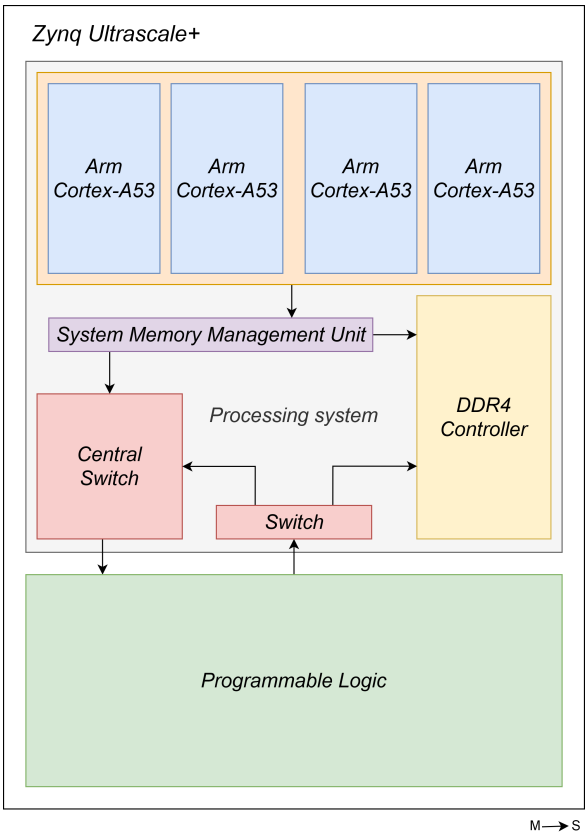


Figure 2.13: Simplified block diagram of Zynq Ultrascale+ MPSoC ZCU104

The FPGA has a rich set of other Processing System (PS) features as well. There are many IO options and, for example, there is a GPU included. However, we are designing hardware on the Register Transfer Level (RTL) in this project. Therefore we will be focusing on the Programmable Logic (PL) hardware. Table 2.1 shows an overview of the most important specifications needed for development. Advanced eXtensible Interface (AXI) is an on-chip memory interface developed and used by Arm [26]. Arm licenses processor IP and provides the processor architecture for the Zynq Ultrascale+ MPSoC ZCU104. Distributed RAM uses logic cells to create RAM blocks while block RAM and UltraRAM use dedicated hardware. UltraRAM is like block RAM, however, has more capacity at the cost of less versatility.

Feature	Resource Count
System Logic Cells	504k
CLB Flip-Flops	461k
Distributed RAM	6.2 MB
Block RAM	11 MB
UltraRAM^{TM}	27 MB
DSP Slices	1728
SODIMM to PL	1 socket
High-performance AXI Masters	4x128 bits

Table 2.1: Zynq Ultrascale+ features and resource count [5, p.9].

2.6.1 Hardware Design Considerations

For Image processing and the use of Fourier transforms the two most important figures in Table 2.1 is the amount of memory and the number of Digital Signal Processor (DSP) slices. The nature of images means that memory usage may become significant, and the Fourier transform will need both memory and DSP slices. The DSP slices are physical hardware needed to perform multiplications in an efficient manner. The elementwise multiplication performed in Equation 2.1 will also need DSP slices.

When operating with Fourier transforms and in the frequency spectrum, complex numbers are present. These complex numbers require more memory. This usually comes in the form of two times 32-bit. This means that if we have a 512 by 512 image and ignore padding, the memory required to store the frequency spectrum in PL is almost 17 MB. A large portion of the available memory. Optimizations should be implemented in order to minimize memory usage.

We will use the Xilinx FFT IP during this project. See Subsection 2.8.4. This means that we may be able to gain insight into the number of DSP slices that are needed based on Xilinx documentation. The smallest implementation of the IP uses four DSP slices, if we were to have a 512 by 512 image [27]. This would require 2000 DSP slices if the Fourier transforms were to be performed in parallel. However, this is not a realistic example since the memory bandwidth of 512 bits(Table 2.1) would be too small for doing the whole transformation in parallel.

The ability to easily parallelize the design is one of the main benefits when designing specified hardware, e.g. accelerators. However, depending on the application, one will meet hard resource limits. When this happens there are two options. Buy a better and more expensive FPGA if available, or serialize, i.e. do operations in sequence. Serializing naturally occurs when there is a memory bandwidth bottleneck. However, when there is a resource limit the design has to change accordingly and implement serialization when the need arises.

2.7 PYNQ

PYNQ is a combination of a Python library and a modified version of Ubuntu [28]. The customized operating system allows for the use of the Zynq FPGA architecture [28]. The Python library contains a feature set that makes it easy to upload and use accelerators on the FPGA. The Python library is the primary component of PYNQ. The FPGA may still be used like a general PS PL system, however, the PYNQ Python library aims to make FPGA accelerators more accessible for developers with little to no knowledge of embedded development with C. However, even for experienced embedded developers, PYNQ allows for rapid development and testing of FPGA hardware modules. This is why PYNQ is to be used in this thesis. However, the use of Python does come with some drawbacks. Python being a high-level language, where the underlying functionality is unknown, is one such problem. This makes debugging harder. Another drawback is the relatively poor performance of Python. However, the ease of development and testing of accelerators outweighs these factors. For a system meant for production, PYNQ should probably not be used.

2.7.1 Overlays

The developer of PYNQ states that an overlay is a "post-bit-stream configurable design" [29]. This bit stream is an output product from the hardware implementation process. The overlay is loaded by a class initialization from the PYNQ library. This means that the loaded hardware is handled like a class. This is again for ease of use and understandability for developers with no embedded accelerator experience.

When loading an overlay, it will have a default IP driver if no driver is given. This default IP driver may handle simple read-write operations to control the hardware. However, there are some common drivers already available. This includes a Direct Memory Access (DMA) driver, see Subsection 2.8.5, and a GPIO driver. The developer may write custom divers, however, this takes time. There are also some predetermined overlays, like the base overlay, that already have drivers. The base overlay is an overlay that enables the entirety of the IO on select developer boards.

2.8 Hardware Components

In Section 2.3 it was explained how to perform CC in the frequency domain. From this, we may observe that there is a need for complex elements in the implementation. This section will explain the functionality for the basic hardware functionality used in the implementation. The presented subsections contain the necessary basic components of the design needed to implement the suggested solution.

2.8.1 AXI-Stream Interface

When designing, a standard interface for the entire system should preferably be used. The interface between hardware modules in this thesis will be the AXI-Stream (AXIS) interface [30]. AXIS is a general purpose one way point-to-point transfer protocol. The interface is developed by Arm and is well-documented. The protocol uses a handshake process to ensure correct data transfer and may transfer data each clock cycle. In Figure 2.14 a simple AXIS example may be seen. In this example, there are the handshake signals *tvalid* and *tready*, the data signal *tdata*, and the last signal *tlast*. As seen, the data in the figure does not change until both handshake signals are high. The last signal is useful if there is a need for packet transfers, like in this project. There are further signals, however, they are not shown since they will not be used in this project and are not strictly required.

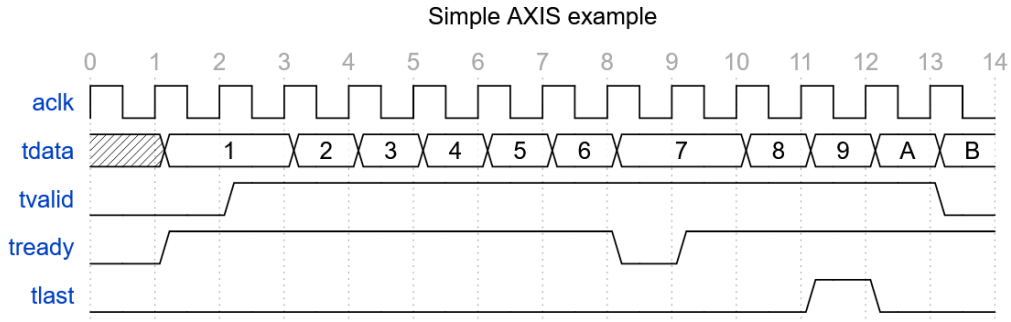


Figure 2.14: AXIS Timing example

The PS memory on the FPGA may be fed into PL via a DMA module. This DMA is a Xilinx IP that may access the memory and outputs as AXIS to the PL. Many other Xilinx IPs also support AXIS. Including the Xilinx FFT IP that is to be used in this project.

2.8.2 Xilinx DSP48E2

Multiplication is a resource-expensive task in hardware. Therefore the best solution is often to use dedicated hardware. The FPGA Zynq Ultrascale+ the physical hardware used for multiplication is called DSP48E2[31]. This is the physical hardware of the DSP slices mentioned in Section 2.6.

DSP48E2 is intended to make DSP applications both faster and more efficient. This means that the component is not only applicable for multiplication and has other functions as well. As a result, when designing the desired hardware modules, one should be aware of these functions. The DSP48E2 is versatile and has many functions, so only the components used in this thesis will be presented. These components are the pre-adder, the multiplier, and the multiply-accumulate functionality. A very simplified schematic, with extra features and every multiplexer removed, of the DSP48E2 may be seen in Figure 2.15.

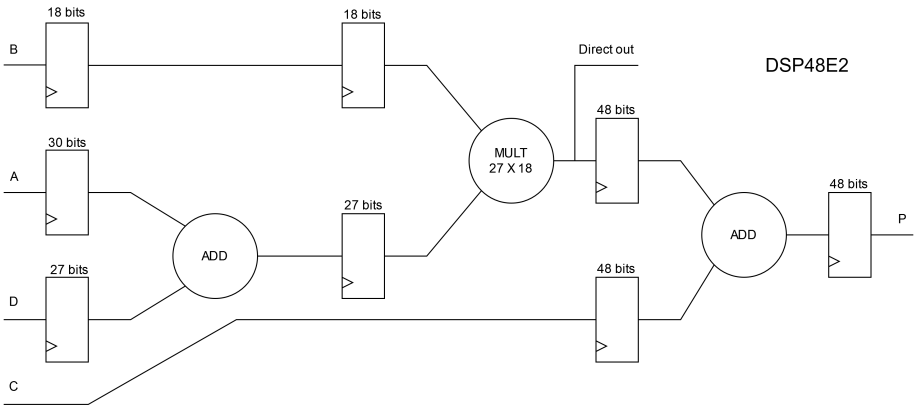


Figure 2.15: Simplified example of DSP48E2 functionality.

The multiplier may multiply up to 27×18 bits and may output a maximum of 48 bits. The pre-adder is an additional hardware component that may be used to employ an addition before the multiplication is performed. Multiply-accumulate is a common task in hardware calculus. This is an additional step after the multiplication one may use to accumulate multiplication results.

The DPS48E2 is used in the complex multiplier in subsection 5.1 and in the Xilinx FFT IP in subsection 2.8.4. However, the knowledge of the DPS48E2 is only employed actively in the complex multiplier.

2.8.3 CORDIC

The need for the absolute value arises when evaluating the result given by Equation 2.1. This is because the result is a complex number. The Coordinate Rotation Digital Computer (CORDIC) algorithm may be used to perform this [32]. CORDIC is a hardware-efficient algorithm used to calculate a wide range of elementary and trigonometric functions, including the absolute value of a vector [33]. The CORDIC algorithm utilizes the same basic principle as Newton's method [34]. This principle may be seen in Figure 2.16. This is where an angle is set and then evaluated if it is greater or smaller than the angle we want to measure. Based on that result the previously set angle is moved a half step of the previous angle in the correct direction and the same operation is carried out again. When this operation is carried out enough times, the set angle will converge upon the result. When setting a new angle, the new endpoint of the vector will be perpendicular to the old vector's endpoint. This means that there is gain when performing CORDIC. This is called CORDIC gain, and it converges to a constant after enough iterations.

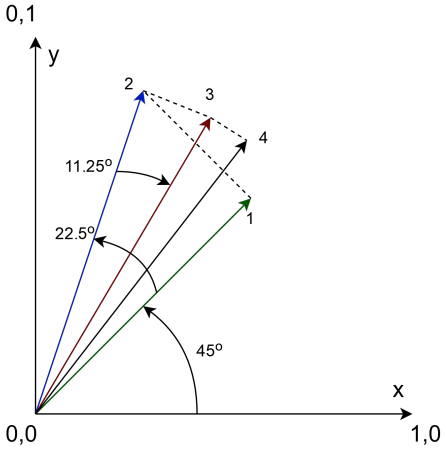


Figure 2.16: CORDIC algorithm illustration.

The mathematical definition for performing vector length operation with the CORDIC algorithm may be seen in Equation 2.5. For every iteration, a right bit shift is performed for division, and based on the sign of y_i an addition or subtraction is performed. When $y_n = 0$ the result will be correct. K is the CORDIC gain.

$$\begin{aligned} x_{i+1} &= x_i - y_i d_i 2^{-i} & x_n &= K\sqrt{x_0^2 + y_0^2} \\ y_{i+1} &= y_i + x_i d_i 2^{-i} & y_n &= 0 \\ d_i &= \begin{cases} +1, & y_i < 0 \\ -1, & otherwise \end{cases} \end{aligned} \tag{2.5}$$

If we use the definition in Equation 2.5 we may create the iterative schematic of the algorithm seen in Figure 2.17. This is the hardware-efficient schematic. However, one may pipeline this iterative process and still use less hardware than what is needed for the multiplication and square root operations needed to do it natively [33].

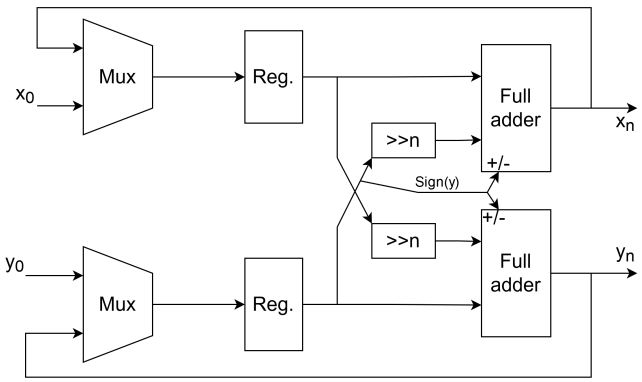


Figure 2.17: CORDIC iterative hardware schematic.

2.8.4 Xililnx FFT IP

Due to the relative complexity of the FFT it is desirable to use the FFT IP provided by Xilinx. This makes development easier and the finished product less prone to errors. The current FFT IP provided by Xilinx is the Xilinx LogiCore IP Fast Fourier Transform v9.0 [35]. This is an IP that implements the Cooley-Tukley FFT algorithm. The IP supports widths of 8-34 bits for both real and imaginary numbers, and transform sizes of 2^m where $m \in$ [3,16]. One may use fixed-point, scaled or unscaled, or block floating-point numbers. For each butterfly step, see [18], in the FFT the bit width will increase. Therefore, if the input and output are to be the same bit width, scaled fixed-point or block floating-point has to be used.

There are four different architectures available. Radix-2 lite burst IO , radix-2 burst IO, radix-4 burst IO or pipelined streaming IO. Pipelined streaming IO is a pipeline of radix-2 butterflies. For more information on radix butterflies, see [35] and [8, p.519]. From these options, one may select the best implementation based on resources and use cases. The radix-2 lite burst IO is the least resource-demanding, however, delivers the lowest performance of the available architectures. The pipelined streaming IO delivers the best performance, however, needs a substantial amount of resources depending on transform length.

2.8.5 Xilinx DMA IP

The Xilinx DMA IP is used to connect memory in PS to PL peripherals with an AXIS interface. The connection to memory is direct and has a high bandwidth. The DMA supports memory map and AXIS width from 32 to 1024 bits depending on the FPGA used. Even though the Zynq Ultrascale+ has a maximum bandwidth of 512, see Sectom, it does still support stream widths of 1024 bits. The AXIS will however not be sending new data each clock cycle if the stream width is wider than 512 bits.

The IP is controlled by an AXI interface. It is nevertheless unfortunate that this AXI interface and the high-performance AXI masters controlled by the DMA do not support clock speeds of 333 MHz and up. This makes the design more complex if it is desired to have greater clock speeds. The average implementation of the FFT IP achieves a maximum clock speed of 456 MHz [27]. If that clock speed were to be used, several clocks would have to be implemented. However, if the design is fully pipelined there would not be any meaningful improvements. The latency would be lower, while the throughput would be the same.

Chapter 3

Partial Cross-Correlation

This chapter will present the proposed solution of Partial Cross-Correlation (PCC) for performing fast Cross-Correlation (CC) on data sets with a large number of speckle images. The main effort of this project has gone into the development of this method. This chapter will present the concept of the method and how the method works. There will also be presented benefits and limitations of the method.

3.1 Problems With the Normal Approach

The images that are to be cross-correlated can be preprocessed into the frequency domain. Thus we will ignore the time required to do this task. Figure 3.1 shows a flow diagram of cross-correlating over a large number of images. To cross-correlate a single image with a large number of images there is a single Fourier transform into the frequency domain. However, there are as many inverse Fourier transformations as there are images to be cross-correlated. When performing a CC in the frequency domain one has to transform the result of the CC to the spatial domain in order to be able to evaluate the result(Section 2.3). It is very inefficient if most of the calculations do not yield a result.

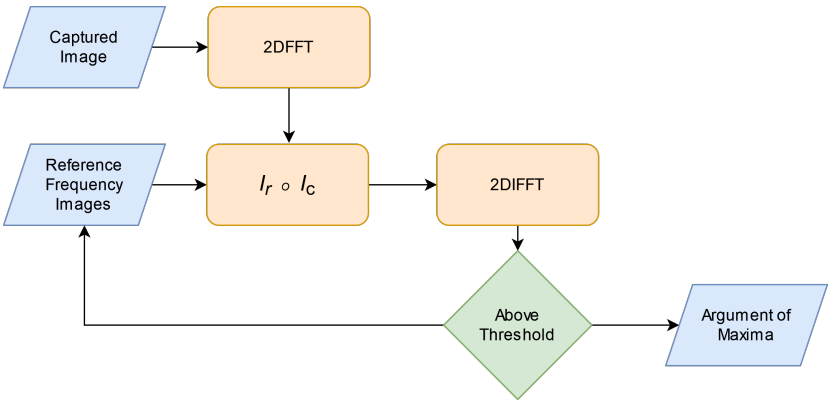


Figure 3.1: Flow diagram of cross-correlating a large number of images.

The computation needed for CC, in big O notation, may be seen in Equation 3.1 where N is the height and width of the image. This makes a single CC relatively slow and will increase linearly when adding several CCs.

$$N^2 Log_2(N) \tag{3.1}$$

For a hardware implementation of the normal approach, memory is very important. This is because the Zynq Ultrascale+ FPGA may not store the entire frequency spectrum in a good manner in the on-chip memory. The total capacity of on-chip memory would perhaps not be used, however, the FFTs and other parts of the design also require a substantial amount of on-chip memory. There will also be a need for a substantial amount of logic if the design is to use AXIS. Therefore, external memory should be used since there is a memory socket connected directly to PL(Section 2.6). This will eliminate the need for a substantial amount of memory and make it easier to create a larger design if needed. This would, however, add to the complexity of the design since a second interface would need to be implemented. The same memory should also be used to store the frequency spectrums that are to be correlated.

3.2 Partial Cross-Correlation

The transformations from the frequency domain to the spatial domain are the main problem when cross-correlating a large number of images. Therefore Partial Cross-Correlation (PCC) is proposed to eliminate the need for a full two-dimensional inverse FFT for every CC. PCC uses two one-dimensional inverse FFTs and two array multiplications to calculate the offset. PCC results in less accuracy if padding is used(Section 4.2), however, it will give a significant speedup, reduction in data, and resource costs.

3.2.1 The Interference Pattern

In Subsection 2.4.4 it is shown that the offset in the spatial domain may be represented in the frequency spectrum as an interference pattern. In Figure 3.2 the frequency spectrum of the CC of two speckle images may be observed. This figure is modified so that the waves in the interference pattern have the same average. The presence of this interference pattern makes it possible to remove most of the frequency spectrum. If a cross-section of the frequency spectrum is examined, one may see a wave pattern. This wave pattern may be employed to evaluate the result of the CC in a very efficient manner.

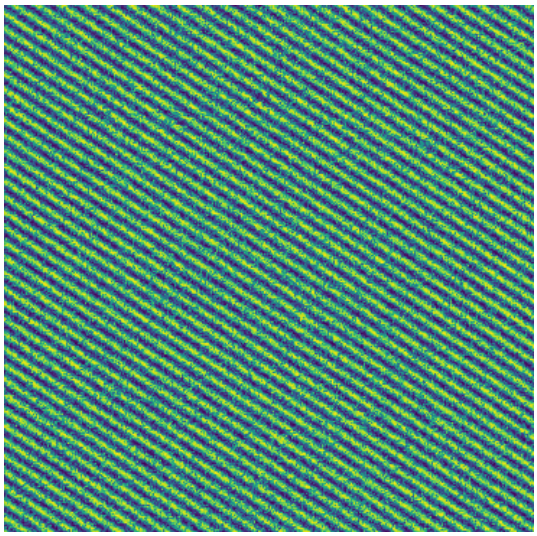


Figure 3.2: The frequency spectrum after elementwise multiplication.

At an early stage of the development, the aim was to use these waves to evaluate if there was a correlation with the accuracy of true or false, i.e. if there is a correlation or not. A strong correlation would give a large magnitude of a single frequency compared to every other frequency. It was noticed that this single frequency corresponds to an exact offset. Thus we may be able to calculate the correct offset in an efficient manner.

3.2.2 The Method

The general concept of PCC is to calculate the offset using only parts of the frequency spectrum. This is done by calculating the CC on a single row and column. One array for each dimension. Two one-dimensional inverse Fourier transforms can now be used to transform back to the spatial domain. This reduces the computation drastically.

The first step of PCC is the same as the normal approach. First, transform the images that are to be correlated to the frequency spectrum. This may be seen in Equation 3.2 where the images f and g are transformed to the frequency spectrum.

$$F = \mathcal{F}\{f\}, \quad G = \mathcal{F}\{g\} \tag{3.2}$$

PCC starts to differ in the multiplication step. The elementwise multiplications may be seen in Equations 3.3 and 3.4. The $*$ denotes that it is the complex conjugate and $\circ$ is the Hadamard product (elementwise multiplication). These equations correspond to a cross-section of the frequency spectrum in each dimension (x and y), i.e. a row and a column. I is the resolution factor. The resolution factor has to be configured depending on the speckle size. If the speckles are large, the resolution factor has to be closer to zero(Section 2.5). We may change this factor depending on what kind of details are of interest and if the speckle size

allows it(Section 2.4.3). If the resolution factor is set incorrectly it may yield poor results. In Figure 3.3 the frequency spectrum cross-sections are represented visually. This figure assumes equal height and width of the image, however, it does not matter if they differ. Note that the frequency spectrum is not FFT shifted.

$$R_y = F(I, y) \circ G^*(I, y) \tag{3.3}$$

$$R_x = F(x, I) \circ G^*(x, I) \tag{3.4}$$

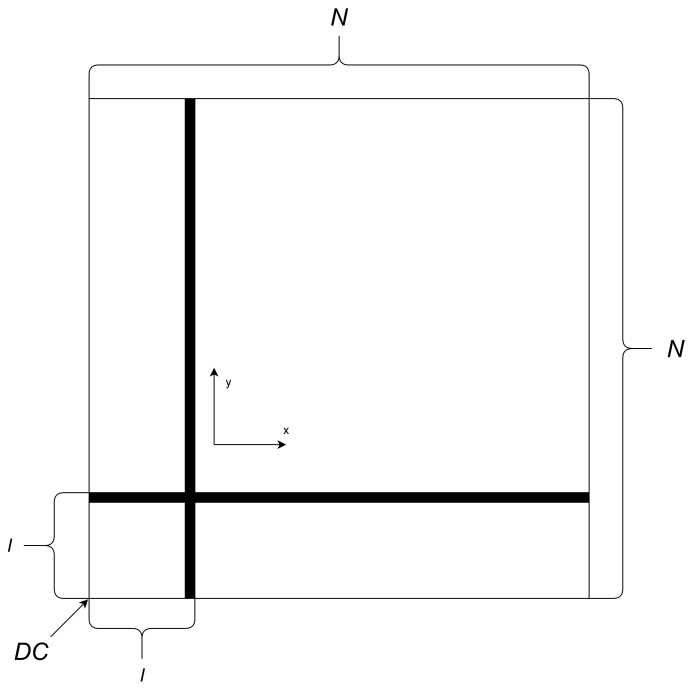


Figure 3.3: The frequency spectrum cross-sections selected to perform PCC.

It has to be noted that if the resolution factor I is set to zero one has to take some precautions. This is because the DC value in the frequency spectrum will most likely saturate the result in turn making it difficult to evaluate. To circumvent this problem, simply set the DC frequency to zero.

After the multiplication is executed, it can be transformed into the spatial domain for the extraction of results. In Equations 3.5 and 3.6 these steps are shown. These are two one-dimensional inverse Fourier transforms, one for each dimension. The same is true for retrieving the calculated offset with the help of the argument of maxima. To evaluate if the calculated result is correct, examine the magnitude of the offset. If the offset value surpasses a given threshold, approve the result. This given threshold needs to be properly tested and calibrated.

$$\begin{aligned} r_x &= \mathcal{F}^{-1}\{R_x\} \\ r_y &= \mathcal{F}^{-1}\{R_y\} \end{aligned} \tag{3.5}$$

$$\begin{aligned} \Delta x &= argmax(|r_x|) \\ \Delta y &= argmax(|r_y|) \end{aligned} \tag{3.6}$$

PCC is also shown in Algorithm 1. This shows how PCC could be implemented in a system. Either hardware or software. PCC is relatively simple to implement and the complexity only increases marginally compared to the normal approach.

Algorithm 1 Partial cross-correlation

$i \leftarrow 0$
$F_c \leftarrow \mathcal{F}\{Captured\ image\}$
$F_x \leftarrow F_c[0 : width,\ I]$
$F_y \leftarrow F_c[I,\ 0 : height]$
while $!Found$ **do**
  $G_x \leftarrow Stored\ reference\ row[i]$ ▷ Frequency spectrum, complex cojugate.
  $G_y \leftarrow Stored\ reference\ column[i]$
  $R_x \leftarrow F_x \circ G_x$ ▷ Elementwise multiplication.
  $R_y \leftarrow F_y \circ G_y$
  $r_x \leftarrow \mathcal{F}^{-1}\{R_x\}$ ▷ Inverse Fourier transform (1D).
  $r_y \leftarrow \mathcal{F}^{-1}\{R_y\}$
  if $max(r_x) + max(r_y) > Threshold$ **then**
    $Found \leftarrow True$
  else
    $i \leftarrow i + 1$
  end if
end while
return $Argmax(r_x), Argmax(r_y)$ ▷ Displacement vector

3.2.3 Computational Costs

The calculations needed to perform PCC are significantly less than what is needed for normal CC in the frequency domain. In big O notation, the complexity is reduced to that of a one-dimensional FFT, as seen in Equation 3.7. This makes PCC faster by a large margin. In Equation 3.8 the reduction of complexity may be observed. With a reduction of N, PCC compares well with normal CC in the frequency spectrum.

$$NLog_2(N) \tag{3.7}$$

$$\frac{N^2 Log_2(N)}{NLog_2(N)} = N \tag{3.8}$$

The reduction of complexity has been presented in big O notation. However, this removes some details. The complexity is dominated by the FFTs. The elementwise multiplication is removed. This elementwise multiplication is also reduced, however, not by as much as the FFTs are. Equation 3.9 shows that the multiplication is reduced by $N/2$. This is not as much as the reduction of the Fourier transforms, however, it is still significantly less than the Fourier transforms.

$$\frac{N^2}{2N} = N/2 \tag{3.9}$$

3.2.4 Benefits

From both a hardware and software perspective, the most prominent benefit is the reduction in memory requirements. The fact that it is $N/2$ fewer points that are to be calculated makes PCC faster by nature. This is not only beneficial for performance. The space needed to store the frequency spectrum is also significantly reduced. If a complex number requires 64 bits, the storage needed for a frequency spectrum of a 512 by 512 image is 2 MB. This gives 1907 frequency spectrums one may store in 4GB RAM. PCC requires $N/2$ less data, which in turn gives 488281 frequency spectrums one may store in the same amount of RAM. This ability to impact memory usage is very beneficial. Unfortunately, the memory bandwidth usage is not affected as much. However, we are now able to store the captured image's frequency spectrum in on-chip memory in an easy manner due to its small size. This doubles the number of frequency spectrums one may stream into PL.

PCC is relatively simple to implement in hardware compared to the normal approach. This is due to the need for reordering(rotate 90 degrees) when using a two-dimensional FFT(Subsection 2.4.1). Reordering requires extra logic and may not be completely pipelined. The reordering would additionally make the design more complex than PCC. PCC requires a single FFT stage per dimension. The normal approach would require the same number of FFT stages as the image size. These FFT stages may not be fully parallelizable since the FPGA available cannot support that many FFT stages in parallel. Thus PCC will be faster by not needing to serialize the design.

3.2.5 Disadvantages

PCC is designed for speckle images. PCC works due to the fact that the frequencies in speckle images are evenly distributed in a cone shape. Normal images will not have this property. In Figure 3.4 this difference may be observed. The image to the left is a Fourier transform of a normal image, while the one to the right is a Fourier transform of a speckle pattern. PCC may work on other images. See Subsection 4.3.2. However, PCC is more likely to succeed if the images have an even distribution in frequency.

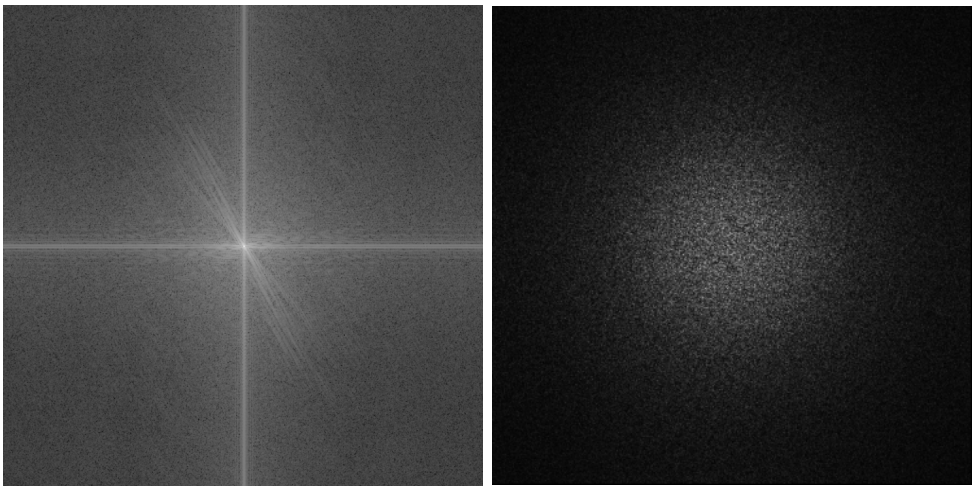


Figure 3.4: Fourier transform of the author(L) and a speckle pattern(R).

Noise may pose a problem for PCC since only parts of the frequency spectrum are used. This makes the result more amendable to change due to the introduction of noise. In Section 4.1, this is shown to not be a detrimental feature. The result changes more with PCC than with the normal approach, however, not enough to determine the viability of PCC. Nevertheless, this may change if the images that are to be cross-correlated change or if the noise characteristics change from white Gaussian noise to something else.

PCC is made for use with images of equal sizes. Little testing has been done on images with differing sizes. However, this limited testing has yielded good results. See Appendix A.2. Due to this limited testing, some more research is needed.

3.3 Novel Two-Dimensional FFT

The two-dimensional FFT consists of performing a series of FFTs in first in one dimension and then in the other dimension(Subsection 2.4.1). This has the inherent flaw of requiring intermediate storage of the entire image. If the image size requires more on-chip memory or resources than is available, it has to be stored in off-chip RAM. For the image size of 512 by 512, there is sufficient on-chip memory on the Zynq Ultrascale+. However, the control logic needed to support this amount of memory, in a simple manner, exceeds what is available. If possible one should try to remove this need for intermediate storage.

Due to the fact that PCC only needs a single row and column in the frequency spectrum, a novel way of performing a two-dimensional FFT resulting in a single row or column has been developed. Removing the need for a complete two-dimensional FFT. This method, shown in Figure 3.5, performs the first series of FFTs followed by a single FFT. As may be observed, performing this will give a single column. As a result, this operation will have to be executed twice to get both row and column. However, the input image has to be rotated 90 degrees

to obtain the row. This method removes the need for intermediate storage. The single row and column that is to be transformed is set by the resolution factor I.

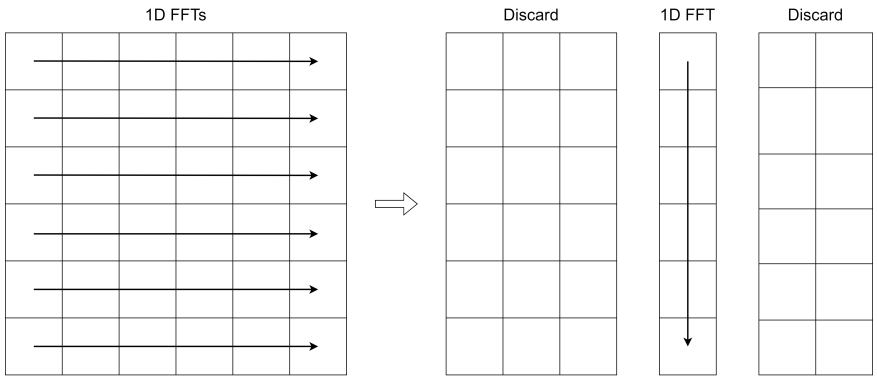


Figure 3.5: Novel way of performing two-dimensional FFT resulting in a single row or column.

The memory bandwidth needed for the approach in Figure 3.5 is greatly reduced. This makes a single transformation faster. However, the same amount of FFTs still need to be performed. The same amount of computation for significantly less data is not ideal, however, it fits the task of PCC. In a Python implementation, there is no need for the use of this novel method. The two-dimensional FFT implementation in NumPy is faster than looping through single FFTs.

Chapter 4

High-Level Testing

To validate the functionality and applicability of PCC, it has been implemented and tested with Python. This Chapter will present the results given by this testing in Python. Python is versatile and enables easy rapid development of an algorithm. The Python library NumPy was used for performance improvements over regular Python. NumPy includes common mathematical tools as well as an FFT implementation.

Every image used in testing is 512 by 512, and the resolution factor(I) for PCC was set to 96. The minimum speckle size was set to 4. For larger speckles, the resolution factor should be reduced. We will not be performing normalization nor have any padding for wraparound handling.

Speckle Images

The speckle images used under testing differ somewhat from real speckle images. In Section 2.5 we could observe this difference. It was also mentioned that we may change the contrast to lessen this difference. In Figure 4.1 we may observe a manually adjusted and a calculated contrast. The manually adjusted contrast does indeed look similar to the real speckle image in Section 2.5. However, it is still decided to use the calculated contrast for testing. This allows for consistent test data even if we change the size of the image and individual speckles. In Appendix A.1 the difference in the results is shown. The takeaway from those results is that using calculated contrast for testing is safe. However, it has to be noted that there is a reduction in SNR for manually adjusted contrast since the image is darker.

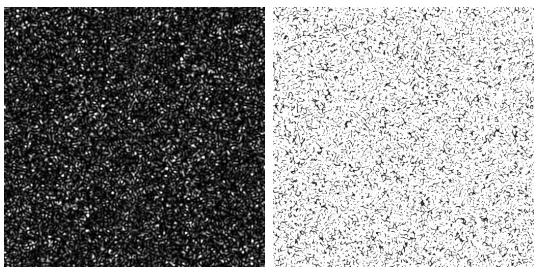


Figure 4.1: Difference in manually adjusted(L) and calculated(R) contrast.

4.1 Performance and Accuracy of Partial Cross-Correlation

The testing of PCC performance in software was done using a simple Python program. This program runs a for-loop from zero to 1023. In each iteration, an elementwise multiplication and an inverse Fourier transform is performed. The length is arbitrary, however, this is the length used for FPGA testing in Section 6.3. The test was run on three different systems. Table 4.1 shows the results of these tests. The Arm system performs the least speedup, however, is also the lowest-performing system. The speedup on the Intel and AMD systems is both over 400 in speedup. PCC has 487 fewer computations than the normal method. This makes the range of the results what we expect.

The higher speedup in the Intel system is somewhat surprising since it is a lower-performing system than the AMD system. The reason for this may be due to thermal limitations making the normal approach slower, and opportunistic boosting making PCC faster.

Hardware	Normal	Partial	Speedup
AMD Ryzen 3900 XT	11.0 ms	26.7 µs	412.4
Intel i5-8250U	22.8 ms	50.3 µs	453.3
Arm Cortex-A53	114.4 ms	367.5 µs	311.3

Table 4.1: Average speedup and time per iteration.

The accuracy of PCC was measured by performing a CC on every offset that does not give an ambiguous result. See Section 2.4.5. This gives 512 by 512 different images and offsets. White Gaussian noise of different levels was also added to the images to see its effect on the accuracy. The noise level is represented in standard deviation. In Table 4.2 the result of this test may be seen. As may be observed, PCC has excellent accuracy within the given area. The first faulty offset calculated is when the noise level has a standard deviation of 20 %. This level of noise far exceeds the level of noise that is present in a normal camera [36]. The signal used to calculate the SNR in [36] is 55 % lower than the signal given by synthetic speckle images. However, all SNR values in [36] are still larger than the calculated SNR values in Table 4.2. Normal CC has a 100% accuracy on all noise levels.

Noise	SNR	Faults	Accuracy
0 %	NA	0	100 %
5 %	25 dB	0	100 %
10 %	19 dB	0	100 %
20 %	13 dB	11	99.99 %
30 %	9.5 dB	2713	98.86 %

Table 4.2: Accuracy with different levels of white Gaussian noise.

It has to be noted that the faulty offsets were close to the corners of the given area. This may

be observed in Figure 4.2 where the offset data is represented as an intensity. The pattern is due to that the result is not shifted(Section 2.4.2). As intensities, it is easy to observe the faults as the wrong offsets break the pattern. This faulty behavior may be explained by the fact that the correlation computation calculates a lower maximum toward the corners. This may be seen in clearer detail later in Section 4.3.

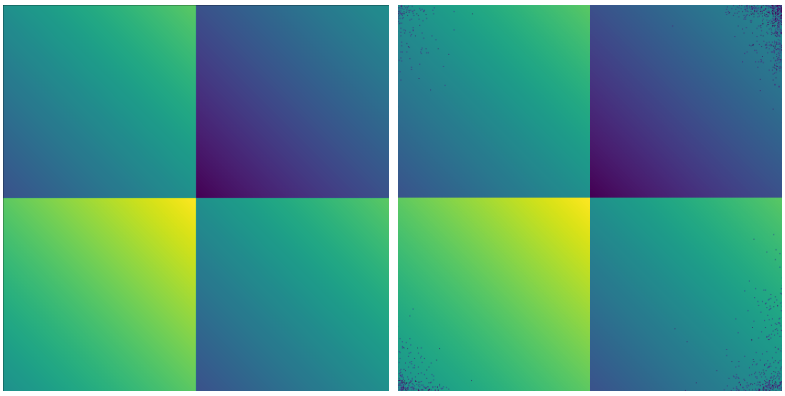


Figure 4.2: PCC offset data as intensities without(Left) and with(Right) noise ($\sigma = 30\%$).

4.2 Comparison Between Normal and Partial Cross-Correlation

The results in Section 4.1 are very good. However, it is also desirable to compare the results of normal CC and PCC. Evaluation of the quality of the result is required because of the need for a threshold. The accuracy of the system does not matter if we are unable to evaluate if the calculated result is correct or not. The test used for comparison may be seen in Figure 4.3. The image of the center is compared to every offset diagonally. This means that for the size of 512 by 512 we are performing 1024 correlation computations. From this test, we may compare the magnitude of the correlations and the offsets given.

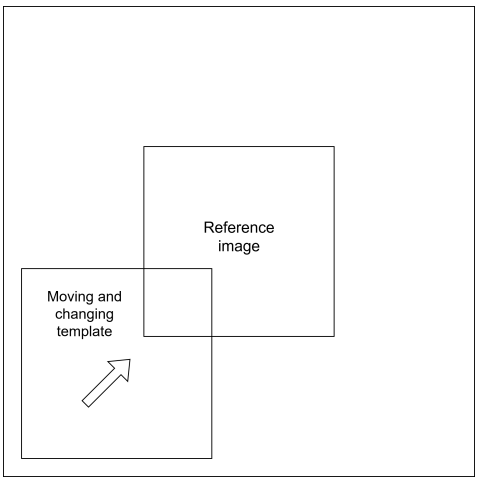


Figure 4.3: Performing PCC on every offset diagonally.

Figures 4.4, 4.5, and 4.6 show the correlation strength, i.e. the magnitude, and the calculated

offset of the CCs. The change in correlation strength when using PCC is very noticeable. The same is true when applying noise, however, in a different manner. One may also observe that it is still possible to evaluate when the result should be correct. The same is true when analyzing the offsets. The red dots are ambiguous offsets due to wraparound while the blue dots are correct results. The red dots overlap the blue ones due to the calculation giving the correct, but, ambiguous result. As observed, the biggest change in accuracy is when changing from normal to PCC. This change is not noticed in Section 4.1 due to not evaluating the ambiguous results. There is also a change when applying noise, however, not as much. The noise seems to make the incorrect results more independent. In summary, these results are good since we are able to set a threshold that will guarantee an exact offset.

Normal Cross-Correaltion

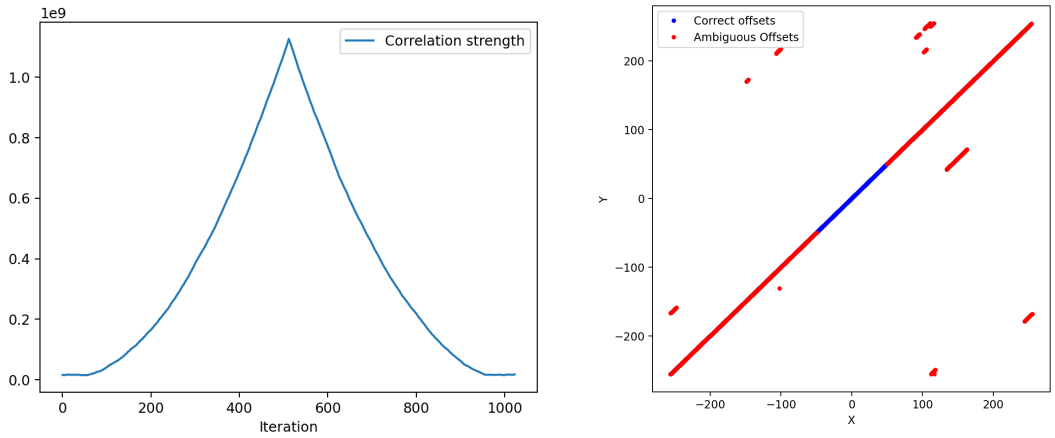


Figure 4.4: A benchmark showing the magnitude and offsets using normal CC.

Partial Cross-Correlation

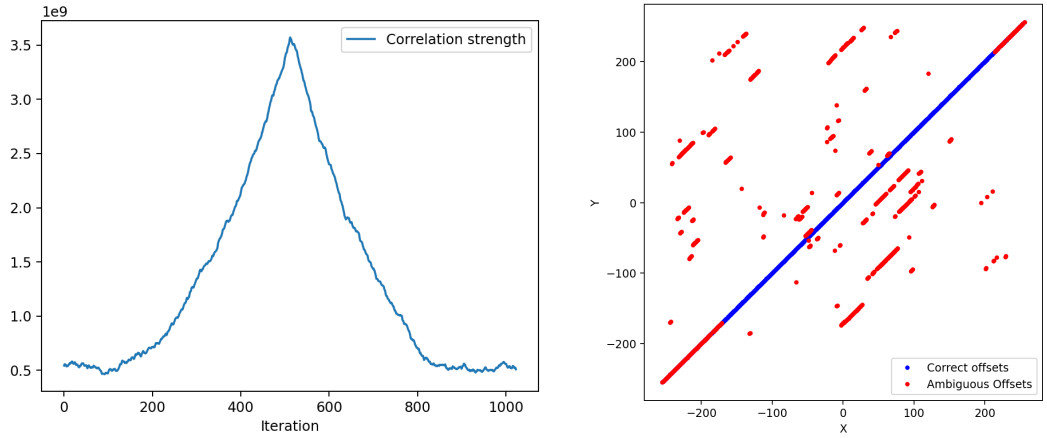


Figure 4.5: The magnitude and offsets using PCC.

Partial Cross-Correlation with Noise

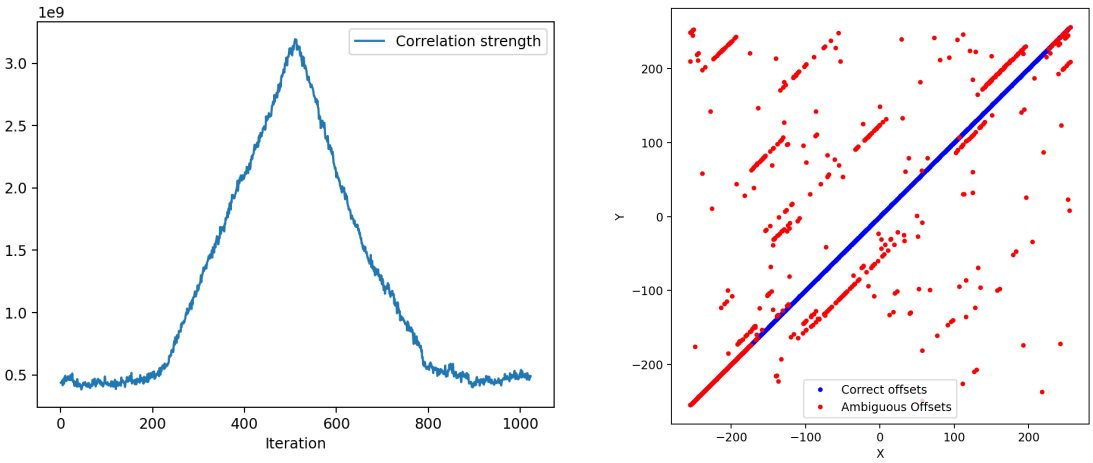


Figure 4.6: The magnitude and offsets using PCC with noise applied($\sigma = 10\%$).

4.3 Correlation Maps

In the previous section, we observed the CC of the correlation of the diagonal offsets. However, it is also desired to observe the behavior of other offsets as well. Consequently, the correlation maps in Figure 4.7 were created. These are intensity maps of the correlation strength of all possible offsets. This means that the correlation maps represent 1024 by 1024 offsets. The maps in Figure 4.7 show that the difference between normal and PCC is in the details. The overall shape and intensity are the same. However, there is a noticeable grid-like pattern in the map using PCC. One may also notice that the lower, i.e. dark, values are more noisy.

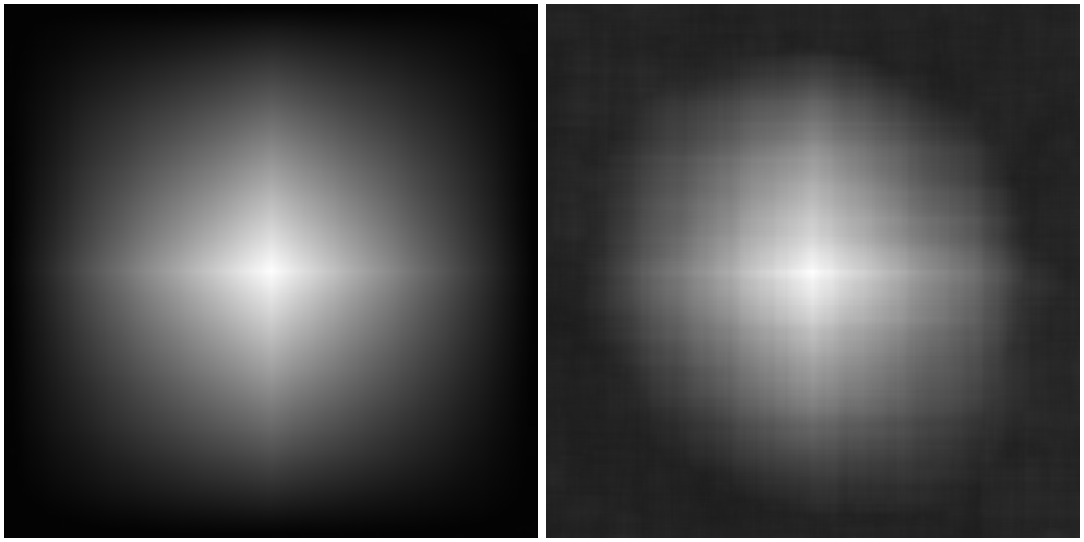


Figure 4.7: Correlation intensity maps. Normal(L) and partial(R) cross-correlaiton.

From the shape of the intensity, we can see that the first point of failure is at the corners.

I.e. there is more darkness at the corners. This is due to the corners being the area where there is the least overlap. These are the areas with the faulty behavior in the accuracy test in Section 4.1.

4.3.1 Threshold Given Area

If a threshold were to be used to avoid wraparound errors there would be a loss in usable area. I.e. a reduction in usable offsets. This may be seen in Figure 4.8 where, within the non-ambiguous area, the correlation strength intensities under a certain threshold are removed. The remaining area covers roughly 60% of the original area. This means that there is a need for overlapping images if this method is to be used. This approach may be used instead of padding. The threshold would need to be calibrated to

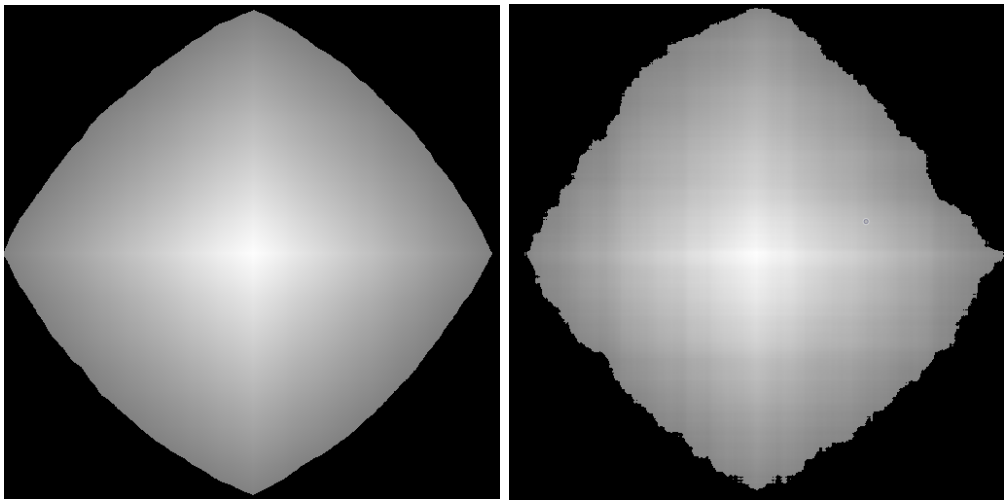


Figure 4.8: Correlation intensity maps with a set threshold. Normal(L) and partial(R).

4.3.2 Other Images

PCC is developed and so far used only on speckle images. PCC relies heavily on the fact that the frequencies are distributed evenly in a cone shape. Therefore it would be interesting to see how PCC performs with normal images. Consequently, the correlation maps in Figure 4.9 were created. As may be observed, the results are mixed. The maps show that PCC is not consistent, however, there is definitely a potential. With further development and optimization, it may be possible to use PCC. However, already, one may potentially use PCC to evaluate if the two pictures are equal. When the pictures are completely overlapping, there is consistently high intensity. The images used were, from top left to bottom right, an image of a starlit sky, a woman with a blurry background, a surfer in the ocean, and a village with fields in front.

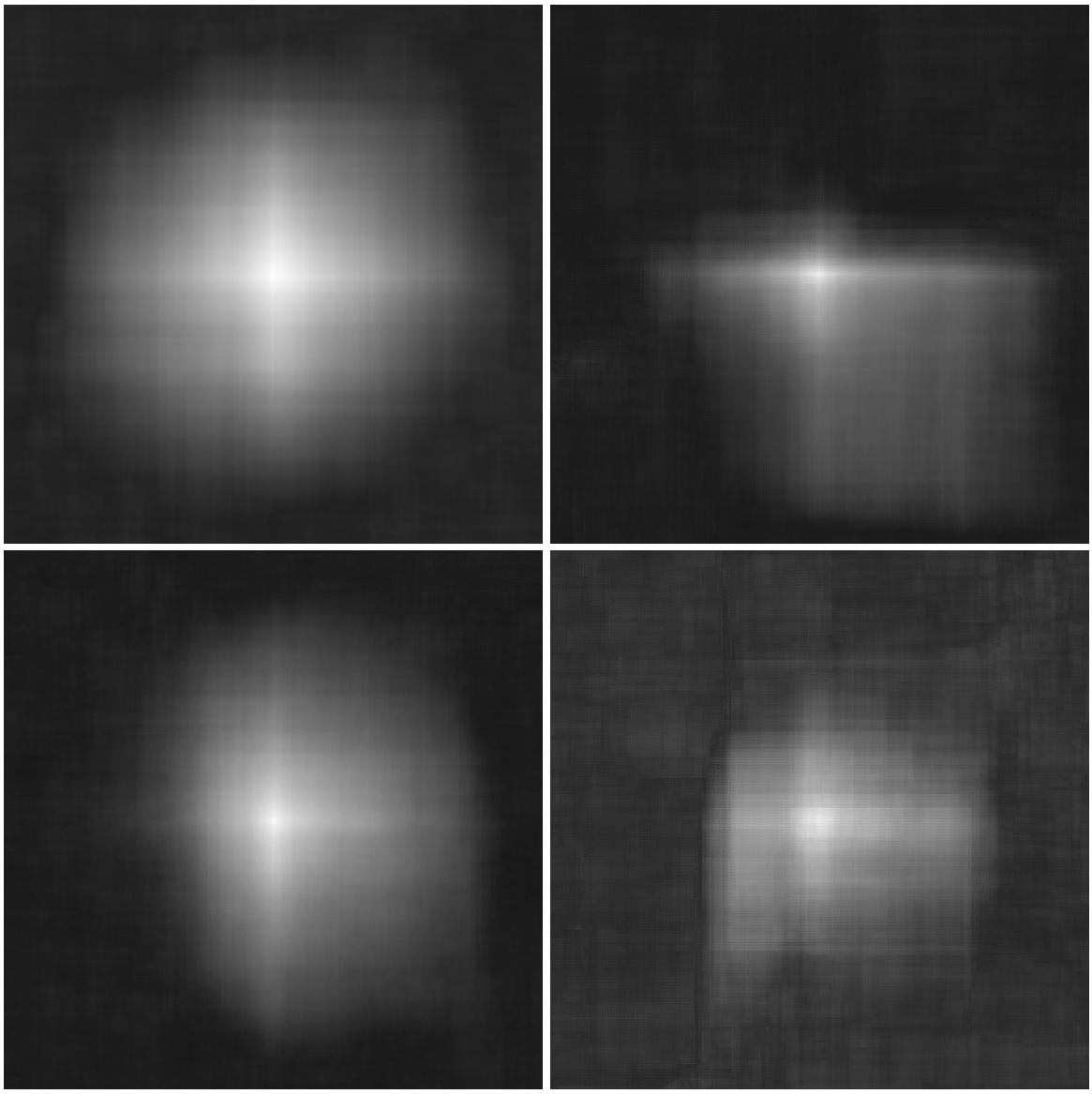


Figure 4.9: Correlation intensity maps using PCC on normal images.

Chapter 5

Hardware Implementation

The hardware modules made during this project will be presented in this chapter. The components needed to perform PCC will be presented first. Then following, a complete PCC and a novel two-dimensional FFT implementation will be presented. Each module will be presented in a broad manner where finer details are omitted. See Appendix C for more details. Every module uses the AXIS interface. Figures do not show the AXIS signals. The modules have differing data input widths, however, every module follows the same AXIS scheme.

5.1 Complex Multiplier

Since the multiplication performed in Section 2.3 is in the frequency domain we do have a need for complex multiplication. Multiplication has a large hardware resource demand, and regular complex multiplication requires four multiplications and two additions as seen in Equation 5.1. Consequently, it is desirable to optimize this task.

$$R + Ij = (a + bj)(c + dj) = (ac - bd) + (bc + ad)j \tag{5.1}$$

5.1.1 Gauss/Karatsuba Algorithm

There exist different solutions for this problem, however, for this project, the chosen solution was the one that was simple to implement and that fits the characteristics of the DSP48E2 as explained in Subsection 2.8.2. The following fast complex multiplication in Equation 5.2 and 5.3 was employed [37]. This method calculates intermediate values to be able to reduce the number of multiplications by one by adding three additions. This method is a variation of the Gauss/Karatsuba algorithm for multiplying complex numbers [38].

$$\begin{aligned} k_1 &= c(b - a) \\ k_2 &= a(c + d) \\ k_3 &= d(a + b) \end{aligned} \tag{5.2}$$

We may also observe that the intermediate values in Equation 5.2 may be split further down due to the additions. This fits the design of the DSP48E2 in Subsection 2.8.2 and we may use the existing pre-adders in each use case. In Equation 5.3, after the intermediate values are computed, these may be used to compute the real and imaginary numbers individually.

$$\begin{aligned} R &= k_2 - k_3 \\ I &= k_1 + k_2 \end{aligned} \tag{5.3}$$

5.1.2 Implementation

Using the Gauss/Karatsuba algorithm will result in the RTL schematic in Figure 5.1. As we can see, each multiply has a single preceding addition and one following addition. This structure fits the DSP48E2, see Figure 2.15, in a good manner, and only one of the six available hardware additions is left unused.

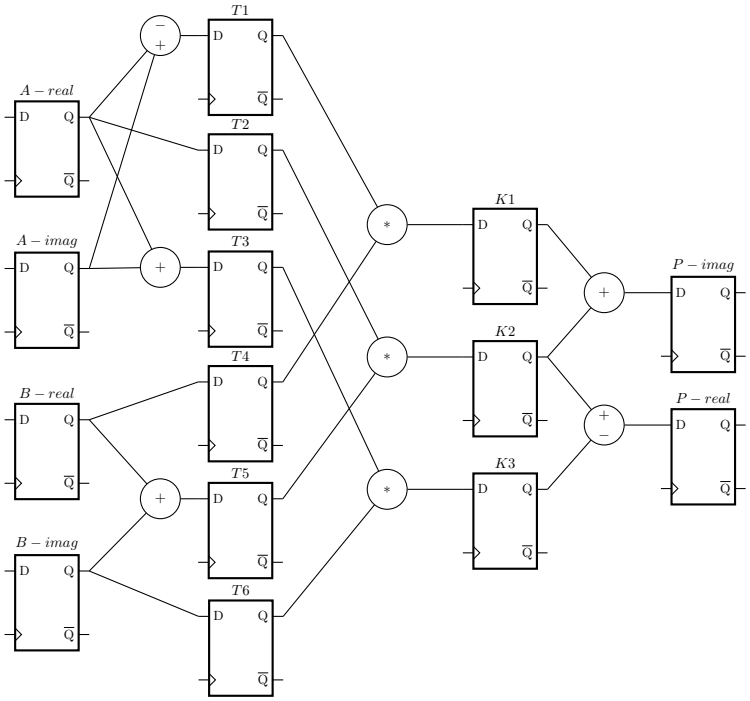


Figure 5.1: RTL schematic of a complex multiplier

When implementing multiplication or addition in hardware, it is important to be aware of the bit growth. For addition, the bit width will increase by one, while for multiplication the bit width will be the sum of the incoming bit width. The Gauss/Karatsuba method has three steps where there is first an addition, then a multiplication followed by another addition. This would result in $2 + \textit{input width}$ bit expansion. However, if we examine the normal method for complex multiplication, there is only one addition step after the multiplications. So the theoretical bit growth is only $1 + \textit{input width}$.

Truncation and if necessary, rounding, is an important aspect when we have bit growth. In this implementation, simple truncation has been used. Truncation is applied to both the input and output of the complex multiplier. The input has truncation so no more than three DSP slices are used. A single DSP slice may multiply 27 by 18 bits. Consequently, there will be a need for extra DSP slices if the input width is over 18 bits. In the implemented configuration, the most significant bits are truncated. However, another range may be configured. This configuration is due to the resolution factor(see Section 3.2.2) being set for higher frequencies. This means that the magnitude will be lower. The range of truncation has to be set according to the speckle characteristics. The output has truncation so the bit width will be equal to that of the Xilinx FFT.

5.2 CORDIC

For the calculation of the absolute value of complex numbers CORDIC is used. CORDIC is a resource-effective and relatively simple to implement. The implemented solution uses the same design as presented in Section 2.8.3, however, with some changes. The first major difference is that we need input processing since the CORDIC algorithm does not support angles over 90 or under negative 90 degrees. Therefore the real numbers at the input are changed to positive if negative. This has no effect on the magnitude. The RTL schematic of the input stage may be seen in Figure 5.2. The output stage of the CORDIC module is also modified by removing the scaling of the CORDIC gain(Section 2.8.3). This is done since the exact value does not matter. It is only the relation of the calculated values that matters. The scaling is constant and therefore ignored.

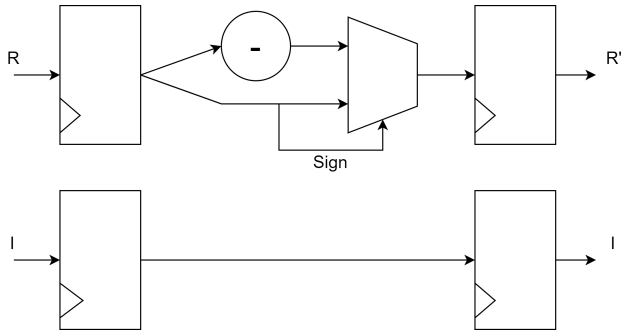


Figure 5.2: RTL schematic of CORDIC input stage.

For performance, the CORDIC implementation is pipelined as may be seen in Figure 5.3. This figure is an example with two stages, however, the implemented solution has 13 stages. The implemented solution has a higher resource cost than the iterative, however, may supply results each clock cycle when saturated. The relationship between resource cost and the performance gain is one-to-one. If the area is doubled the performance will double.

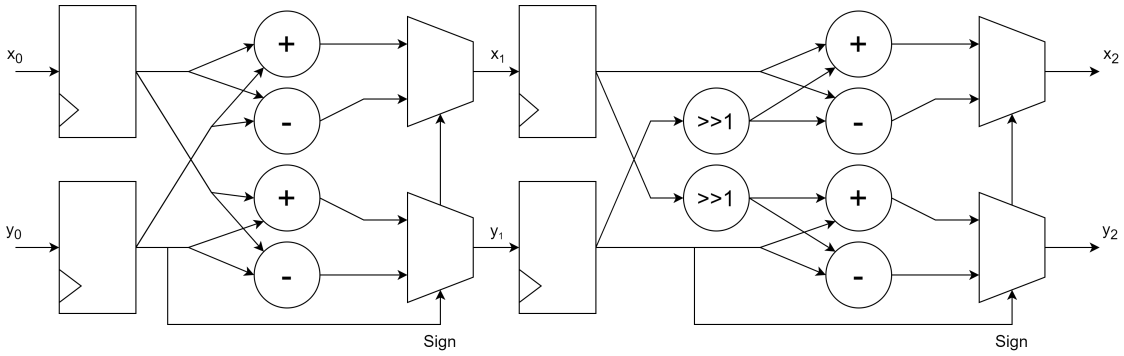


Figure 5.3: RTL schematic of a pipelined CORDIC for better performance.

5.3 Argument of Maxima

The output of the PCC module is to be the argument of maxima and its associated magnitude. Therefore the argument of maxima module was created to perform this task. This module was intended for both finding the argument of maxima and evaluating the magnitude with a threshold. For testing and the FPGA implementation, this was unfortunately not done. It was more practical to evaluate all the results in software than it was not receiving anything at all.

In Figure 5.4 the core functionality of the module may be observed. Every incoming value is compared to the previous highest value, and if the new value is higher the new high is set. This is accompanied by a counter that indexes the captured value, i.e. the current argument of maxima. The result is output on a bus where the lower half is the magnitude and the upper half is the argument of maxima.

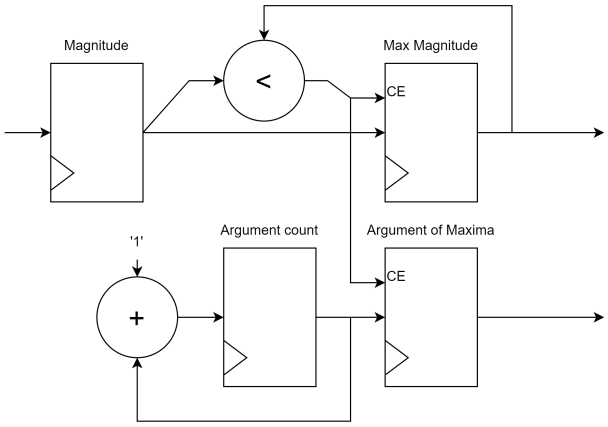


Figure 5.4: The RTL schematic of the argument of maxima module.

The module gives an output value after receiving the AXIS last signal. In a production design, an interrupt should be used to issue when a result is ready. However, for the FPGA implementation, a result is issued for every calculation, and after 1024 PCC iterations the

AXIS last signal is issued. This gives that the module issues one last signal for every 1024 last signals received. This functionality was designed for ease of use of the DMA and due to it being easier to analyze larger data sets. It also has to be noted that the DMA does not work especially well with single entries.

5.4 Partial Cross-Correlation

We may combine all the previously presented hardware modules and create the desired PCC module. The block design of this is shown in Figure 5.5. As observed, the block diagram has all the necessary components, as described in Chapter 3, needed to perform PCC in a single dimension. As a result, if both dimensions are to be calculated simultaneously, two of the design in Figure 5.5 should be put in parallel. Never than less, one may use the design in Figure 5.5 sequentially to calculate both dimensions.

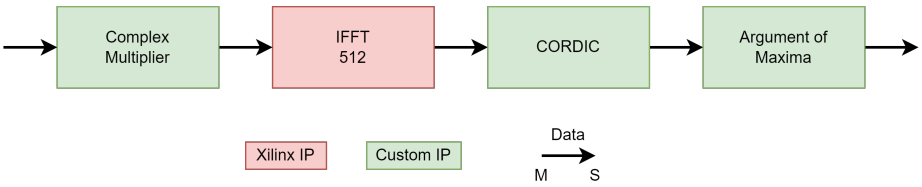


Figure 5.5: Block diagram of PCC.

The inverse FFT module uses the pipelined IO architecture and has scaled input and output. This is because it is desirable to have a pipelined system and, for example, trying to pipeline the radix-2 architecture(Subsection 2.8.4) would require more resources and have much higher latency. The design in Figure 5.5 has a latency of 1698 clock cycles whereof the inverse FFT stage has 1677 clock cycles in latency.

5.5 Novel Two-Dimensional FFT

In Section 3.3 a novel two-dimensional FFT was introduced. The novel two-dimensional FFT has been implemented to make it possible to test how it will perform in hardware. In the hardware implementation, it was chosen that the pipelined FFT architecture was to be used. This will make the design streamlined and the performance will be greater. This is more costly resource-wise, however, the memory bandwidth limits how many FFTs one can perform in parallel. The Zynq Ultrascale+ is able to support 4x128-bit buses from PS to PL(Section 2.6). This means that when using eight bits per pixel, the maximum parallelization is 64. This leads to eight FFT iterations for an image with a size of 512 by 512. With the Zynq Ultrascale+, even the largest image sizes will still have a memory bandwidth limit. With the size of 512 by 512, an image will need 792 of the 1728 DSP slices available. However, even with an image size of 65536 by 65536, the DSP slice count will only increase to 1400. When increasing the image size, the number of FFT iterations will also increase.

The data type chosen in the FFTs is fixed-point unscaled since the precision of eight-bit would be unacceptable[12]. When using unscaled, the bit width will increase with $log_2(Transform\ \ size) + 1$ where the transform size is equal to the image size [35, p.30]. When the input width is eight bits the resulting bit width will be 28 bits at the output of the second FFT stage.

In Figure 5.6 a block diagram of the structure of the novel two-dimensional FFT module is shown. The index and order control manages AXIS signals between the FFT stages. In its current implementation, it stalls the first FFT stage for a minimum of 64 clock cycles while transferring data. It will also keep count of how many rows have been processed and will issue the AXIS last signal at the appropriate time.

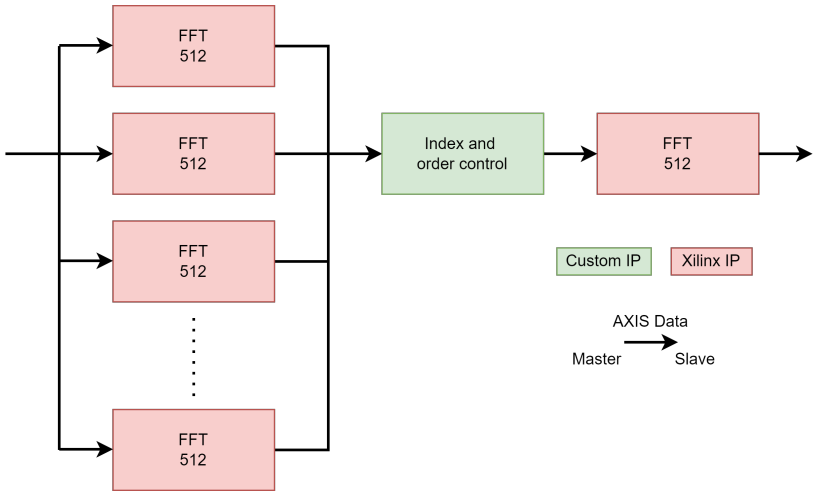


Figure 5.6: Block diagram of novel two-dimensional FFT.

5.6 FPGA Implementation

Testing accuracy and performance of PCC in hardware is one of the key goals of this thesis. Both the novel two-dimensional FFT and partial correlation have their respective FPGA implementation. However, both follow the same structure and only the bus width and clock speeds are different. In Figure 5.7 the structure of both implementations is shown. The figure shows the typical configuration of a hardware accelerator controlled in software. This is where the PS gives instructions to the DMA to stream from a memory address. The DMA will then stream the data in memory through the accelerator. Which in our case is either the novel two-dimensional FFT or PCC.

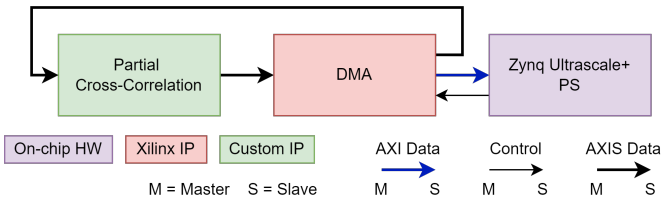


Figure 5.7: Block diagram of hardware for FPGA testing.

When the FPGA implementation is done one may review the resource usage of the implementation(Section 2.6). Table 5.1 shows the resource usage for each implementation. By analyzing the usage we may observe that PCC is very hardware-efficient. However, it has to be noted that this is only a single dimension. This was done to make it easier to implement it on the FPGA and make the testing less prone to faults. This means that resource usage doubles if both dimensions are to be calculated simultaneously. Even when the resource usage doubles for PCC there is still a substantial amount of resources left that may be used to parallelize for better performance or to add more functionality. Unfortunately, one will be limited by the memory bandwidth. For both dimensions to be processed simultaneously, two complex numbers have to be streamed if the spectrum of interest is stored in PL.

Resource	PCC	Novel 2D FFT
Logic Cells	2.6%	43.8%
Flip-Flops	2.1%	38.5%
Distributed RAM	1.8%	26.3%
Block RAM	1.3%	32.1%
DSP Slices	3.0%	45.8%

Table 5.1: PCC and novel two-dimensional(2D) FFT resource usage on Zynq Ultrascale+.

The FPGA implementations have the image size set to a hard 512 by 512. This is because it was deemed not critical for the project to have configurable transform length. Ignoring configurable transform lengths for the FFT also makes implementation easier. If configurable transform lengths were to be implemented one has to set a hard max transform length, i.e. max image size. This is because the pipelined IO architecture for the FFTs only supports a shortening of transform lengths.

5.7 Suggested Production Design

The implementation of a production design, i.e. a complete product, was outside the scope of this project. However, how the production design may be implemented is still interesting. In this way, we may evaluate which components that still need implementation. Consequently, a suggested production design will be presented in this section.

The complete product has three requirements. First, be able to perform PCC. Second, have a two-dimensional FFT in hardware that is fed directly from the camera. Third, the image's frequency spectrums to be correlated should be stored in memory connected directly to PL. The last requirement is not essential, however, does provide an increase in performance and offloads the PS. If desired both PL and PS memory may be used in concert for a performance uplift. The memory bandwidth would still be the bottleneck.

In Figure 5.8 one may observe a block diagram of a possible layout. The PS controls the hardware modules and the PL calculates the correlation. The hardware modules in this design are not equivalent to the previously presented modules. Both the novel FFT and the PCC are meant to have a higher versatility in this design. For example, both should have an interrupt to the PS and the PS should be able to control the workflow, set threshold, and image sizes.

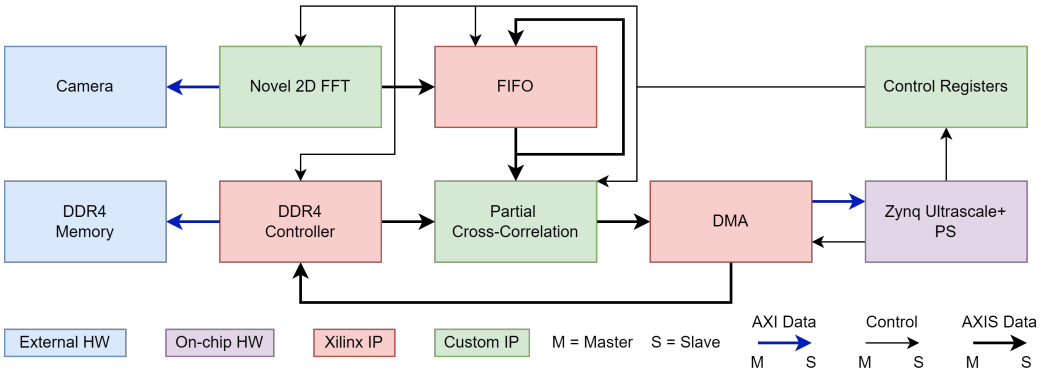


Figure 5.8: Block diagram of suggested production design.

The First In First Out (FIFO) module in Figure 5.8 is supposed to store the captured partial frequency spectrum and then loop through the data as the PCC module does the calculations. By doing this no memory bandwidth will be used to transfer the same data over and over again.

Chapter 6

Hardware Testing

This chapter will test and analyze the results using the hardware designed in the previous chapter. The chapter will follow the same structure as Chapter 4. However, the focus will be on the difference between the software and hardware implementation of PCC instead of the difference between normal and partial cross-correlation. It has to be noted that the sections, except Subsection 6.4.1, uses speckle patterns with symmetry(Section 2.5).

6.1 Simulation

Every developed hardware module was simulated and debugged with the help of the Modelsim simulator. The use of AXIS made the testbenches uniform. The only major difference was bit widths and ordering for the AXIS. The AXIS drivers in the testbenches were supplied by Universal VHDL Verification Methodology (UVVM) [39]. This makes the simulation more robust against a faulty implementation of AXIS. There are also UVVM functions for the evaluation of the output.

Stimuli for the simulations were generated with Python. This made the generation and evaluation of stimuli easy. The stimuli enable a predictable outcome when simulating the hardware modules.

The simulation of edge cases or coverage was not performed in a satisfactory manner. This is due to the lack of a full license for Modelsim. As a result, the simulation is exceptionally slow. Consequently, it was only performed debugging for the functionality and a few edge cases. The lack of edge case testing and coverage is not ideal. However, for an initial hardware implementation, it is of higher value to evaluate the general functionality and performance of the hardware. Due to the slow nature of simulation, hardware was quickly synthesized and implemented on the FPGA to continue functionality testing. This is not good practice, however, necessary if results were to be produced. The bad practice is indeed introducing some instability under testing. The PCC hardware was more thoroughly simulated and there is no perceived instability. The FFT hardware on the other hand crashes every few fifthly thousand iterations.

6.2 Testing with PYNQ

For the testing of the FPGA implementations, PYNQ and Jupyter Notebook were used. Jupyter Notebook is a web interface that may contain executable code, rich text, and plots. The web interface is hosted by the Zynq Ulrascale+ PS. Jupyter Notebook is used for the execution of the PCC using the PYNQ library.

The hardware is uploaded to the FPGA to start testing. This hardware is then mapped as an overlay object. The overlay object contains all the necessary addresses to operate the hardware. For the PCC and FFT designs this means the DMA address as it is the only memory-mapped device in the design.

To use the FPGA hardware, one needs to allocate memory and then stream the data of this memory to PL. Both these operations have provided PYNQ drivers. There is also a functionality to pause the code execution to wait until all data is received. This is not ideal and as mentioned in earlier sections there should be implemented an interrupt scheme.

When streaming to PL the allocated memory must be in the correct format. For the PCC implementation this is 1024 frequency spectrum cross-sections. This is because the allocation and DMA drivers are not designed to receive a single number. The data may be arranged as a matrix where each element to be multiplied shares an index. We will need to stream $1024 \times 512 \times 4$ numbers for PCC execution. Using a $(1024, 512, 4)$ shape we are able to stream two complex numbers at a time. The imaginary and real parts of the complex numbers are stored separately giving four elements for two complex numbers. The allocation for the novel two-dimensional FFT is somewhat more complex due to the parallelization implemented and the two-dimensional nature of an image. The allocated space for a single novel two-dimensional FFT is $512 \times 8 \times 64$ 8-bit elements with a shape of $(512 * 8, 64)$. The streaming width of 512 bits requires this shape. To make sure the stream order is correct, the image must be split and stacked in this form. The DMA will read all elements in a column before going to the next row. This will jumble the data if no precaution is taken. Figure 6.1 shows the matrices before and after the split and stack operation. The numbers indicate the streaming order. When parallelizing an FFT one needs to make sure either the row or column needs to stay constant until completed. In Figure 6.1 we may observe that we are parallelizing the rows. This means that the row needs to stay constant, however, it does not do this until after the split and stack operation.

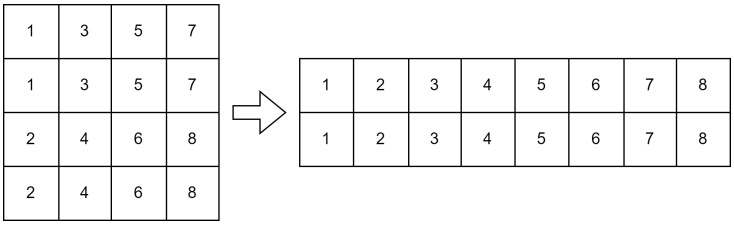


Figure 6.1: Vertical split and horizontal stack needed for ordering.

6.3 Performance and Accuracy

To measure the performance of PCC in hardware, the same test was used as in Section 4.1. However, in this test, we are not looping since a single use contains 1024 instances of PCC. Thus the time measured when performing a single PCC call was divided by 1024 to calculate the average. The time used to perform 1024 correlations was 1.976 ms. This gives the per PCC operation time of 1.93 µs. The times measured include the initial latency of 5.3 µs. However, even with this latency and the overhead from using Python we are still close to the theoretical maximum speed of 1.6 µs per PCC when the FPGA is operating at 322 MHz.

In Table 6.1 we may observe the speedup of every previously measured software implementation to that of the FPGA. The speedup by using FPGA over the partial software implementation is quite good. The worst speedup is for the FPGA implementation against the AMD system. However, this is still very good, considering that the AMD system may consume up to over a hundred times more power. The best speedup is against the Arm system located on the same SoC as the FPGA. This speedup is substantial and the FPGA is using half the power compared to the PS. The power metric was included in the implementation report. When comparing the FPGA implementation to the normal approach we may see the most gain. The speedups are very good. By using the FPGA and PCC we may have a performance increase of 60 000, more than four orders of magnitude. Without changing hardware. The comparison between the normal approach and PCC on the FPGA also shows that most of the gain is caused by the use of PCC.

Hardware	Software Method	Software Time	FPGA Time	Speedup
AMD Ryzen 3900 XT	Partial	26.7 µs	1.93 µs	13.8
Intel i5-8250U	Partial	50.3 µs	1.93 µs	26.1
ARM Cortex-A53	Partial	367.5 µs	1.93 µs	190.4
AMD Ryzen 3900 XT	Naive	11.0 ms	1.93 µs	5699.5
Intel i5-8250U	Naive	22.8 ms	1.93 µs	11813.5
ARM Cortex-A53	Naive	114.4 ms	1.93 µs	59274.6

Table 6.1: Software (SW) V. FPGA at 322 MHz. Average speedup and time per iteration.

The performance of the FPGA implementation is better than the software performance. The same may not be said about the accuracy. As shown in Table 6.3 the accuracy is lower than the accuracy in Section 4.1. The reason for fewer noise levels is due to the instability of the novel two-dimensional FFT(Section 6.1) which makes testing time-consuming. It has to be noted that the reason for this is the novel two-dimensional FFT. This is further analyzed next in Section 6.4.

Noise	SNR	Faults	Accuracy
0 %	NA	0	100 %
10 %	19 dB	8745	96.66 %

Table 6.2: Accuracy with and without white Gaussian noise.

We may observe in Figure 6.2 that this faulty accuracy occurs primarily in the corners. The figure shows the offset data as intensities of one dimension. Performing the PCC in only one dimension is again due to the instability and time needed to perform novel two-dimensional FFTs(Section 6.1). Like the faulty offset in Section 4.1, the faulty offsets are at the corners. However, there are more in two of the corners. The intensities in the correlation map in Section 6.5 show a clear difference in corner intensities. The lower-intensity corners are where most of the faulty offsets are located. Why this is the case is not known. However, it may be due to the fact that the PCC is performed in only one dimension. Further faulty offsets may occur at the other corners if we include the other dimension as well.

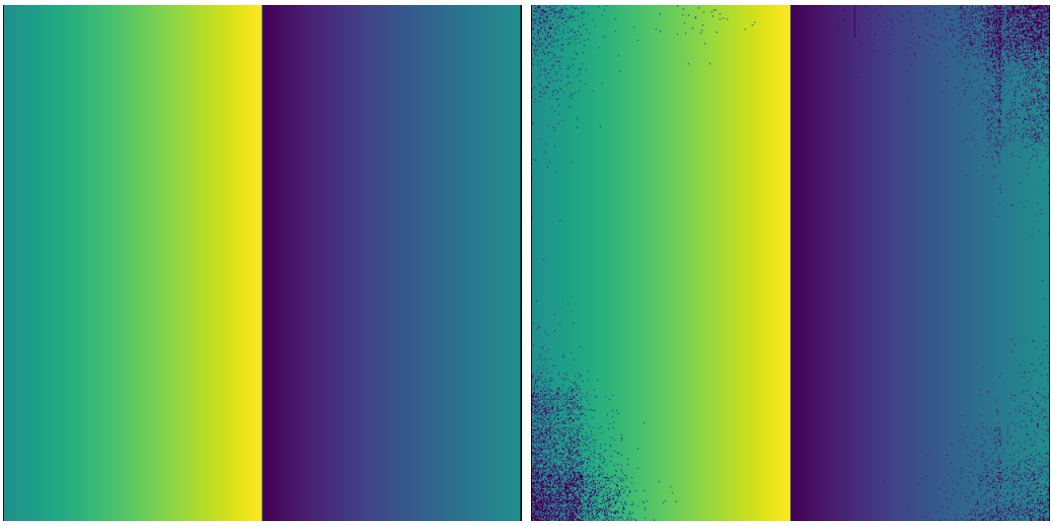


Figure 6.2: PCC offset data without(Left) and with(Right) noise.

6.4 Differences in Hardware and Software

In Chapter 4 we observed and analyzed the difference between the normal approach and PCC. In this section, we will look at the difference between software and hardware. First, we will look at PCC and novel two-dimensional FFT separately. Lastly, we will study the result of the two combined. The test we will be using is the same as the one used in Section 4.2 where we performed PCC on every diagonal offset. The frequency spectrums used in Subsection 6.4.1 are calculated in software.

6.4.1 Partial Cross-Correlation

The difference in the results between software and hardware PCC is primarily scale. This is due to the Xilinx FFT scaling during computation so that the bit width will not increase(Subsection 2.8.4). This gives the FPGA implementation a lower magnitude, even when the CORDIC gain is not accounted for (Subsection 2.8.3). To observe the difference we first multiply the hardware result by the average difference between software and hardware. Then we subtract that product from the software result. This operation may be observed in Equation 6.1. If we use this equation on a PCC result we will get the difference plot shown in Figure 6.3. A maximum difference of 100 is acceptable considering the correlation strength magnitude is in the billions. The maximum deviation has an effect of -142 dB. This is in the range one would expect since the Xilinx FFT documentation shows an -120 dB effect when using 24 bits [27]. The small effect makes the use of PCC in hardware as reliable and as accurate in software. However, it has to be noted that the complex multiplier input does not truncate in its current configuration. Thus the difference between hardware and software may change if the truncation is changed.

$$SW_i - \left(\frac{\sum_{n=0}^{N-1} SW(n) - \sum_{n=0}^{N-1} HW(n)}{N} \right) * HW_i = D_i \tag{6.1}$$

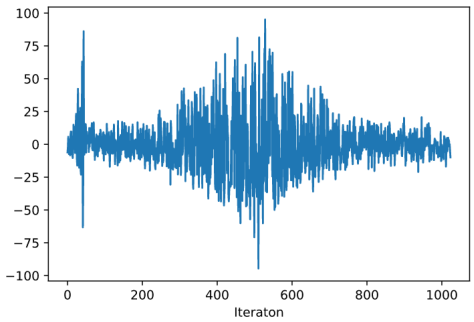


Figure 6.3: The difference in hardware and software PCC.

6.4.2 Two-Dimensional FFT

In the accuracy presentation in Section 6.3, it was mentioned that the accuracy reduction was due to the novel two-dimensional FFT. If we look at the frequency spectrums in Figure 6.4 we may indeed observe a large difference. The reduction in magnitude is expected due to the FFT architecture even though there is no scaling [35, p.30]. However, the difference in shape is not expected. When reviewing this result it was assumed that something was wrong. Performing a single PCC of the frequency spectrums dispelled this notion. The PCC was done in software and the results may be seen in Figure 6.5. The peaks indicate

the offset. The PCC of the two different frequency spectrums returns the correct offset. The hardware-calculated frequency spectrum even delivers a lower noise floor than what the software-calculated does. These are good results. However, if noise is applied, the noise floor calculated from the hardware frequency spectrum will quickly surpass the noise floor given by the software-calculated frequency spectrum. This means that the degradation in accuracy shown in Section 6.3 is due to the novel two-dimensional FFT performed in hardware. The reason for this is not known.

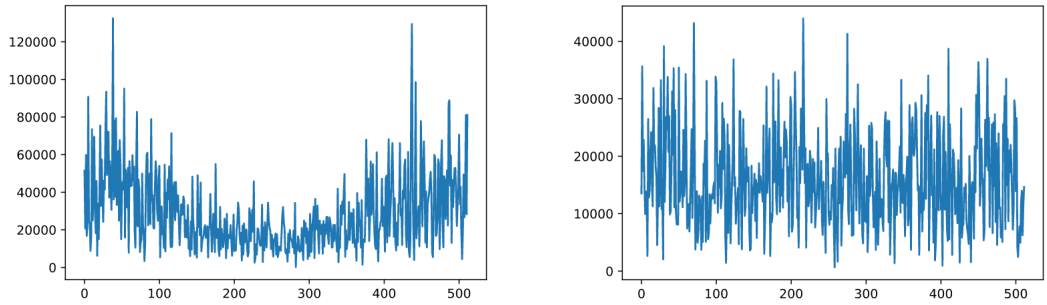


Figure 6.4: Cross-section of the frequency spectrum. SW(Left) and Hardware (HW)(Right).

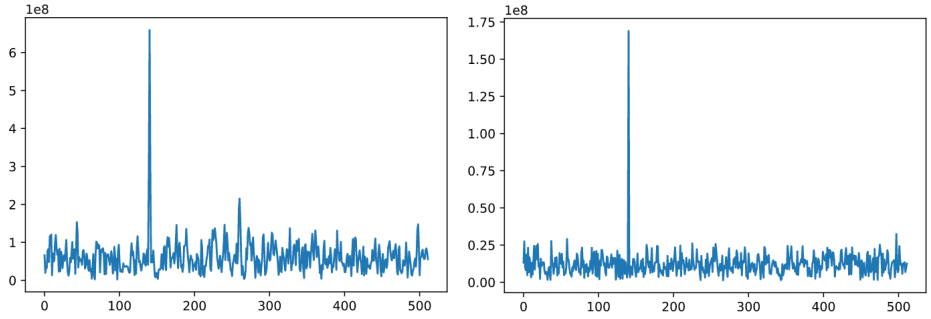


Figure 6.5: PCC results using SW(Left) and HW(Right) frequency spectrums.

6.4.3 Hardware FFT and PCC Combined

The previous subsections showed that the reason for poor accuracy when performing PCC in hardware was due to the novel two-dimensional FFT that was also performed in hardware. In this subsection, we will observe how PCC and the novel two-dimensional FFT perform together and observe the impact of noise. In Figure 6.6 the cross-section of the diagonal sweep and the offset are shown. The shape and dynamic range, the ratio from the highest to the lowest value, of the of PCC and novel two-dimensional FFT are what to expect. The difference when scaling is accounted for between the software and hardware results is much larger. The difference is in the thousands instead of the hundreds. The offsets calculated in hardware are, however, correct and have only minor deviations in the ambiguous area. Note that it is only in this figure it is possible to observe the symmetry of the speckle pattern. I.e. the correlation strength is mirrored.

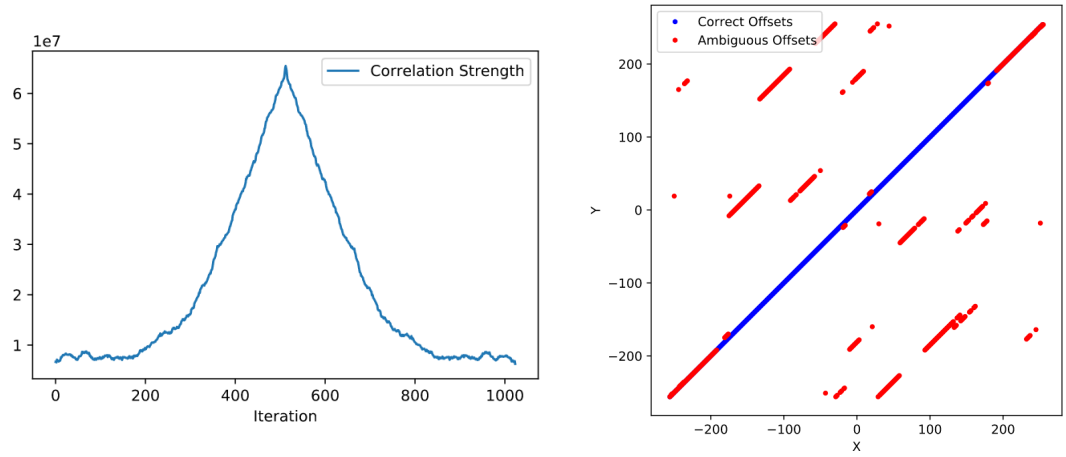


Figure 6.6: The offsets and correlation strength. PCC and FFT in hardware without noise.

In Figure 6.6 the effect of using novel two-dimensional FFT in hardware was not noticeable. This is not the case when white Gaussian noise is added. In Figure 6.7 there is a clear degradation in the shape and the dynamic range. As may be observed, the shape is no longer symmetrical. However, it is the dynamic range that suffers the most when noise is applied. The dynamic range is approximately halved. This is a change that does not happen when adding noise when calculating in software. It is the novel two-dimensional FFT that is the cause of this. One may argue that the correct offsets have a correlation strength that makes it possible to remove all ambiguous results, as it is suggested in Subsection 4.3.1. However, this falls short since the dynamic range changes depending on noise which in turn changes the threshold. This means that configuring the threshold is a much more strenuous task when using novel two-dimensional FFT in hardware.

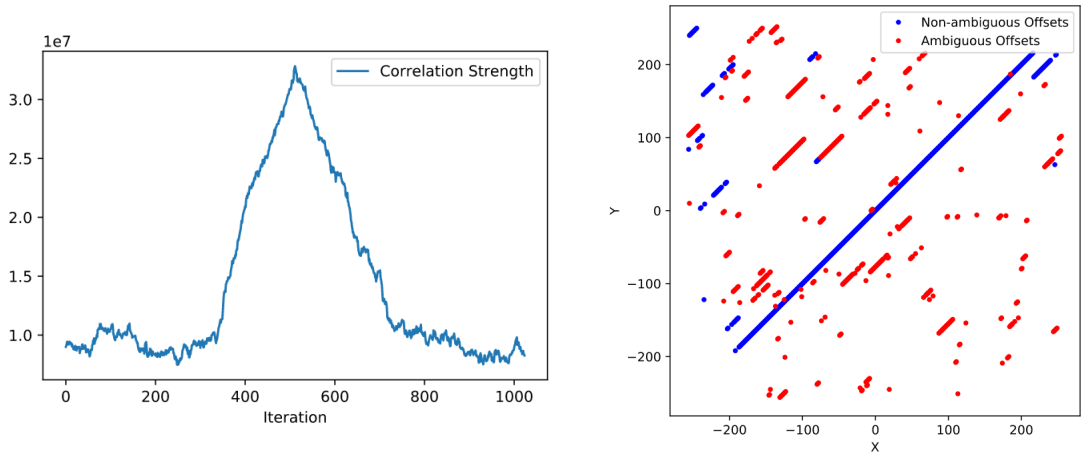


Figure 6.7: The offsets and correlation strength. PCC and FFT in HW with noise($\sigma = 10\%$).

6.5 Correlation Map

In Figure 6.8 the correlation strength of every offset calculated in both hardware and software is shown. There is no noise added and as mentioned earlier, only one dimension was calculated due to the instability of the novel two-dimensional FFT. When noise is not present there is a clear improvement in dynamic range, i.e. the background is darker. However, the shape of the hardware result is worse as the shape is not as circular. As shown in Section 6.3, this dip in intensity matches the area where incorrect offsets occur when noise is applied. The reason for this is not known, however, PCC with novel two-dimensional FFT in hardware should be performed for both dimensions to observe if the intensity reduction persists.

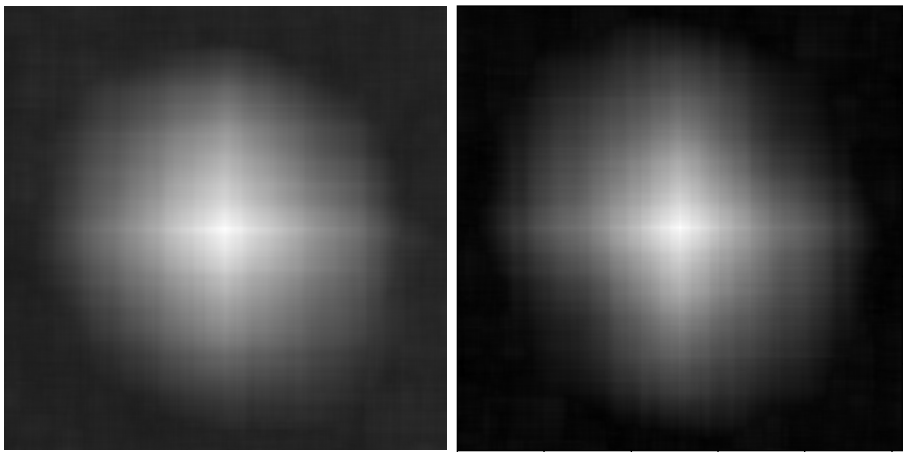


Figure 6.8: Correlation intensity maps. SW(Left) and HW(Right) PCC.

Chapter 7

Discussion and Future Work

This thesis has shown the reasoning behind and implementation of PCC. It was, however, not a complete implementation. There are also important elements ignored to limit the scope. This chapter will mention and discuss some of these elements and parts of the implementation that needs improving.

7.1 Partial Cross-Correlation

PCC should be used if there is a large number of speckle images to be correlated. For a few images, it is not necessary to implement PCC. It will be slower and modern computers make the implementation, understanding, and drawbacks of PCC not worth the developer's time. PCC is not suited for most images. Therefore, work should be performed on PCC making it viable for other image types as well. If PCC is made to work on normal images it would greatly increase the viability of PCC.

When developing and testing PCC some aspects have been ignored. This is to limit the scope of the thesis, however, it also means that important elements are not implemented. These elements have to be accounted for when PCC is implemented.

7.1.1 Threshold Scheme and Padding

Threshold and padding have been ignored during development. This is due to the multiple approaches to solving these problems. It depends on the necessary requirements for each use case. However, in general use for speckle images with PCC, it may be advantageous to ignore padding altogether. This may be done by setting the threshold high enough so that ambiguous results are ignored. An example of this scheme is shown in Subsection 4.3.1. The lack of padding does, however, mean that there is a need for overlapping images. This is not a usual approach since it increases the number of correlations. The increase does not however have a large impact when using PCC. The extra computation needed to perform a padded FFT on the captured image is much more than what is needed to perform twice as many correlations with PCC.

7.1.2 Normalizaton

When CC is to be performed, normalization usually has to be implemented. However, normalization has been ignored in this thesis. This is because the speckle images used do not require normalization. However, it is still beneficial to have normalization for stable threshold evaluation. If normalization were used, the dynamic range reduction in Subsection 6.4.3 would not be as large a problem as it is without normalization.

The type of normalization best suited for PCC has to be evaluated. However, a phase correlation type of normalization may be best suited for PCC. This is because of the relative ease of implementation and the fact that it is performed in the frequency spectrum. NCC is performed in the spatial domain for every pixel. How the PCC results affect this is not known. However, it may be faster to perform NCC if different speckle patterns have the same standard deviation. If true, NCC will be faster than phase correlation due to not needing to calculate the standard deviation for every image.

7.1.3 PCC with Normal Images

PCC was implemented for and works well with speckle images. However, PCC may be applied to other images as well has not been evaluated in great detail. In Subsection 4.3.2, correlation intensity maps of normal images made with the use of PCC were shown. These correlation maps show that there is a potential for further use. It is clear that, in its current implementation, PCC will not be able to calculate the offset reliably. However, it may still potentially be used to evaluate if two images are entirely equal. This functionality is not needed if there are few images to be evaluated. Nevertheless, the potential speedup, if millions of pictures are to be evaluated, is massive. This would also mean that much less storage space is needed to store the images. One million 1920 by 1080 images would require over 2 terabytes of storage. This is if the images are in the spatial domain. If all the images are stored in the frequency domain for faster calculation, it would require over 16 terabytes of storage. The storage required for the frequency spectrums used by PCC for the same amount of images is 24 gigabytes. The 24 gigabytes needed for PCC gives an improvement of over 666 if the images are to be stored in the frequency spectrum. If the images are stored in the spatial domain, the improvement is over 83.

7.2 Hardware Implementation

The hardware implementation of PCC in a relevant setting is in need of further improvements. In this thesis, only the individual components novel two-dimensional FFT and PCC were tested. This means that an implementation of a system is missing. The components that have been implemented have some problems as well. Especially the novel two-dimensional FFT has stability and noise issues. There were not any perceived issues with the PCC component, however, both components need further testing and validation in order to be used in a system.

7.2.1 Production System

Unfortunately, a system that may be deployed for use was not implemented. This production system will need to improve the two implemented components, add memory and camera interfaces, and enable ease of use of the system. An example of how this production system may look was presented in Section 5.7. There is also the need for software drivers. These drivers should support ease of use, however, also enable detailed control of the hardware system.

The novel two-dimensional FFT and PCC components have been implemented, however, are in need of further functionality. The novel two-dimensional FFT would benefit from a configurable resolution factor and transform length. The PCC should have a configurable transfer length as well. A configurable threshold will also be of use in the PCC. Both the components should have interrupt functionality with the PS implemented.

7.2.2 Noise

The hardware implementation of the novel two-dimensional FFT has been shown to be more sensitive for noise compared to the software implementation. This is not ideal. The reduction in dynamic range, shown in Subsection 6.4.3, is especially an unwanted behavior. Research into why this is the case needs to be carried out. However, it is likely due to the use of the unscaled fixed-point data type in the Xilinx FFT IP. It may be possible that noise will propagate more when this data type is used. The fact that the dynamic range is better in hardware compared to software, when no noise is present, supports this hypothesis.

7.2.3 Performance and Power Usage

The performance of PCC in hardware has been shown to be very good when compared to normal CC in software. The performance gain, compared to when using PCC in software, is not as large. However, there is a large gain in power efficiency. The best performing PCC in software may use over a hundred times more power than what the hardware implementation uses. The least gain in efficiency for the hardware implementation is compared to the slowest performing software system, however, then the performance gain is much larger. Nevertheless, even with hardware, the largest gain in overall efficiency is the use PCC and not normal CC.

Chapter 8

Conclusion

This thesis has conducted research into the optimization of cross-correlating a large number of speckle images. The research has yielded the solution of Partial Cross-Correlation (PCC). PCC performs the correlation computation on a single row and column of the frequency spectrum and does still give an exact offset. The suggested solution of PCC has a significant performance uplift, compared to the normal approach. There is also a reduction in data needed to store each image in the frequency spectrum. PCC is designed for speckle images and does not currently support any other use case. However, it has been shown that there also can be a potential for other use cases. The complexity added by the use of PCC is relatively small and it is as easy to implement as normal CC.

For a hardware implementation, PCC is cost-effective and has a relatively easy pipelined design. The limitation in hardware performance is the memory bandwidth and not the PCC implementation. The difference in results between a PCC implementation in hardware and software is shown to be insignificant. The performance gains of using physical hardware are not as much as the performance gain by use of PCC instead of normal CC, however, it is still significant.

The inherent data reduction when using PCC has allowed for a memory-friendly implementation of a two-dimensional FFT. This novel two-dimensional FFT eliminates the use of intermediate storage at the expense of having to input the image twice. The current novel two-dimensional FFT hardware implementation has been shown to be more sensitive to noise than the software implementation.

A PCC implementation for use with speckle images has a great performance advantage over normal CC in the frequency domain and should be used in future systems.

Bibliography

[1] J. Goodman, "Statistical properties of laser speckle patterns," *Laser Speckle and Related Phenomena*, vol. 9, p. 57, Dec. 1, 1963, ISSN: 978-3-540-13169-4.

[2] R. Sass and A. G. Schmidt, in *Embedded Systems Design with Platform FPGAs: Principles and Practices*, Morgan Kaufmann, Sep. 10, 2010, pp. 146–147, ISBN: 978-0-08-092178-5.

[3] M. Mutchler, S. Beckwith, H. Bond, *et al.*, "Hubble space telescope multi-color ACS mosaic of m51, the whirlpool galaxy," *Bulletin of the American Astronomical Society*, vol. 37, no. 2, Jan. 1, 2005.

[4] K. E. Whitaker, M. Ashas, G. Illingworth, *et al.*, "The hubble legacy field GOODS-s photometric catalog," *The Astrophysical Journal Supplement Series*, vol. 244, no. 1, p. 16, Sep. 2019, Publisher: The American Astronomical Society, ISSN: 0067-0049.

[5] *ZynqUltrascale+ MPSoC ZCU104 user manual*, 2018. [Online]. Available: `https://docs.xilinx.com/v/u/en-US/ug1267-zcu104-eval-bd` (visited on 09/09/2022).

[6] R. Gonzalez and R. Woods, *Digital Image Processing*, 3rd ed. Pearson Education, 2010.

[7] G. G. Lovas, in *Statistikk for universiteter og hogskoler*, 4th ed., Universitetsforlaget, 2018, pp. 290–293, ISBN: 978-82-15-03104-0.

[8] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing*, 4th ed. Pearson Prentice Hall, 2007, 1084 pp.

[9] K. Aznag, E. O. Ahmed, and E. El Bachari, "Main problems and proposed solutions for improving template matching," *JOIV: International Journal on Informatics Visualization*, vol. 3, p. 2019, Mar. 12, 2019.

[10] J. Lewis, "Fast normalized cross-correlation," *Ind. Light Magic*, vol. 10, 2001.

[11] J. W. Goodman, *Introduction to Fourier Optics*. Roberts and Company Publishers, 2005, 520 pp., Google-Books-ID: ow5xs_Rtt9AC, ISBN: 978-0-9747077-2-3.

[12] E. Fykse. "Performance comparison of GPU, DSP and FPGA implementations of image processing and computer vision algorithms in embedded systems." (2013), [Online]. Available: `https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2370873` (visited on 12/09/2022).

[13] S. Mattoccia, F. Tombari, and L. Di Stefano, "Reliable rejection of mismatching candidates for efficient ZNCC template matching," in *2008 15th IEEE International Conference on Image Processing*, ISSN: 2381-8549, Oct. 2008, pp. 849–852.

[14] L. Di Stefano, S. Mattoccia, and F. Tombari, "ZNCC-based template matching using bounded partial correlation," *Pattern Recognition Letters*, vol. 26, no. 14, pp. 2129–2134, Oct. 15, 2005, ISSN: 0167-8655.

[15] "Normalized cross-correlation - SPC wiki." (2023), [Online]. Available: https://web.psi.edu/spc_wiki/Normalized%20Cross-Correlation (visited on 04/17/2023).

[16] H. Foroosh, J. Zerubia, and M. Berthod, "Extension of phase correlation to subpixel registration," *IEEE Transactions on Image Processing*, vol. 11, no. 3, pp. 188–200, Mar. 2002, Conference Name: IEEE Transactions on Image Processing, ISSN: 1941-0042.

[17] B. Reddy and B. Chatterji, "An FFT-based technique for translation, rotation, and scale-invariant image registration," *IEEE Transactions on Image Processing*, vol. 5, no. 8, pp. 1266–1271, Aug. 1996, Conference Name: IEEE Transactions on Image Processing, ISSN: 1941-0042.

[18] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965, ISSN: 0025-5718, 1088-6842.

[19] S. W. Smith, in *The Scientist & Engineer's Guide to Digital Signal Processing*, 2nd ed., 1999, pp. 410–416.

[20] J. Vangindertael, R. Camacho, W. Sempels, H. Mizuno, P. Dedecker, and K. Janssen, "An introduction to optical super-resolution microscopy for the adventurous biologist," *Methods and Applications in Fluorescence*, vol. 6, Feb. 9, 2018.

[21] B. van den Bogaert, "Chapter 1 - finding frequencies in signals: The fourier transform," in *Data Handling in Science and Technology*, ser. Wavelets in Chemistry, B. Walczak, Ed., vol. 22, Elsevier, Jan. 1, 2000, pp. 3–31.

[22] H. Ghandoor, G. Youssef, M. El-Aasser, and A. Tarek, "Investigation of laser speckle patterns of solar cells," *International Journal of Advanced Research*, vol. 7, no. 8, pp. 61–70, 2019.

[23] *Personal communication about speckle image properties*, in collab. with G. Slettemoen, E-mail, 2023.

[24] D. Duncan, S. Kirkpatrick, and R. Wang, "Statistics of local speckle contrast," *Journal of the Optical Society of America. A, Optics, image science, and vision*, vol. 25, pp. 09–15, 2008.

[25] "Zynq UltraScale+ MPSoC ZCU104 evaluation kit," Xilinx. (2023), [Online]. Available: https://www.xilinx.com/products/boards-and-kits/zcu104.html (visited on 04/19/2023).

[26] "AMBA AXI protocol specification." (), [Online]. Available: https://developer.arm.com/documentation/ihi0022/latest/ (visited on 06/20/2023).

[27] "Xilinx FFT IP resource usage exsamples." (2022), [Online]. Available: https://www.xilinx.com/htmldocs/ip_docs/pru_files/xfft.html (visited on 12/08/2022).

[28] "PYNQ introduction — python productivity for zynq (pynq)." (2023), [Online]. Available: https://pynq.readthedocs.io/en/latest/ (visited on 04/19/2023).

[29] *Xilinx/PYNQ*, original-date: 2016-01-20T01:16:27Z, Apr. 19, 2023. [Online]. Available: https://github.com/Xilinx/PYNQ/blob/de6b6fc3a803945d59f8f06523addfe0d9b60a1c/pynq/overlay.py (visited on 04/19/2023).

[30] H. Gao, H. Dai, and Y. Zeng, "High-speed image processing and data transmission based on vivado HLS and AXI4-stream interface," in *2018 IEEE International Conference on Information and Automation (ICIA)*, Aug. 2018, pp. 575–579.

[31] "DSP48e2 • vivado design suite reference guide: Model-based DSP design using system generator (UG958) • reader • AMD adaptive computing documentation portal." (), [Online]. Available: https://docs.xilinx.com/r/en-US/ug958-vivado-sysgen-ref/DSP48E2 (visited on 04/19/2023).

[32] J. Volder, "The CORDIC trigonometric computing technique," *IRE Transactions on Electronic Computers*, vol. 8, no. 3, pp. 330–334, 1959.

[33] J. S. Walther, "A unified algorithm for elementary functions," in *Proceedings of the May 18-20, 1971, spring joint computer conference*, ser. AFIPS '71 (Spring), New York, NY, USA: Association for Computing Machinery, May 18, 1971, pp. 379–385, ISBN: 978-1-4503-7907-6.

[34] R. A. Adams and C. Essex, in *Calculus: a complete course*, 9th ed., OCLC: 1039886914, Toronto: Pearson Canada Inc., 2017, p. 222, ISBN: 978-0-13-415436-7.

[35] *Xilinx FFT LogiCORE v9*, 2023. [Online]. Available: https://xilinx.eetrend.com/files-eetrend-xilinx/download/201606/10227-24631-pg109-xfft.pdf (visited on 06/15/2023).

[36] R. N. Clark. "Clarkvision: Digital camera review and sensor performance summary." (2023), [Online]. Available: https://clarkvision.com/articles/digital.sensor.performance.summary/index.html (visited on 06/13/2023).

[37] R. G. Lyons, *Understanding Digital Signal Processing*, 1st ed. USA: Addison-Wesley Longman Publishing Co., Inc., 1996, 544 pp., ISBN: 978-0-201-63467-9.

[38] M. Lakshmanan, A. Bharathi, and J. A.N., "REVIEW ON FAST COMPLEX MULTIPLICATION ALGORITHMS AND IMPLEMENTATION," vol. 6, pp. 2393–8374, May 15, 2019.

[39] "Uvvm / UVVM · GitLab," GitLab. (Mar. 21, 2023), [Online]. Available: https://opensource.ieee.org/uvvm/uvvm (visited on 05/31/2023).

Appendix A

Additonal Results

A.1 Calculated and Manual Contrast

In Chapter 4 it was stated that using calculated contrast for testing was safe. To a degree, this appendix proves this claim. Figure A.1 shows the difference between calculated and manually adjusted contrast. The test is the same as in Section 4.2 and both use a resolution factor of 96. As may be observed, the manually adjusted contrast result is not as smooth as the calculated and the magnitude is different. However, the dynamic range is the same. If we tweak the resolution factor we may achieve a smoother result.

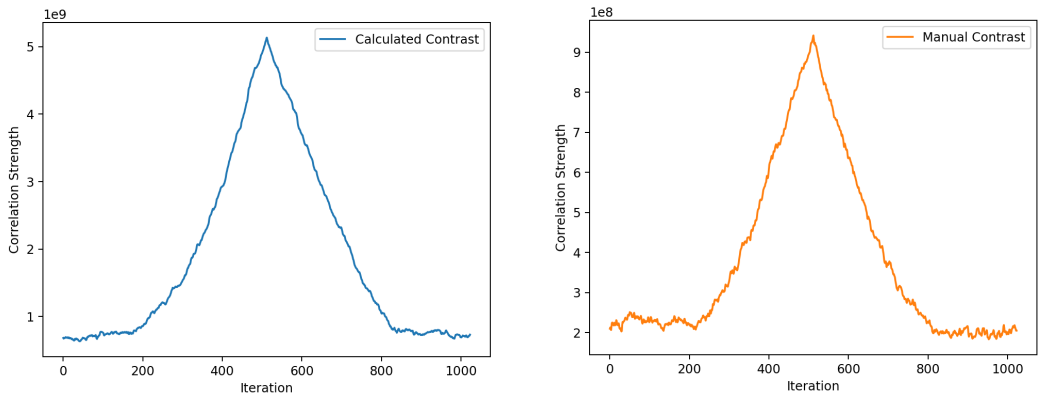


Figure A.1: Correlation strength using calculated(Left) and manual(Right) contrast.

Figure A.2 shows the difference between calculated and manual contrast offsets when the noise level is at 30%. Manually adjusted contrast is weaker to noise, however, this is expected since the SNR is lower. Figure A.2 is using 16 as the resolution factor.

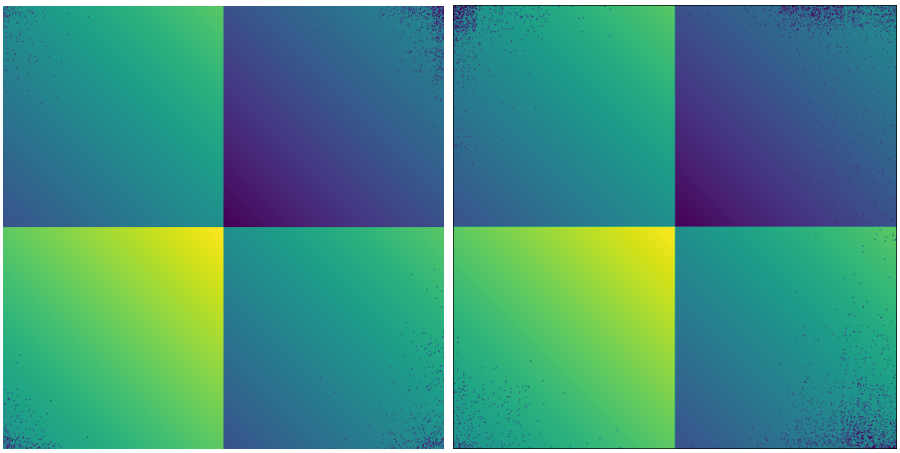


Figure A.2: Calculated(Left) and manual(Right) contrast raw offset data noise($\sigma = 30\%$).

The manual contrast has less accuracy as is shown in Table A.1. If we compare this table to Table 4.2 in Section 4.1 we may observe that the most drastic difference is when the noise level is 10 and 20 percent. However, this reduction in noise is not enough to make PCC not viable. For testing, it was decided to use calculated contrast to make the results consistent.

Noise	SNR	Faults	Accuracy
0 %	NA	0	100 %
5 %	15 dB	0	100 %
10 %	9 dB	22	99.99 %
20 %	3 dB	496	99.81 %
30 %	-1 dB	3070	98.83 %

Table A.1: Accuracy with different levels of white Gaussian noise.

From Table 4.2 and A.1 we may guess that PCC will fail in the given area when the SNR is between 13 and 15 dB. However, we may also observe that manually adjusted contrast performs better at 9 dB SNR. The reason for this is not known. Why there is not a catastrophic failure when the SNR is negative is also an interesting question. The reason for this may be because of the type of noise used.

A.2 Different Sized Images

CC is often performed with different-sized images. This is outside the scope of this thesis, however, it is still interesting to evaluate if PCC may be used. In Figure A.3 PCC offset data between 512 by 512 and 128 by 128 speckle images is shown. From this, we may observe that PCC does indeed work with images of different sizes. Note that the smaller image must have the same center after padding is applied. Padding is needed to make the rows and columns of equal length. The centering need is for reference. The resulting offset will be skewed otherwise.

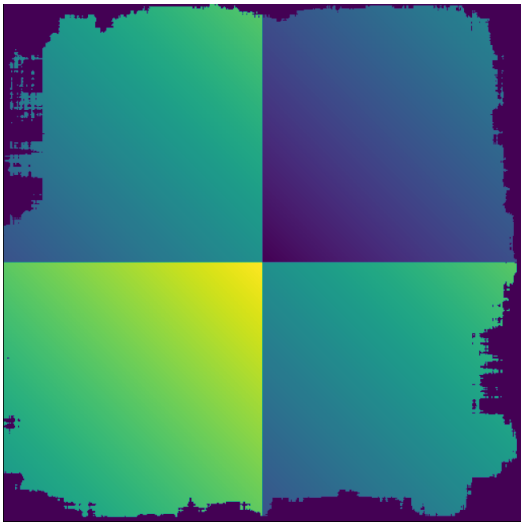


Figure A.3: The PCC offset data when using 512 by 512 and 128 by 128 images.

Appendix B

Source Code

B.1 Python Code

Python code that has functions for normal CC, PCC, and speckle image generation.

```
from PIL import Image, ImageOps
from math import pi
import numpy as np

def speckle_image(image_size, speckle_size):
    """
    This code is an adaptation of Gudmun Slettemoens Matlab script. The method is described in:
    http://pdxscholar.library.pdx.edu/cgi/viewcontent.cgi?article=1119&context=ece_fac
    """
    L = image_size                      #Image length
    D = L / specle_size                 #Diameter
    R = int(D/2)                        #Radius
    pad = int(L/2)                      #Padding
    speckle_img = np.zeros((L, L))

    # Generating random exponential noise in a circle in the image.
    # The speckle size is decided by the diameter. The smallest speckles give D = L/2.
    for i in range(pad - R ,pad + R):
        for j in range(pad - R, pad + R):
            if np.abs((pad-i)*(pad-i) + (pad-j)*(pad-j)) < D*D/4:
                speckle_img[i, j] = np.exp(np.random.uniform(-pi, pi))

    #Performing an FFT and removing imaginary numbers. This results in a speckle pattern.
    speckle_img = np.fft.fft2(speckle_img)
    speckle_img = np.multiply(speckle_img, np.conjugate(speckle_img))

    #PIL Image does not like complex values(Even if i = zero).
    speckle_img = np.abs(speckle_img)

    #The dynamic range is too large. Need to change the contrast.
    mean = np.sum(speckleImg)/(L**2)
    speckleImg = speckleImg * (1/sqrt(mean))

    img_out = Image.fromarray(speckle_img)
    img_out.save('speckle.webp', lossless = True)

```

```
37  def xcorr(img1, img2):
38      """
39      Cross-correlation function. Accepts two images.
40      Returns correlation strength and offset.
41      """
42
43      #Transforms the images to the frequency domain.
44      img1_f = np.fft.fft2(img1)
45      img2_f = np.fft.fft2(img2)
46
47      #Correlation multiplication
48      R = np.multiply(img1_f, np.conj(img2_f))
49
50      #Transforms back to the spatial domain.
51      r = np.fft.ifft2(R)
52
53      #Get offset and correlation strength
54      x, y = np.unravel_index(np.argmax(r), r.shape)
55      cc_strength = np.max(np.abs(r))
56
57      #Change reference point to get offset vector
58      if x > r.shape[0]/2 - 1:
59        x = x - r.shape[0]
60      if y > r.shape[1]/2 - 1:
61        y = y - r.shape[1]
62
63      return x, y, cc_strength
64
65
66  def partial_xcorr(img1, img2, res_fac=96):
67      """
68      Partial cross-correlation function. Accepts two images and resolution factor.
69      Returns correlation strength and offset.
70      """
71
72      #Transforms the images to the frequency domain.
73      img1_f = np.fft.fft2(img1)
74      img2_f = np.fft.fft2(img2)
75
76      #Correlation multiplication
77      Rx = np.multiply(img1_f[:,res_fac], np.conj(img2_f[:,res_fac]))
78      Ry = np.multiply(img1_f[res_fac,:], np.conj(img2_f[res_fac,:]))
79
80      #Transforms back to the spatial domain.
81      rx = np.fft.ifft(R)
82      ry = np.fft.ifft(R)
83
84      #Get offset and correlation strength
85      x = np.argmax(rx)
86      y = np.argmax(ry)
87      cc_strength = np.max(np.abs(rx) + np.abs(ry))
88
89      #Change reference point to get offset vector
90      if x > rx.shape[0]/2 - 1:
91        x = x - rx.shape[0]
92      if y > ry.shape[0]/2 - 1:
93        y = y - ry.shape[0]
94
95      return x, y, cc_strength
96
```

B.2 Matlab Code

Matlab code courtesy of Gudmund Slettemoen.

```
function [speckleImage] =...
  createSpeckleImage(dimImage, minimumSpeckleSize, blurStDev)

%% Ref: Katrin Philipp?? Technische Universit\"at Dresden
% A common way of generating fully developed speckle patterns
% is by a fast Fourier transform of a phase matrix, e.g. as described in
% reference[1]. But to me (Gudmund) it seems that what they call the
% average speckle size is 2 times the minimum speckle size.
% reference[1]:
% http://pdxscholar.library.pdx.edu/cgi/viewcontent.cgi?article=
% 1119&context=ece_fac
% To simulate optical resolution, I have added a gaussian blur
% with a standard deviation of "blurStDev".
% The image has been normalized with 3 times the average of the 25%
% "speckleImage" central part.

% Create speckle image according to ref [1].
L=dimImage;
D=L/minimumSpeckleSize;
speckleImage = zeros(L, L);
pad = L/2; % paddin
for k=pad-D:D+pad
  for l=pad-D:D+pad
    if abs((pad-k)^2+(pad-l)^2) < D^2/4
      speckleImage(k,l) = exp(unifrnd(-pi,pi));
    end
  end
end
mfft = fft2(speckleImage);
speckleImage = mfft.*conj(mfft);

% To simulate the resolving power of the camera pixels we blur
% the speckleImage with a gaussian blur with a standard deviation
% of "blurStDev".
speckleImage=imgaussfilt(speckleImage,blurStDev);
imageMean=mean(mean(speckleImage(L/2-L/8:L/2+L/8,L/2-L/8:L/2+L/8)));
speckleImage=speckleImage/(3*imageMean);
end
```

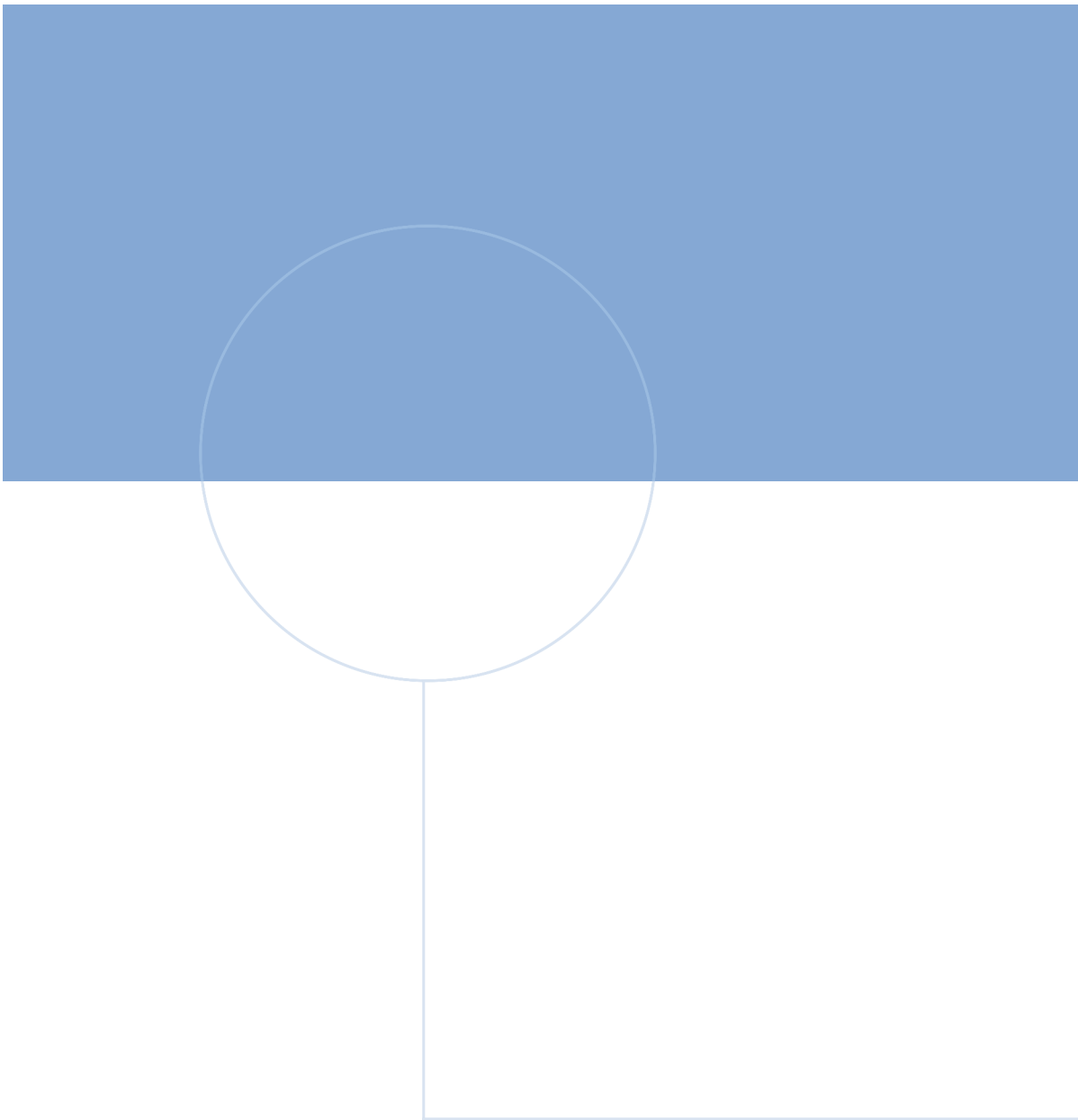
Appendix C

Github Repository

A Github repository was created if it was desirable to review the VHDL source code. This repository contains all the necessary modules to recreate the PCC and novel FFT modules, used in this thesis, within Vivado. The repository is however not documented well. For recreation, use this thesis as documentation.

Located at :

– https://github.com/Asmami/partial-CC



NTNU

Norwegian University of
Science and Technology