Master's thesis

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

Eirik Sebastian Runshaug

TEMPLATE MATCHING OF SPECKLE IMAGES USING A SUM OF ABSOLUTE DIFFERENCES ALGORITHM ON FPGA

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

NTNU
Norwegian University of
Science and Technology

Eirik Sebastian Runshaug

TEMPLATE MATCHING OF SPECKLE IMAGES USING A SUM OF ABSOLUTE DIFFERENCES ALGORITHM ON FPGA

Master's thesis in MTELSYS
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

Eirik S. Runshaug

TEMPLATE MATCHING OF SPECKLE IMAGES USING A SUM OF ABSOLUTE DIFFERENCES ALGORITHM ON FPGA

Master thesis
for the degree MSc in Electronic System Design and Innovation

Supervisor: Per Gunnar Kjeldsberg
Co-supervisor: Tormod Njølstad

Trondheim, June 2023

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

NTNU
Innovation and Creativity

NTNU

Norwegian University of Science and Technology

Master thesis
for the degree of MSc in Electronic System Design and Innovation

Supervisor: Per Gunnar Kjeldsberg
Co-supervisor: Tormod Njølstad

Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Task description

Original title: Fast correlation computation and efficient memory handling

Fast correlation computations are usually needed in medical image processing, in video analysis, for tracking, in radar systems and in many other applications. This assignment will include a literature review and will explore the possibilities for and implementation aspects related to performing fast correlation computations and template matching using a system-on-a-chip field-programmable logic array chip (SoC FPGA) on real-time camera images. For the practical part of this project, a Xilinx ZCU104 development board (for a Xilinx "multiprocessor system-on-a-chip" (MPSoC) Zynq Ultrascale+ FPGA will be utilized.

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 a MPSoC, one or more parallel correlation computations can be executed very fast in the on-chip programmable logic (PL), while efficient memory handling may be controlled by the on-chip microcontroller system (PS) connected to 4GB DDR4 memory. By using Jupyter and the PYNQ framework, a test bench 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. For inspiration: `https://ieeexplore.ieee.org/document/9122194`

Abstract

Template matching is the process of locating an image called a template inside a larger image called a reference. This is a powerful tool within image processing which can be used for a variety of applications including object tracking, material strain measurements and facial recognition. Template matching can become a computationally complex process when images get larger, efficient implementations are therefore important for fast template matching.

This thesis investigates template matching algorithms and examples of their hardware implementations for efficient image processing. The studied algorithms are Normalized Cross-Correlation, Zero-mean Normalized Cross Correlation, and Sum of Absolute Differences, which are commonly used for template matching applications. A comparison is also made with a Fourier transform approach for Cross-Correlation.

The chosen algorithm for implementation on an FPGA is Sum of Absolute Differences. However, due to incorrect implementation and sub-optimal memory organization, the hardware template matching process was not completed. Estimations indicate that the slow memory organization would result in slower performance compared to a high-level alternative. Theoretical execution with more efficient memory organization demonstrates significant speed improvements.

In terms of accuracy, the algorithm achieves 100 % accuracy for template images half the width of the reference. For templates one-quarter the width of the reference, accuracy ranges from 82 % to 94 %, with maximum errors of 1 pixel away from the correct location. Smaller template widths exhibit further reduced accuracy, with a maximum accuracy of 52 % for an eighth of the reference width and significant errors, including a maximum error of 191 pixels for a 256-pixel-wide reference.

Sammendrag

Malmatching er det å lokalisere et bilde, kalt en mal, inni et større bilde, kalt en referanse. Dette er et kraftig verktøy innen bildeprosessering og kan brukes til en rekke applikasjoner, deriblant sporing av objekter, måling av materiell belastning og ansiktsgjenkjenning. Denne prosessen kan bli tung beregningsmessig etterhvert som bildene blir større, derfor er det viktig med raske implementasjoner.

Denne oppgaven undersøker algoritmer for malmaching og eksempler på deres maskinvareimplementasjoner for effektiv bildebehandling. Algoritmene er Normalisert Kryss-korrelasjon, Null-gjennomsnittlig Normalisert Kryss-korrelsjon og Sum av Absolutte differanser, som er vanlige å bruke i malmatching. Utførelse av kryss-korrelasjon gjennom Fourier transform er også sammenlignet.

Den valgte algoritmen for implementasjon på FPGA er sum av absolutte differanser. Derimot, på grunn av feil implementering og ikke-optimal minnehåndtering var malmatchingprosessen ikke gjennomført. Esimater indikerer at minneorganisasjonen i et fungerende systemunsett ville vært tregere enn en alternativ høynivå modell. Teoretisk utførelse med bedre minneorganisasjon viser til tydelige akselerasjoner på minst 25 ganger relativt en høynivå modell.

Støymessig er algoritmen 100 % treffsikker for maler med bredde halvparten så stor som bredden til referansen. For maler med bredde lik en kvart referanse synker treffsikkerheten til rundt 82% til 94%, med de største feilene på 1 piksel fra korrekt plassering. Mindre malbredder viser ytterligere redusert treffsikkerhet, hvor den største var for 52% for en bredde på en åttendedel av referansen, med feil så store som 191 piksler i et referansebilde 256-piksler bredt.

Preface

This master's thesis is written at the Department of Electronic Systems at the Norwegian University of Science and Technology. I want to thank my supervisors Tormod Njølstad and Per Gunnar Kjeldsberg, as well as Gudmund Slettemoen for help and guidance. I also want to thank my fellow students, for creating a fun and encouraging environment at campus.

Contents

List of Tables

List of Figures

Source code

List of abbreviations

FPGA Field-Programmable Gate Array

FFT Fast Fourier-Transform

DFT Discrete Fourier-Transform

CC Cross-Correlation

NCC Normalized Cross-Correlation

ZNCC Zero-mean Normalized Cross-Correlation

MAC Multiply Accumulate

MACC Multiply Accumulate Chain

COMAC Concurrent Multiply Accumulate

CAMAC Cascading Multiply Accumulate

SAD Sum of Absolute Differences

HDL Hardware Descriptive Language

AD Absolute Difference

DSP Digital Signal Processing

RAM Random Access Memory

BRAM Block RAM

PS Processing System

PL Programmable Logic

PYNQ Python Productivity for ZYNQ

DMA Direct Memory Access

FIFO First In First Out

AXIS AXI Stream

Chapter 1

Introduction

1.1 Motivation

Acquiring information about our surroundings and recognizing objects and movement is an ability humans are great at. A challenging task is to give this ability to machines, which is what the field of image processing tries to accomplish [4]. In recent years there has been great advances within the field of image processing, and an important task is *template matching* [5].

Template matching is a technique used in computer vision and image processing to identify the best match of a template image in a larger reference image. It is often used to find and recognize objects in images and videos, such as faces, vehicles, or specific shapes. This technique can be useful for a variety of applications, such as object tracking, image recognition, and image alignment [5]. It can also be used to measure surface deformation in materials and structures as well as fingerprint matching [6][7]. It is often used in conjunction with other computer vision techniques to provide more robust and accurate results.

1.2 System specification

The system in this report will be made to perform template matching on speckle images created by the refraction of lasers. Capturing an image of the laser's refraction creates a unique pattern. The pattern's uniqueness is great for template matching, as a template will only have one exact match.

A template matching system must be able to correctly perform template matching on these images and find the location of the template, or its best match within the reference. The system should be able to locate a template when the images has presence of noise prevalent in images captured by current camera technology.

In Figure 1.1 is a flowchart of the system to be implemented. The system will be input a reference image, and a template will be extracted to be located by the system. The system will iterate through the reference and extract sub-images to be compared with the template. The sub-images and template will be decimated in order to easy computational complexity. Since speckle images are not trivial to produce, they will be computer generated [8]. In an experimental setup, the template would be computer generated, while the reference is an image captured by a camera. In order to account for this, it will be added image noise to the reference. Noise prevalent in camera technology and how to emulate this in software is described in Section 2.5. The template and reference will be compared using a similarity calculating algorithm. There are multiple different algorithms one can utilize for template

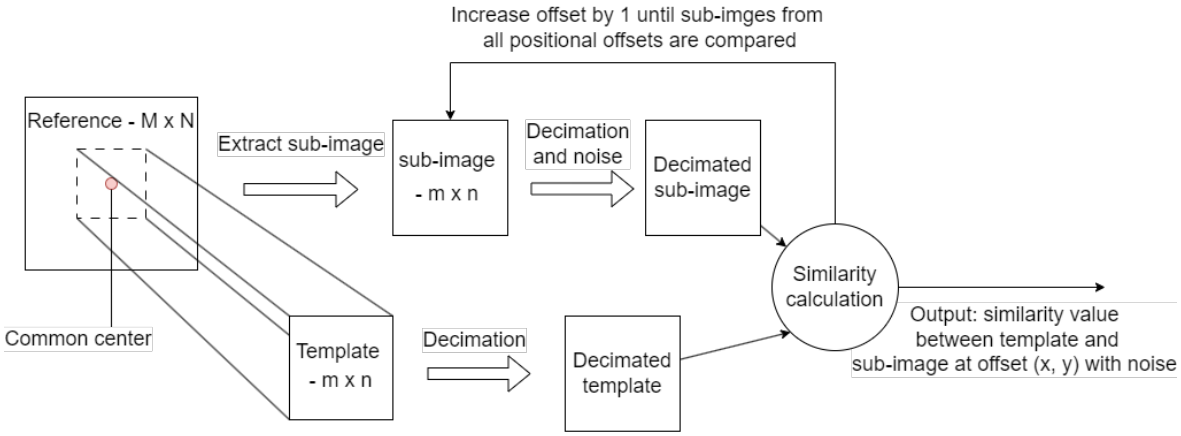


Figure 1.1: Flowchart of system design. Reference is created by a computer script, and noise is added after the template is extracted. The offset is increased first in the x-direction until $x = M - m$, then reset x-offset and increase y-offset until $y = N - n$ and $x = M - m$.

matching, some of which are presented in Section 2. The system will find the sub-image which is the most similar to the template, thus locating the template within the reference.

1.3 Objectives and limitations

The goal of this thesis is to implement a template matching algorithm on a Field-Programmable Gate Array (FPGA) in order to accelerate the template matching process relative to an alternative approach using traditional programming. The FPGA is Xilinx' ZCU104 evaluation kit [9]. The template matching accelerator is the focus of the thesis, therefore pre-existing modules, Intellectual Properties (IPs), will be used in the design where needed. The system will be tested in the PYNQ framwork for convenience, as this framework is designed for quick and efficient testing and chracterization of hardware.

Because of the limited timescope of the thesis, the main goal is to implement a template matching algrithm on the FPGA, with a chosen similarity measurement technique as its basis. The characterization of this implementation relative to an implementation using traditional programming is of interest. In this context, characterizing the system implies testing it on different reference and template sizes in order to see how the system operates in different configurations.

Further, the accuracy of the system in noisy conditions is an area for research. This includes implementing techniques for generating noise of varying degree and adding this noise to the speckle images, as well as inputting these images to the system to see at what amounts of noise the system starts to fail at locating the template.

1.4 Structure

The report will first introduce a selection of similarity measurement techniques which are commonly used for template matching in Chapter 2.

Chapter 3 describes previously designed hardware architectures which perform template matching with the similarity measurement techniques presented in Chapter 2. The architectures execution time will be compared for different reference and template sizes.

In Chapter 4 the design and implementation of the algorithm with the chosen similarity measurement technique will be presented.

Chapter 5 will present the results for the implementation described in Chapter 4.

Chapter 6 contains discussion around the results and why they turned out the way they did, as well as thoughts on how to improve the system and what lies ahead as future work.

Lastly, Chapter 7 presents a conclusion.

Chapter 2

Theory

In this chapter, the theory for some different similarity measurement techniques commonly used for template matching is presented. Because of template matching's plenty of use cases, many different methods have been devised to perform this technique [10]. The information presented was first written during a literature study related to a specialization project during fall of 2022. The theory is still relevant in this thesis and has therefore been included, however it has been built upon and refined.

2.1 Correlation

One method of measuring the similarity between two images is Cross-Correlation (CC) [5]. The equation for cross correlation can be seen in Equation 2.1, where $T_{i,j}$ and $R_{i,j}$ is template and reference pixel in row i and column j respectively [1].

$$CC(x,y) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} T_{i,j} R_{x+i,y+j} \tag{2.1}$$

The principle of cross correlation is to "slide" the template image over the reference image, and for every position (x, y) of the template within the reference, sum the products of the values for each overlapping pixel. Figure 2.1 shows a template T slided over a reference R. The position (x, y) of the top left pixel of the template is called the *offset* which Equation 2.1 is evaluated on. The offsets range from $0 \leq x \leq M - m$ and $0 \leq y \leq n$. Therefore, after Equation 2.1 has been applied to all offsets, one can find which offset yields the highest output value. Finding the highest value should correspond to the reference's sub-image that is the most similar to the template.

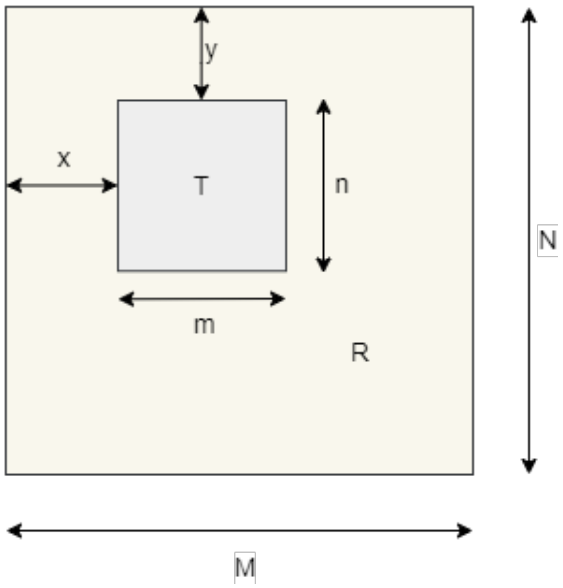


Figure 2.1: Sliding window similarity calculation. The template T is slid over the reference R. For each offset (x, y) the similarity is calculated resulting in a single value. Based on Figure 1 from [1].

This CC technique however is not sufficient for accurate template matching [5]. In an image with varying lighting conditions, very bright areas will have a high CC output, even though the area does not match the template. Similarly, dark areas will give small outputs despite a good match [11].

Template

1	1	1
1	2	1
1	1	1

Reference

6	7	5	6	7
6	8	7	4	3
5	7	1	1	1
9	5	1	1	1
8	5	1	1	1

Figure 2.2: Example template and reference.

In Figure 2.2, it is obvious that the best match of the template is at offset $(x, y) = (2, 2)$, as this is the part of the reference that is the most similar. If we use Equation 2.1 to compare both $(2, 2)$ and for example $(0, 0)$ with the template we get

$$CC(2,2) = 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 + 2 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 = 10$$
$$CC(0,0) = 1 \cdot 6 + 1 \cdot 7 + 1 \cdot 5 + 1 \cdot 6 + 2 \cdot 8 + 1 \cdot 7 + 1 \cdot 5 + 1 \cdot 7 + 1 \cdot 1 = 60.$$

Offset (0,0) gets a higher correlation output while not being similar at all, and this needs to be accounted for.

2.1.1 Normalized cross correlation

Normalization is a process where each CC output is divided by the *energy* of the two images as seen in equation Equation 2.2 [1]. This way, one accounts for the possibility that parts

of the reference are more illuminated than others and the output should be more resilient to variable lighting conditions [5]. Note that only the templates energy is constant, as the entire template is constant and slides over the reference. The reference energy must however be calculated for each offset, as the sub-image that is correlated with the template is new for each offset.

The equation for Normalized Cross-Correlation (NCC) can be written as seen in Equation 2.2 [12].

$$NCC(x,y) = \frac{\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(T_{i,j})(R_{x+i,y+j})}{\sqrt{\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(T_{i,j})^2 \cdot \sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(R_{x+i,y+j})^2}} \tag{2.2}$$

We can use the NCC technique on the example in Figure 2.2.

$$\text{Template energy:} \quad \sqrt{\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(T_{i,j})^2} = \sqrt{12}$$

$$\text{Reference energy (0, 0):} \quad \sqrt{\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(R_{i,j})^2} = \sqrt{334}$$

$$\text{Reference energy (2, 2):} \quad \sqrt{\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(R_{2+i,2+j})^2} = \sqrt{9}$$

$$NCC(0,0) = \frac{60}{\sqrt{12 \cdot 344}} = 0.93$$

$$NCC(2,2) = \frac{10}{\sqrt{12 \cdot 9}} = 0.96$$

Using the NCC technique one would find the correct location of the best match, in contrast with the CC technique. Also note that a perfect match with this technique would output a value of 1. This is also a benefit of the normalization, where possible output values are normalize within the range 0 to 1 [12, pp. 121].

2.1.2 Zero-mean normalized cross correlation

Another variation of the CC technique is Zero-mean Normalized Cross-Correlation (ZNCC). ZNCC is similar to NCC, however with the addition of subtracting the template's and reference's mean from all the summations, as seen in Equation 2.3 [1].

$$ZNCC(x,y) = \frac{\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(T_{i,j} - \overline{T})(R_{x+i,y+j} - \overline{R}_{x,y})}{\sqrt{\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(T_{i,j} - \overline{T})^2 \cdot \sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(R_{x+i,y+j} - \overline{R}_{x,y})}} \tag{2.3}$$

where $\overline{T}$ is the mean of the Template image and $\overline{R}_{x,y}$ is the mean of the sub-image at offset (x, y) of the reference.

Again, we can use this algorithm on the example shown in Figure 2.2.

$$\overline{T} = 1.11$$
$$\overline{R}_{0,0} = \frac{60}{9}$$
$$\overline{R}_{2,2} = 1$$

$$\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(T_{i,j} - \overline{T})(R_{0+i,0+j} - \overline{R}_{0,0}) = 2.22$$
$$\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(T_{i,j} - \overline{T})(R_{2+i,2+j} - \overline{R}_{2,2}) = 0$$

Zero-mean template energy: $$\sqrt{\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(T_{i,j} - \overline{T})^2} = 0.89$$

Zero-mean reference energy (0, 0): $$\sqrt{\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(R_{i,j} - \overline{R_{0,0}})^2} = 5.79$$

Zero-mean reference energy (2, 2): $$\sqrt{\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(R_{2+i,2+j} - \overline{R_{2,2}})^2} = 0$$

$$ZNCC(0,0) = \frac{2{,}22}{0.89 \cdot 344} = 0.074$$
$$ZNCC(2,2) = \frac{0}{0.89 \cdot 0} = undefined$$

We see that in the template's best match in the reference, all the elements are the same. Because of this, the denominator in the ZNCC equation becomes zero, which creates undefined behavior. This needs to be addressed, and one possibility is to assign the output to zero for all sub-images where this operation is undefined. This is how the MATLAB function for the ZNCC algorithm is implemented [13]. Also note that the operation is also undefined for templates with all equal elements, the MATLAB function handles this by not accepting templates where this is true [13].

2.2 Sum of absolute differences

Sum of absolute differences is an algorithm with same basic principle of cross-correlation algorithms, in that the template is slid over the reference and a similarity-value is computed at each offset. The formula for the algorithm can be see in Equation 2.4 [14].

$$SAD(x,y) = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1}|T_{i,j} - R_{x+i,y+j}|. \tag{2.4}$$

In the SAD algorithm the offset which results in the best match is not found by finding the maximum value of the outputs, but the minimum. We can use the algorithm on the example from Section 2.1 to see this.

$$\begin{aligned} SAD(2,2) &= |(1-1)| + |(1-1)| + |(1-1)| + |(1-1)| + (2-1)| \\ &\quad + |(1-1)| + |(1-1)| + |(1-1)| + |(1-1)| && = 1 \\ SAD(0,0) &= |(1-6)| + |(1-7)| + |(1-5)| + |(1-6)| + |(2-8)| \\ &\quad + |(1-7)| + |(1-5)| + |(1-7)| + |(1-1)| && = 41 \end{aligned}$$

We see that the template's best match at offset $(2, 2)$ gives 1 as output when compared with the template. Thus, the more two images look alike, the smaller the result of the SAD algorithm is. When the images have been processed, one can find the offset that results in the lowest value. Sum of absolute differences is widely used in hardware systems because of its simplicity [15]. There is no normalizing element in this algorithm, and it can therefore be more susceptible to illumination variations.

2.3 Correlation through Fourier transform

Another method for template matching is applying the Fourier transform on the template and reference images.

The convolution theorem is stated in Equation 2.5, where $*$ denotes the convolution between the two functions [12, pp. 283].

$$\mathscr{F}\{x(t) * y(t)\} = \mathscr{F}\{x(t)\}\mathscr{F}\{y(t)\} \tag{2.5}$$

Convolution and cross-correlation are in principle the same operation, where one of the sequences are complex conjugated and time-inverted [12]. Because of this, Equation 2.5 can be rewritten into Equation 2.6, which is knows as the *correlation theorem* [12, pp. 284].

$$x(t) \star y(t) \ = \mathscr{F}^{-1}\{\mathscr{F}\{x(t)\}\,(\mathscr{F}\{y(t)\})^*\} \tag{2.6}$$

In Equation 2.6, $\star$ denotes cross-correlation, $\mathscr{F}^{-1}$ is the inverse Fourier transform operator and * means the complex conjugate.

The Fourier transform, from [12, pp. 236] is defined as

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-2j\pi ft}dt \tag{2.7}$$

The inverse operation is similarly defined as

$$x(t) = \int_{-\infty}^{\infty} X(f)e^{2j\pi ft}dt \tag{2.8}$$

Since images in hardware are not continuous signals, but sequences of numbers, eq. (2.7) cannot be used. There is however the Discrete Fourier Transform (DFT), which can be used for this purpose [12, pp. 456]. The definition of DFT can be seen in Equation 2.9.

$$X(k) = \sum_{n=0}^{L-1} x(n)e^{-j2\pi kn/L} \tag{2.9}$$

To use this for images, the equation must be extended to include more than one dimension. The definition of the two-dimensional Discrete Fourier-Transform (DFT) can be seen in Equation 2.10 [16, pp. 127].

$$F(m,n) = \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} f_{k,l}\, e^{-j2\pi\left(\frac{k}{M}m + \frac{l}{N}n\right)} \tag{2.10}$$

$M \times N$ is the image's size and $f_{k,l}$ is the pixel value of pixel at position (k,l). The result $F(m,n)$ is a new image, which has the same size $M \times N$, containing complex valued elements representing the prevalence of the frequency component represented by the indices m and n.

Similarly to Equation 2.8, the inverse two dimensional discrete Fourier transform is

$$f(k,l) = \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} f_{k,l}\, e^{j2\pi\left(\frac{k}{M}m + \frac{l}{N}n\right)} \tag{2.11}$$

The definition presented in Equation 2.10 is incredibly slow for larger sequences [12, pp. 511]. Thus, a more efficient variant of the DFT has been developed, the Fast Fourier Transform (FFT) [12, p. 512].

The Fast Fourier transform uses a divide and conquer approach to divide the DFT calculation into multiple smaller calculations. This produces the same results, but exponentially quicker for larger sequences. The algorithm is most efficient for sequence lengths that are a power of 2 [12, p.519].

2.4 Speckle images

The purpose of this project is to consider these different template matching algorithms in order to use them on speckle images. Speckle images in this context are images of a laser reflected off of a surface. The roughness of the surface will scatter the light. The scatter will purely be dependent on the surface, and will be unique for that exact position of laser and surface [8]. By capturing images with cameras of the scattered light, one can exploit the uniqueness of the scatter in order to perform accurate template matching [8].

Experimental creation of speckle images are not within the scope of this thesis, they will be computer generated [8]. The image generating script has been developed in Matlab by [8]. It is done by taking the Fourier Transform of a phase matrix, as described in [17].

Using computer generated images one can create images of different sizes and different amounts of noise for system testing purposes.

Figure 2.3 depicts two speckle images, the left one generated by the computer script created by [8] and the right one experimentally created by [8].

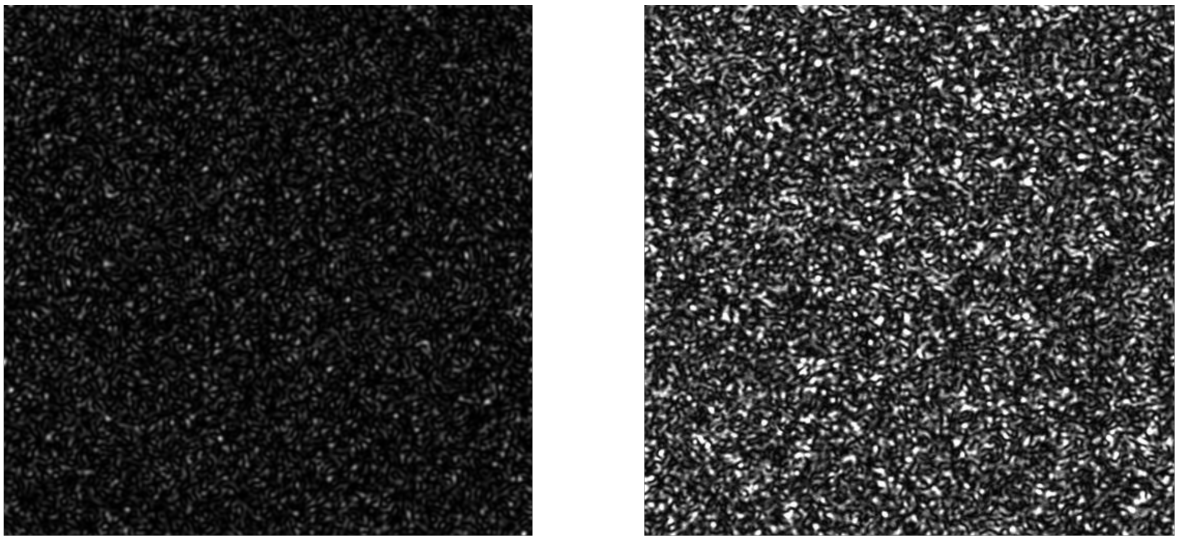


Figure 2.3: Computer generated speckle image compared to an experimentally generated speckle image.

There is a clear difference between the two images, however for our testing purposes the computer generated ones should hold up. The script is more for the convenience of creating these images in different sizes.

2.5 Image noise

As with any sensor, there is bound to be noise in the final output of the camera sensor taking images. By knowing the main sources of noise and their characterization, it is possible to simulate the noise in a theoretical situation. This way, one can test the system and get an understanding of how well the system might function in a real application.

A detailed explanation of image noise is not within the scope of this thesis, and will therefore not be explained in detail. However, since it of interest to test a template matching system on realistic images, some theory on the subject will be presented.

Images are captured by photons hitting a detector, and *photon shot noise* is created by how the photons spatially arrive [18]. Photon shot noise is described using the Poisson distribution, which can be approximated to a Gaussian distribution for large amounts of arriving photons per pixel [18]. Shot noise is dependent on the signal intensity, where the amount of noise increases by the square root of the signal [18].

2.6 PYNQ

Python Productivity for ZYNQ (PYNQ) is an open-source project from AMD which enables programmers to utilize the programmable logic of FPGAs using a higher level of abstraction [19]. The PYNQ framework runs as a Jupyter Notebook, this way one can write high level Python code which interacts with the programmable logic of the FPGA. The programmable logic circuits are imported into the PYNQ framework as *overlays*, which can be seen as a hardware library, but are analogous to software libraries [19]. This enables engineers to create Adaptive Computing systems [19]. In this thesis, the framework will be used as a convenience

tool for testing and characterizing the hardware created. The Jupyter notebook can run the speckle-image script described in Section 2.4 and communicate this data to the hardware as familiar numpy-arrays, which is convenient.

Chapter 3

Existing hardware architectures

In this section some hardware architectures implementing the similarity measuring algorithms in Section 2 are presented. This section, as with Section 2, is based on previous work from specialization project in fall 2022. However, the information is relevant to this thesis and has therefore been further improved and refined.

3.1 Hardware implementations of CC

The backbone of the CC algorithms are Multiply-Accumulate operations (MAC), which are simply the multiplication of many pairs of numbers, and summation of all the products. The amount of MAC operations necessary in order to perform template matching is large, therefore it is needed to find smart ways to calculate in parallel.

In this chapter, two implementations of MAC calculation will be presented: Concurrent Multiply Accumulate (COMAC) and Cascading Multiply Accumulate (CAMAC) respectively.

3.1.1 COMAC

The COMAC unit can be seen in figure 3.1. These can be cascaded to form a Multiply Accumulate Chain (MACC) [1].

$$\sum_{i=0}^{m-1} a_i b_i = a_0 b_0 + a_1 b_1 + \dots a_{M-1} b_{M-1} \tag{3.1}$$

Equation 3.1 is the expression that is to be calculated using the MAC. Consider the first MAC operation. The multiplication of a_0 and b_0 and addition of 0. Using the COMAC unit, a_0 is input on A_{in} and b_0 is input on B_{in}. The numbers are multiplied and the result is saved in the intermediary register E. This saved result is added with the value in the result register R, which initially is 0. The next iteration, a_1 and b_1 are input, multiplied and saved in register E. This time, the addition is between the previous result $a_0 b_0$ and the current result $a_1 b_1$.

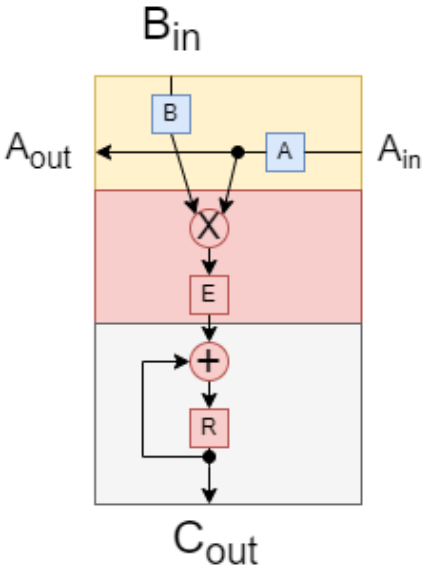


Figure 3.1: COMAC unit, based on Figure 2.b in [1].

Equation 3.2 shows the split up sum of the cross-correlation equation. If one compares with Equation 3.1, it is clear to see that Equation 3.2 is just an instance of Equation 3.1 for each offset l.

$$CC(l) = \sum_{i=0}^{M-1} a_i b_{i+l} \tag{3.2}$$

Knowing this, the configuration can be parallelized. The result of this parallelization can be seen in Figure 3.2 This configuration can calculate c sums concurrently, where c is the amount if MAC units in the chain. If the length of the reference sequence r is larger than the amount of MAC units, i.e. $r > C$, the configuration needs multiple computing rounds to compute Equation 3.2 for each l [1].

A table showing the flow of data and which MAC unit that contains which data each clock cycle can be found in the original report [1].

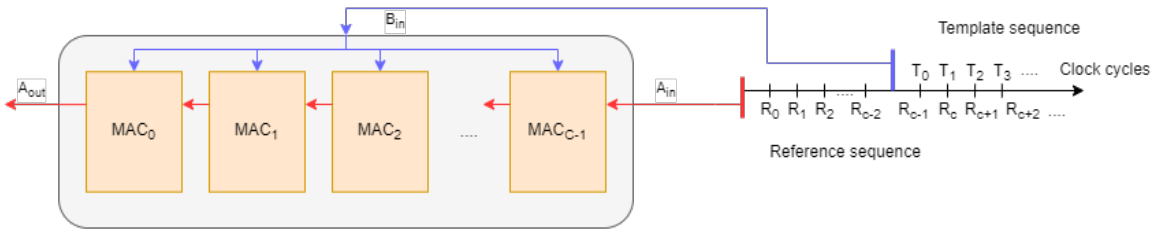


Figure 3.2: Multiply accumulate chain consisting of COMAC units, based on Figure 4 in [1].

Computations from [1] show that the amount of clock cycles needed to compute the cross correlation between a template image and a reference image with sizes $m \times n$ and $M \times N$ respectively when the MACC length is C is

$$T_{COMAC} = (m + C - 1)n \cdot \frac{M \times N}{C} \tag{3.3}$$

Table 3.1: Experimental results for COMAC cross-correlation [1].

Reference size, kernel size (Width x Height)	Time consumption CC (ms)	Time consumption ZNCC (ms)	Total time consumption (ms)
128x128, 64x64	0.654	0.360	1.014
320x256, 64x64	12.005	1.801	13.806
176x176, 128x128	0.802	0.651	1.453

Experimental results for COMAC based cross-correlation can be seen in Table 3.1 [1]. The results are from a setup with a two-dimensional MACC with dimensions 32×34 running at 100 MHz. The ZNCC part of the calculation consists of data transfer between a main processor running at 500 MHz calculating the standard deviation as well as the division of the CC calculation the standard deviation [1].

3.1.2 CAMAC

A cascading multiply accumulate unit can be seen in Figure 3.3. In this setup, the unit has an extra input compared to the COMAC unit, C_{in}. This is where intermediary results are input when chaining the units.

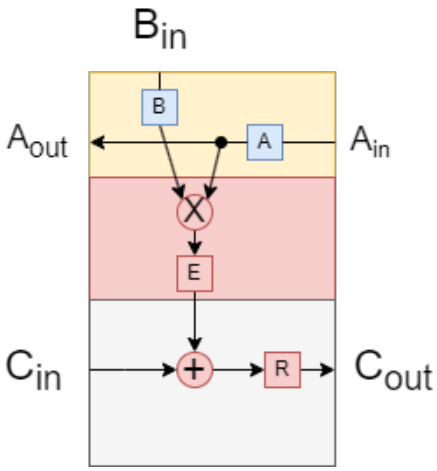


Figure 3.3: CAMAC unit, based on Figure 2.a in [1].

The CAMAC based MACC can be seen in Figure 3.4. In this case, the template pixels are pre-loaded into the MACs, and the reference is iterated through. Each MAC adds an addend to the sum, e.g. MAC_0 first adds $a_0 \cdot b_i$ to the chain, which is cascaded through the chain and added to the similar addends from the other MACs, MAC_1 adds $a_1 \cdot b_i$ and so on. After enough clock cycles, the finished sums are output each clock cycle on C_{out} of MAC_{c-1}.

If more than one computation rounds are needed, i.e. $C < k$ where k is the length of the reference sequence, the partly completed sums output from C_{out} after the previous round are input on C_{in}. As with COMAC, a detailed table showing the dataflow can be found in [1]. Similarly as for COMAC shown in Section 3.1.1, a formula for the number of clock cycles for a template with dimensions $m \times n$, a reference with dimensions $M \times N$ and a MACC length of C can be seen in Equation 3.4.

$$T_{CAMAC} = ((M - m + C)(N - n) + C)\frac{m \cdot n}{C} \tag{3.4}$$

Experimental results for the implementation using CAMAC units can be seen in Table 3.2. The experimental setup is the same as for COMAC in Section 3.1.1 [1].

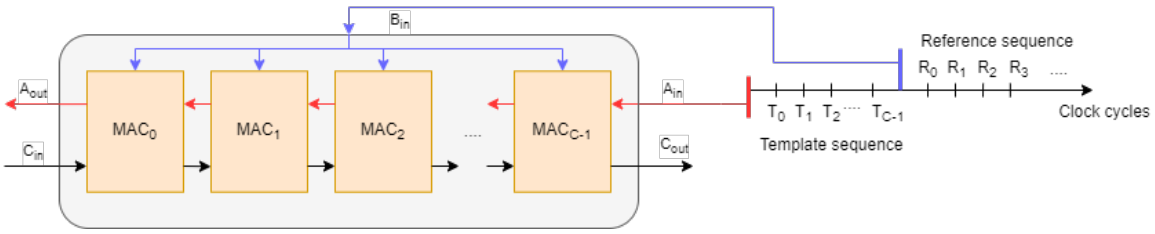


Figure 3.4: MACC built by CAMAC units, based on Figure 3 in [1].

Table 3.2: Experimental results for CAMAC cross-correlation [1].

Reference size, template size (Width x Height)	Time consumption CC (ms)	Time consumption ZNCC (ms)	Total time consumption (ms)
128x128, 64x64	0.464	0.360	0.824
320x256, 64x64	2.724	1.801	4.525
176x176, 128x128	1.804	0.651	2.455

3.2 Hardware implementations of sum of differences

3.2.1 SAD

Sum of absolute differences is a slow algorithm to implement in software running on processor cores, because of the sequential characteristic of these implementations [20]. One simple way of implementing Sum of Absolute Differences (SAD) in hardware, can be seen in figure 3.5 [3]. This module has W Absolute Difference (AD) units which receives W template and reference pixels in parallel. The output of each AD block is sent to an adder with another AD block's output. The sum is trickles down until the absolute difference of all W pixel pairs have been added. Then the next W pixel pairs can be input.

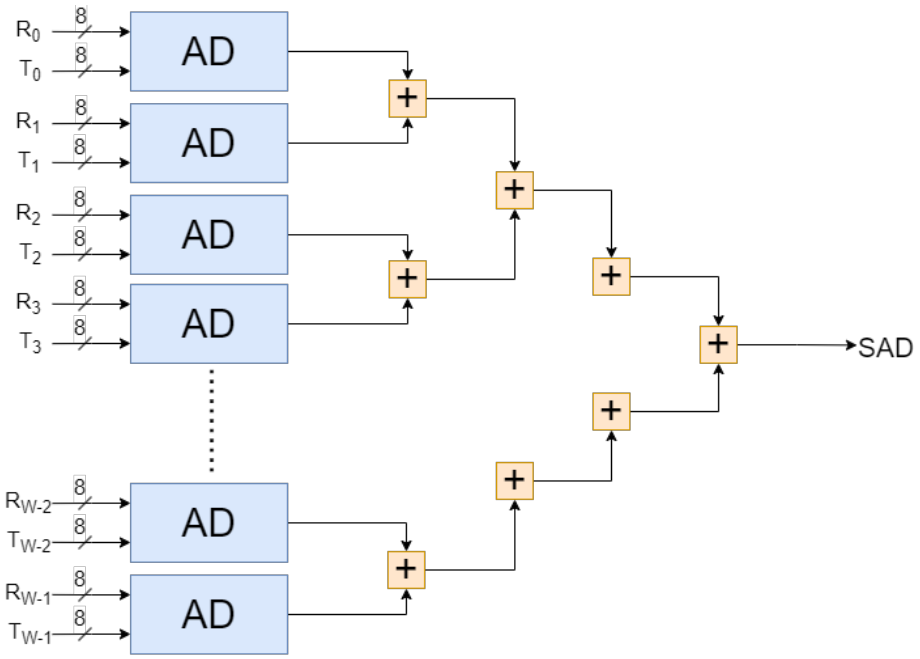


Figure 3.5: Generic SAD implementation in hardware, based on Figure 1 in [3].

This configuration can also benefit heavily from pipelining by placing intermediary registers between every layer in the tree, giving potential for increased clock frequency.

In [21], a similar architecture has been used for human detection in images. The implementation is specialized for templates with a size of 100×40 pixels and references with a size of 640×480. The implementation uses 4000 AD units, i.e. $W = 40 * 100$. This means that one round can compute the SAD of the entire template and subimage, and the computation time is purely dependent on the frequency and reference size.

Simulation results from [21] show that for a reference image with dimensions 640×480 pixels and FPGA frequency of 100 MHz, one complete computation takes 3.072 ms. This is one cycle for each possible offset in the reference. Thus, as long as the chosen FPGA has enough resources for $W \geq m \times n$, one can easily get the theoretical computation time for a given reference size.

For every offset (x, y) there is performed $m \times n$ AD operations. The offset's ranges are $0 \leq x \leq M - m$ and $0 \leq y \leq N - n$, meaning that there are $(M - m) \cdot (N - n)$ sub-images. For each sub-image there are $m \cdot n$ pixels that need to be compared with the corresponding pixel in the $m \times n$ large template. The total number of SAD operation are therefore

$$T_{SAD} = (M - m)(N - n) \cdot mn \tag{3.5}$$

By taking into account that W pixels can be input in parallel, one can easily find an expression for how many operations are needed. The expression can be seen in Equation 3.6.

$$T_{SAD} = (M - m)(N - n) \cdot \left\lceil \frac{mn}{W} \right\rceil \tag{3.6}$$

where $\lceil \dots \rceil$ is the ceiling function rounding up. This is to account for configurations where W is not a power of two. Since the template dimensions are powers of two, dividing by a factor which is not a factor of two means that the last computation round would compute less that W pixels. However, this would not change the fact that the system would need a full clock cycle to compute.

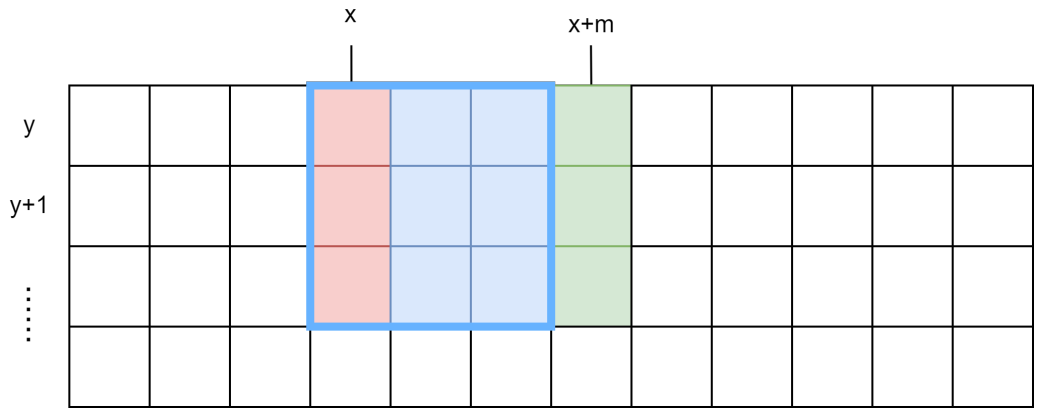


Figure 3.6: Optimization of SAD algorithm. In the next clock cycle, column x is subtracted and column $x + m + 1$ is added.

3.2.2 Optimizing the SAD Architecture

The implementation seen in Figure 3.5 is not be the most efficient, as it calculates many of the same values multiple times. The difference between $SAD(x, y)$ and $SAD(x+1, y)$ is only two columns containing in total $2 \cdot n$ pixels. Therefore, a more efficient way of calculating $SAD(x + 1, y)$ after calculating $SAD(x, y)$ is subtracting the column (x, y) and adding the column $(x + m, y)$, both with n pixels [15]. This optimization technique is depicted in Figure 3.6. The figure shows the current template inside the blue square. During this clock cycle, the SAD of the square is calculated, with top left corner at pixel (x, y). In the next clock cycle, the red column starting at pixel (x, y) is simply subtracted while the green column is added. This yields an equivalent result, but requires only n additions every clock cycle, where n is the template's y-dimension. This is because as only the pixels in green need to be added. This hardware naturally needs extra hardware in order to keep track of all the columns' partial sum. One way of implementing this is through a First In First Out (FIFO) buffer [15].

The technique may also the amount of data that needs to be transferred between memory and logic performing the algorithm, as an entire template an sub-image is not needed every clock cycle. SAD template matching can in other words be optimized by reusing the pixels which overlap in the subsequent calculations that have already been calculated.

Another optimization technique is placing intermediary register at every level in the SAD tree depicted in Figure 3.5, meaning that all adders output their intermediary sum into a register. This shortens the critical path through the SAD tree and may enable the system to run at a higher frequency [22].

One could also implement a customized adder-architecture for the system [3]. This can potentially further increase maximum frequency, as well as decreasing resource usage and power consumption.

3.3 Hardware implementation of FFT

When using the Fast Fourier Transform to make correlation calculations, one requires three FFT calculations: one for the template, one for the reference, and one inverse for their product to get the resulting correlation. However, the template only needs to be transformed once, since it is constant throughout the template matching process. Thus, the time it takes

for the implementation to transform a reference sub-image, perform a complex multiplication of two matrices, and inversely transform the product is of interest.

[2] proposes a spectral correlation architecture which performs this. The proposed architecture can be seen in Figure 3.7.

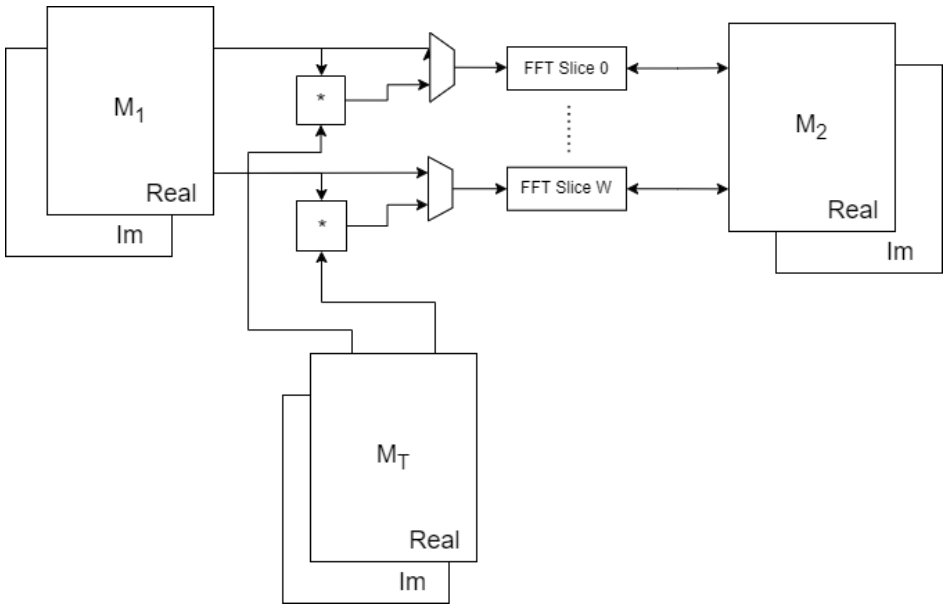


Figure 3.7: Proposed FFT architecture, based on Figure 2 in [2].

In this architecture, there are three internal memories on the FPGA: M_1, M_2 and M_T. M_T contains the transformed template, i.e. the result of using Equation 2.10 on the template which remains unchanged throughout the template matching process. M_1 stores the input reference and the end result. M_2 stores the intermediary result. When a reference image is received and the template is to be located inside it, it is first loaded into M_1. At this point, no transforms has been done other than the transformation of the template which is stored in M_T. First, all the rows of M_1 are transformed and stored in M_2. Then, the columns of M_2 are transformed and stored in M_1. After these two steps, M_1 contains the FFT of the reference. Next, the rows of M_1 and the conjugate of the corresponding rows of M_T are multiplied and inversely transformed and stored in M_2. Lastly, the inverse transform of the columns of M_2 are computed and stored in M_1. After these four steps, M_1 contains the cross-correlation result [2].

Each FFT slice has to read and write data in multiple memory blocks. Because of this, the implementation requires a set of multiplexers/demultiplexers between the FFT and the memory blocks. When two FFT slices need to read or write data to the same block in memory, access collisions occurs, which must be avoided. Thus, each FFT slice are offset by one clock cycle during the first and third step, i.e. the memory is accessed diagonally. This means that when an FFT slice accesses a different row or column it can read and write data every clock cycle. The memory access organization can be seen in Figure 3.8 [2].

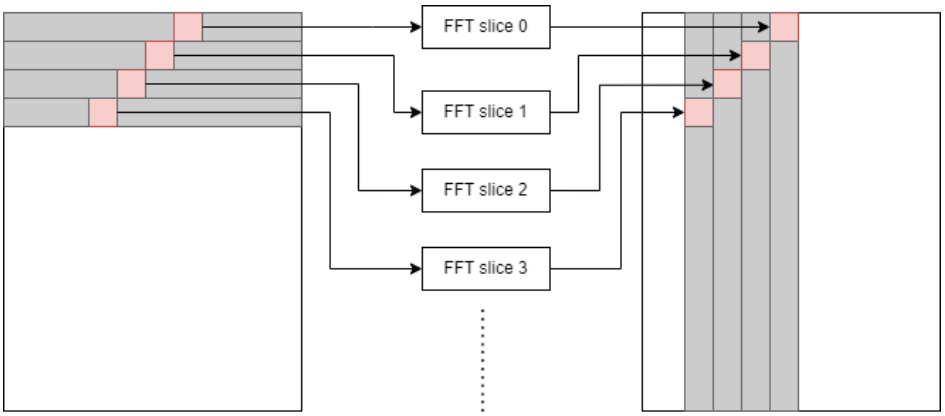


Figure 3.8: Memory organization for the FFT implementation, based on Figure 3 in [2].

Experimental results from [2] compare this FFT implementation with a direct CC implementation using embedded Digital Signal Processing (DSP) slices in the FPGA which computes MAC operations. The performance results can be seen in table 3.3. The results was obtained on a Virtex-4 XC4SX55-11 [2].

Table 3.3: Experimental results for one correlation computation of a 256×256 template and reference. 16 FFT slices were used for FFT implementation, and for direct CC implementations, three different kernel sizes were used [2].

Implementation	Computation time (ms)	Maximum clock frequency (MHz)
Direct CC (12x12)	0.162	403.7
Direct CC (16x16)	0.160	410,2
Direct CC (20x20)	0.163	401.6
FFT CC (16 FFT slices)	0.145	244.1

The computation time in Table 3.3 are the times spent for calculating the CC table for a given template and reference image, where the reference image had a size of 256×256. For direct CC computation, three different template sizes was tested. For FFT CC, only one template size was tested, as the algorithm is independent of the template size, as long as it is smaller than the reference size. It only depends on he amount of FFT slices that can be allocated [2]. As discussed in section 2.1, cross correlation needs normalization in order to be resistant to local lighting conditions such that the algorithm can give an accurate result. The computation times table 3.3 shows does not include the normalization. The normalization and comparison can be made n parallel with correlation computation so that no significant performance penalty is introduced [2].

In [2], an additional computation is done for normalization in their final experimental results when comparing two images with the two methods. In these experiments, the two images were compared using different kernel sizes ranging from 12×12 to 50×50. However, the entirety of both images were not correlated with each other. Based on the kernel size, a set number of regions were tested. In Table 3.4, results for the two methods with different kernel sizes. In this table are also how many regions are computed on for each kernel size.

Table 3.4: Computation time of correlation with different template sizes over multiple regions between two 256×256 images [2].

Correlation algorithm	Number of regions	Computation time (s)
Direct CC (12x12)	54	0.114
Direct CC (16x16)	27	0.056
Direct CC (20x20)	18	0.038
Direct CC (32x32)	9	0.057
Direct CC (50x50)	3	0.032
FFT CC (12x12)	54	0.153
FFT CC (16 x16)	27	0.076
FFT CC (20x20)	18	0.051
FFT CC (32x32)	9	0.025
FFT CC (50x50)	3	0.008

This means that the proposed spectral implementation can perform three 50×50 correlation computations with normalization in 8 ms with a kernel size of 50×50 and 16 FFT slices. However, dividing Fast Fourier-Transform (FFT) CC computation time by the number of regions for a kernel size of 50×50 gives an estimate that it takes about 2 ms to compute and normalize one region. When this estimate is multiplied by the number of regions for different kernel sizes, the resulting times are approximately the same as those described in Table 3.4. The discrepancy in time may be due to memory loading latency, as mentioned in [2].

3.4 Summary architectures

The presented algorithms and their simulated computation times are not trivial to compare, as they are tested on different platforms, with different parameters. This section will attempt to compare the algorithms and in what situations they may be suited.

The COMAC architecture excels at situations where the template is large compared to the reference, and is mathematically more efficient than the CAMAC architecture when $M < 2m$ [1]. The NCC and ZNCC implementations utilize normalization which makes them resilient to noise and local lighting variations. This normalization requires extra computation, but this can be implemented in parallel with little to no penalties [2].

The FFT approach is based on correlation, and is an alternative route to the same goal based on the correlation theorem [2]. This means that a normalization component would also have to be implemented for this algorithm, and could be computed in parallel.

The SAD algorithm is relatively simple with high potential for parallelism. It has no inherent normalization component, which might make it less resilient to noise compared to the normalized correlation approaches.

Looking at computation time, this highly depends on the level of parallelism. Figure 3.9 depicts the esimated number of clock cycles for the NCC, ZNCC and SAD algorithms. The SAD algorithm has been plotted for different values of W. The FFT algorithm has been excluded from this plot, as there is no estimation available for the number of clock cycles, as it is heavily dependent on the FFT implementation.

Figure 3.9 are estimations for a reference with dimensions 1024×1024 and square template sizes smaller than 512×512. For these combinations of reference and template, the estimations show that the CAMAC algorithm is more efficient than COMAC for all template sizes smaller than 512×512. With a MACC length of 256 as in [1], the SAD algorithm would need a minimum width of $W = 256$ in order to be quicker than CAMAC, which makes sense, as

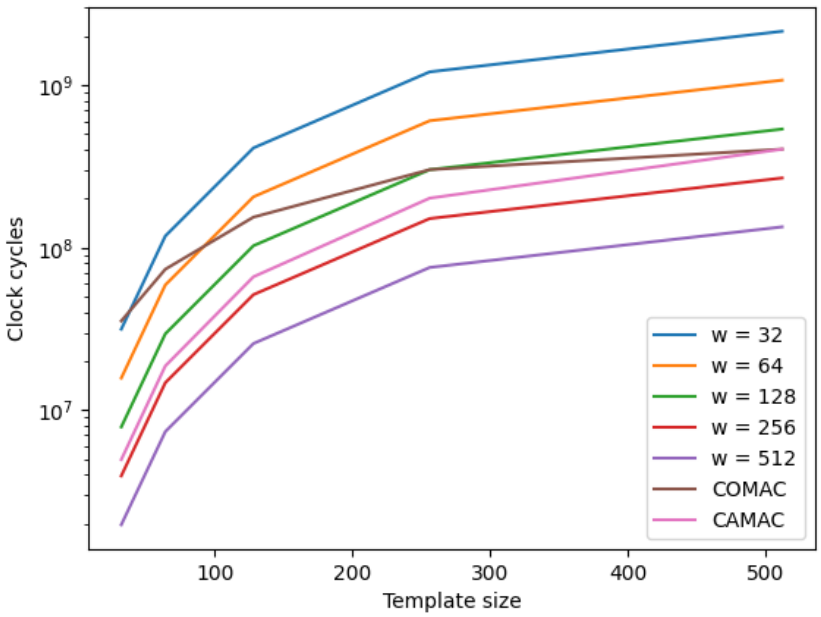


Figure 3.9: Estimated number of clock cycles for CAMAC and COMAC with length 256 and SAD with different widths W.

the amount of pixels to be calculated are the same. This means that with respect to speed, the amount of parallelism is essential, and can be achieved with all three algorithms. The FFT approach's benefit is that the computation time is constant with template size, as the algorithm is only dependent on the reference size and the amount of FFT slices which can be allocated [2]. However, the architecture presented in Section 3.3 requires multiple internal memories, which greatly reduces the maximum reference and template size compared to the other implementations.

Chapter 4

Implementation

In this chapter, a choice of similarity calculation will be made and it's implementation on a ZCU104 evaluation kit will be presented. First the use of a high level model will be presented, as well as a simple function for adding noise to speckle images. Then the original preliminary design is presented . Limitations of hardware and new information about design environment is then discussed, as this information forced changes to this original design. After this, the design which has been implemented will be described, along with the changes made because of the points made previously.

4.1 Choice of algorithm

The SAD algorithm has been chosen to be implemented. From the comparisons and discussions made in Section 3.4, the algorithm has a similar potential for efficiency as the correlation based algorithms. In addition is it a relatively simple algorithm in comparison. The simplicity was an important aspect, as it was thought to give a faster design time than the other more complex architectures.

4.2 High level model

A high level model i Python has been developed for this thesis. This is a matter of convenience and understanding. When testing a system in hardware, it is convenient of having a high level which can be used to compare results. One can use the high level model to verify that the system has been implemented correctly, i.e. it gives the same results as the high level model. In the case that the system and high level model outputs different results, one can use the use the high level model to help debug where the implemented system deviates. Also, simply implementing high level model gives the developer a better understanding of the algorithm and the system as a whole, which can be valuable.

In this thesis, the high level model will be used to confirm that the ultimate goal of template matching using SAD is possible. It will also be used to confirm Equation 3.6 about number of operations in SAD, which has been derived in this thesis. The high level model will also be used as a comparison for speed-up by running the model on the FPGA to see how much an accelerator would improve performance.

A simple noise function has also been created for the high level model, in order to study the accuracy of the algorithm in the cases where there is no exact match between the reference sub-image and the template. It is created as a Gaussian additive noise, as the shot noise described in Section 2.5 can be approximated as such for larger amounts of photons. There

are other types of image noise relevant in CMOS and CCD camera technology like *fixed pattern noise* and *Fano noise*, however this will not be accounted for. This may create less realistic images, but seeing as creating realistic models for image noise is not the goal of this thesis, a simplified model which only adds photon shot noise is used.

In Figure 4.1 one can see a speckle reference before and after the noise has been added.

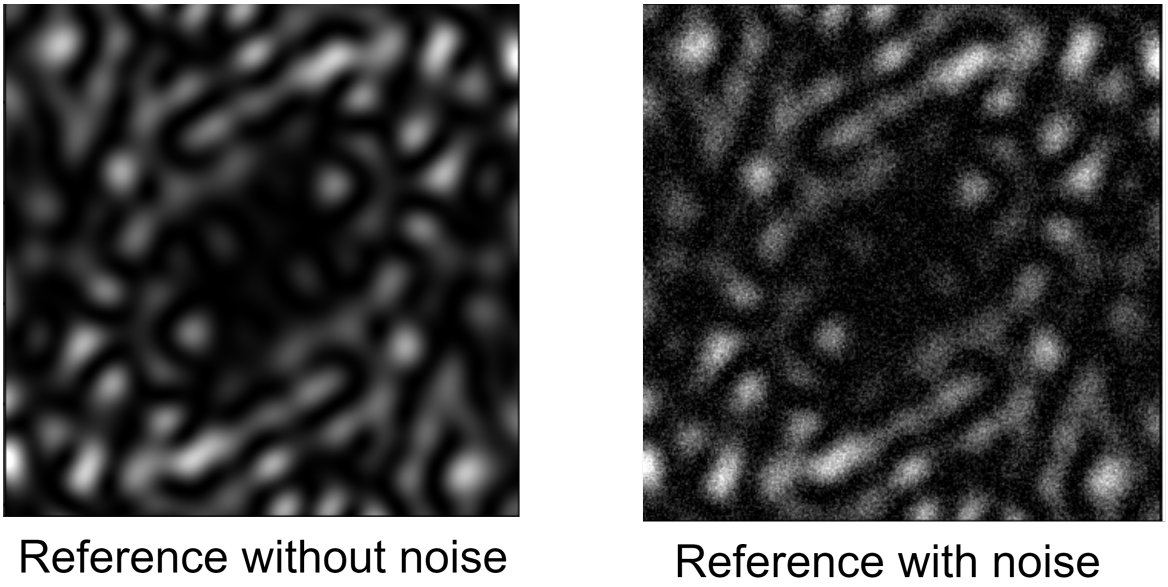


Figure 4.1: Speckle image with and without added noise.

4.3 System design

The SAD algorithm can be separated into two steps. Step one is calculating the absolute difference between the two inputs containing a number of reference and template pixels. Step two is adding all these differences together. Optimizing both the AD units as well as the summation-process can greatly shorten the critical path making the system able to run at much higher clock frequencies. Comparisons from [3] show that different implementations of Absolute Diffrence units and adders can have big impacts on maximum clock frequency as well as power usage. The comparison is also done for different input sizes ranging from 4×4 up to 32×32.

4.3.1 Original design

The first iteration of the SAD accelerator design can be seen in Figure 4.2. The reference and template are first stored in Block RAM (BRAM). The BRAM is a fully synchronous memory which can be read and written through independent read and write channels [23]. BRAM makes it possible to store larger amounts of memory on-chip, and allows the stored data to be accessed in one clock cycle. One BRAM block can store up to 36864 bits, meaning it can store 4608 8-bit pixel values. This means that a reference with dimensions 1024×1024 would require 234 36Kb blocks of BRAM. With a reference this large, it would still be enough available storage for a template with dimensions 512×512, since the ZCU104 development board has 312 36Kb BRAM blocks [9].

The template is first sent through the LogiCORE CIC compiler which decimates the image. When the template is decimated, the sub-images are passed through the CIC compiler

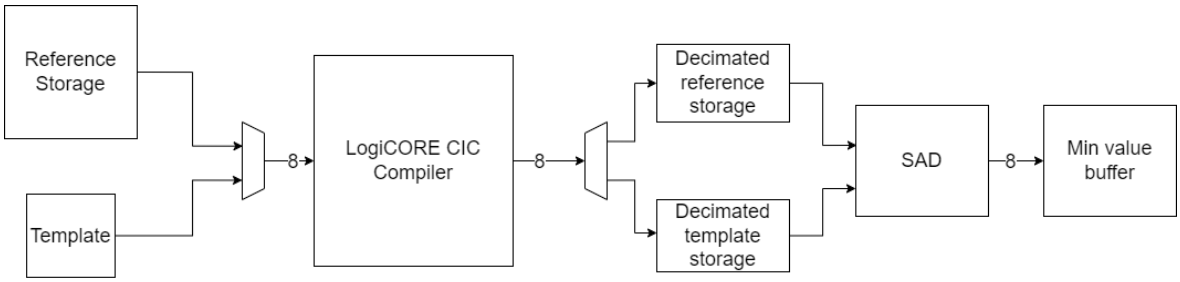


Figure 4.2: Block diagram of the system design in hardware.

and also stored close to the SAD block which calculates the new score between the current sub-image and the template. The subsequent sub-images are passed through one after the other until all sub-images have been compared. The best score and offset are then stored in the min value buffer to be sent to PS and can the be used further if desired.

4.3.2 Changes to the design

As the design implementation began, more information about the design environment and the given hardware was acquired, which meant that the design had to go through some changes.

The memory storage became a problem during the implementation of the original design. The ZCU104 development board has 312 BRAM blocks yielding a total storage of 1.4 MB, however it was not figured out how to utilize all these blocks. In the the Vivado development environment, one can use a BRAM controller IP which creates a BRAM generator. However using this method only generates 8 36Kb BRAM blocks, making it less ideal to use the BRAM as the storage for template and reference images. The interconnecting between the SAD accelerator and the BRAM would be too complex using this method as the the BRAM Controller communicates with the AXI protocol [24]. Thus another method for inferring and accessing this amount of BRAM would have to be found. This is further discussed in Section 6. Therefore it was decided to utilize the memory within the PS, and establish an AXI interface between this memory and the SAD accelerator. This is done by utilizing Direct Memory Access (DMA), which allows parts of the Programmable Logic (PL) to read and write from the processor's memory with less communication with the processor. Using a DMA IP from Xilinx, PS can store data in memory and start a transfer of the data to the accelerator through DMA. When the transfer is started, the processor is free to do other tasks. By importing the overlay of this system to a Jupyter Notebook in PYNQ, it is possible to create the reference and template image and store them directly in the memory ready to be transferred.

The bandwidth of the AXI interface between PS and the SAD accelerator is limited to 128 bits. In addition to the fact that the LogiCORE CIC compiler can at most receive 32-bit numbers, means that there is no reason to decimate the images. The point of decimation is to make the later calculations less computationally complex. In this system however the process would be a bottleneck. This is because the algorithm has to parse all pixel values through the SAD accelerator whether the CIC compiler is in the system or not, so by removing the CIC compiler on can parse 16 bytes at a time instead of 4. One clearly needs a $W = 16$ wide base of the SAD architecture to accommodate the bandwidth.

4.3.3 implemented SAD accelerator

The design updated with changes based on the discussion in Section 4.3.2 is depicted in Figure 4.3. An AXI Smartconnect is connected to the Zynq Processing System, on which the PYNQ framework is run. The channel between the AXI Smartconnect and PS communicates both ways, which should not create problems as data mostly flows to the accelerator, except for when the template matching process is finished and the best offset coordinates are written to memory . The PS communicates with the DMA IP through an AXI interface which is translated to an AXI Lite version which the DMA IP understands. This is how the PS initiates transfers of images as well as gets information of completed transfers.

The Zynq Ultrascale+ PS to PL communication channels support 128 bit wide data buses. This limits the system to transfer only 16 pixels at a time to PL. These 16 pixel values are read as memory mapped values in the PS memory, meaning the DMA IP knows the memory address where the images start as well as how much data is to be transferred. It then directly reads the memory, 16 bytes at a time. As the DMA receives the 16 bytes of pixel data, it translates the data from the AXI memory mapped protocol to an AXI Stream (AXIS) protocol, which the SAD Accelerator can receive. the AXI Stream protocol is a much simpler protocol with less signals and no memory address associated with the data that is transferred. The SAD Accelerator is therefore receiving a constant stream of pixel values and continuously calculating the SAD of the incoming pixel values.

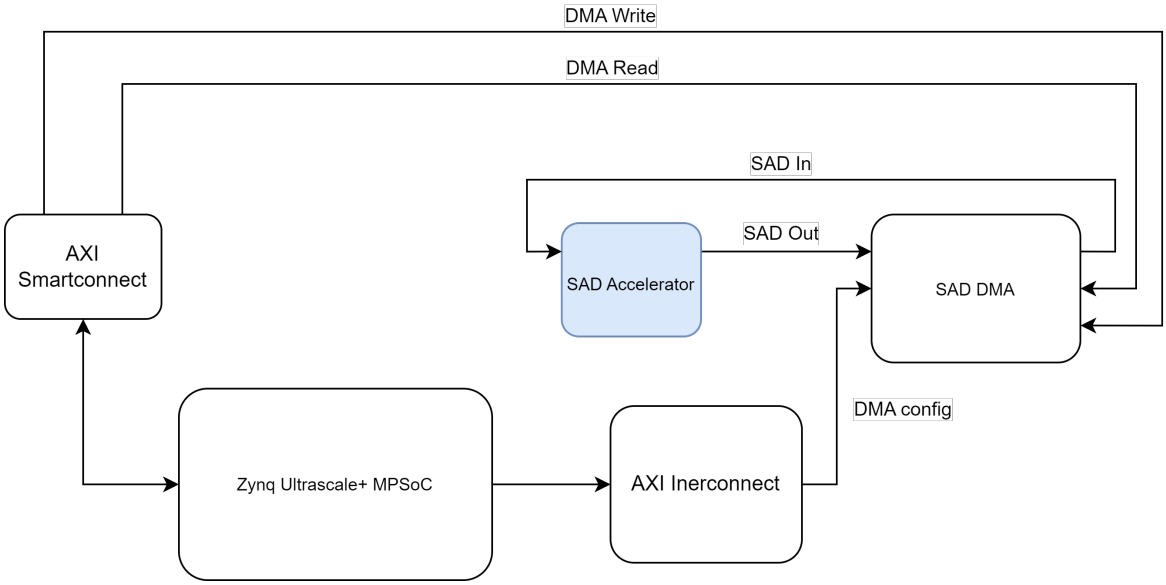


Figure 4.3: Block design of the implemented system with changes.

There has to be some considerations on how the reference sub-images and template is stored in the PS memory. In this system, there is no large memory close to the SAD to store a template. Therefore, the template and sub-image is transferred together from the memory to the accelerator. When the PS tells the DMA to initiate a transfer of data from memory to accelerator, the pixel values have to be stored in such a way that the 16 bytes that are read are the correct pixels to be compared. Therefore, a memory organization must be made, which can be seen in Figure 4.4. In this organization, one block of reference pixels and one reference of template pixels are transferred to the accelerator each clock cycle. The subscript of each block in the figure represents which clock cycle it is transferred. $m \times n$ is the template dimensions and W is the width of the SAD unit as well as the bus width, i.e.

the amount of pixel values which can be transferred at the same time. Each block contains $W/2$ pixels. In the implementation, $W = 8$ since this is the most amount of pixels that the communication bus between PS and PL has bandwidth for. Therefore each colored block in Figure 4.4 contains 8 pixels of either the reference image or template image.

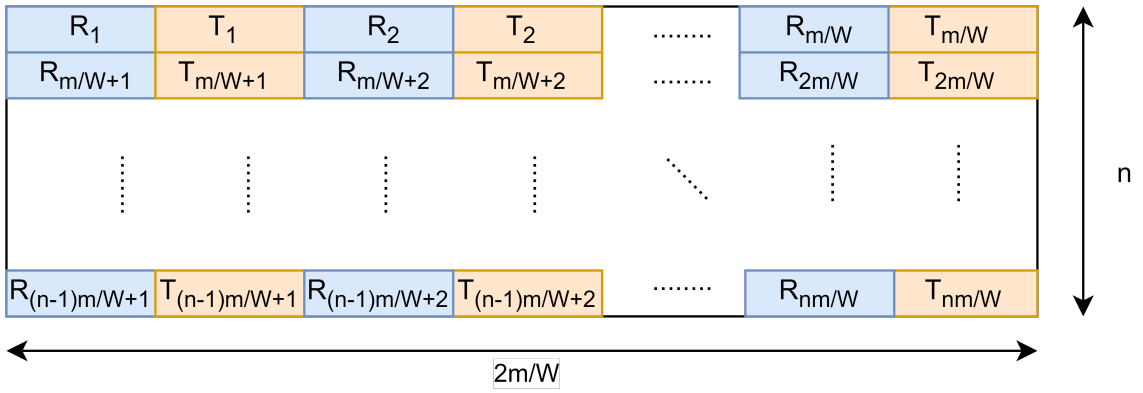


Figure 4.4: Memory organization in PS RAM, which is transferred to SAD accelerator through DMA.

Figure 4.5 shows a block diagram of the implemented system design in hardware. The system receives pixel data through the AXIS (AXI Stream) slave interface [24]. When pixel values are received, the AXIS ready and AXIS valid signals are both high, which indicates to the control path (in green) that pixel values are received. The control path has three counters which are enabled by the reception of pixel values. Index counter counts the number of operations done of the comparison of the template and one sub image. The amount of operations are dependent on the dimensions of the template, as well as the bandwidth of the communication bus between PS and PL, i.e. the width W of the SAD unit as shown in Figure 3.5.

$$\text{Index count top} = \frac{m \cdot n}{W} \tag{4.1}$$

Equation 4.1 is the value that the index counter counts towards, where $m \times n$ is the template dimensions and W is the number of pixels computed in parallel. After the index counter reaches the top value, it signals to increment the x-offset, preparing the system for the next sub image. The preparation includes signaling the score and offset update unit (in blue) to compare the score currently output from the SAD unit (in red) with the score it has stored as the best score so far. If the new score is better than current best score, the best offset registers are updated with the current offsets and the best score register is updated with the new score When x is incremented it is compared with the known top value of the counter equal to $M - m$. If the comparison is true, the x-counter is reset to zero and the y-counter is signaled to increment. When the counters reach $x = M - m$ and $y = N - n$, the control path signals the AXIS Master interface to start a transmission. The transmission packet consists of the two coordinates of the best offsets, and is transmitted to the processing system.

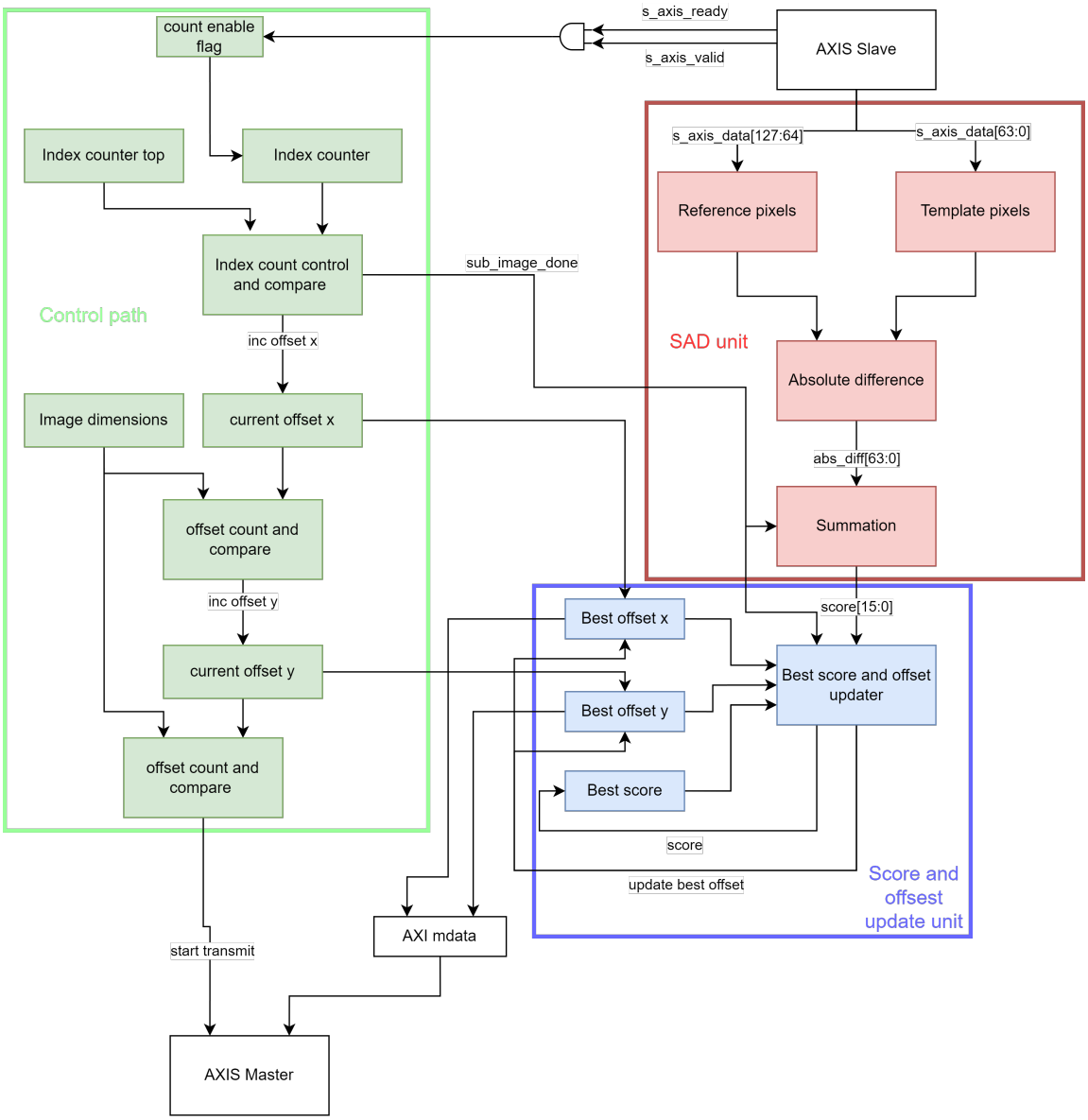


Figure 4.5: Block diagram of the system in hardware.

4.3.4 On the ZCU104 development board

The design is managed in Xilinx' Vivado Design Suite 2022.1, and has been synthesized and implemented with Vivado's built in tools. An overlay is created which can then be loaded onto the FPGA using PYNQ, which allows us to access the hardware components using a high level, object oriented way. Thus, the reference images can be created and the template can be extracted by the processing system before it is placed in memory and DMA transfers are initiated.

The system was unfortunately not implemented on the FPGA correctly, and a complete SAD template matching process could not be completed. The parts of the system which was operational was the placement of pixel data in memory and the initiation of data transfer through AXI DMA. We can use this to at least estimate the systems efficiency in an operational state. The process which needs characterization in order to do this is the

storage of memory in PS RAM. The implementation of the system requires that the template and sub-image are interleaved in the memory in order for the data to be input correctly to the accelerator. This must be done for every sub-image in the reference. The time it takes to store and transfer all the images, would be the time the system takes to perform the template matching process.

Chapter 5

Results

In this chapter the various results from the implementation will be presented. This includes the resource usage of the implemented design for various configurations of template sizes as well as power usage. It was described in Section 4.3.4 that an operational system was not realized on the FPGA. In this chapter an attempt at estimating the effectiveness of the proposed system will be made nonetheless, and compared to the results from simulation.

5.1 High level model

Template matching on a variety of reference and template sizes was tested using the high level model.

The first goal of the high level model as stated in Section 4.2, was to verify that template matching using the algorithm is possible. When no noise is present in either reference nor template image, the algorithm has a 100 % accuracy rate in the high level experiments. This makes sense, as without any noise the SAD output value of the template and the correct sub-image will always be zero, as the best match is an identical image. This is of course assuming all sub-images in the reference are unique.

More interestingly is how the algorithm handles noise. A simple Gaussian noise model was implemented in order to test how the algorithm is affected by differences between reference and template. As described in Section 2.5, a Gaussian distribution can be used to approximate image shot noise for larger amounts of noise.

The testing of the high level model was done by creating a reference image for a given reference size, and creating a copy with added noise. Then, templates were extracted from the reference without noise and attempted to be located within the noisy reference. For every combination of reference and template, 50 iterations were done. The results of these tests can be seen in Table 5.1.

Table 5.1: Table showing results of high level model of template matching using SAD on reference images with added noise. Hit rate is percentage of correctly located templates. Maximum error is the maximum error in either x or y direction for the worst test, in number of pixels off from the correct answer. The sample size is 50 tests for each configuration.

Reference dimension	Template dimension	Hit rate [%]	Maximum error
256	32	54.0	191
	64	94.0	1
	128	100.0	0
512	32	2.0	445
	64	20.0	245
	128	88.0	1
	256	100.0	0
1024	32	0.0	911
	64	8.0	851
	128	32.0	2
	256	82.0	1
	512	100.0	0

5.2 Resource usage

In Table 5.2, the resource usage for the synthesized design for different configurations are presented. It is implicit that the template dimension has to be smaller than the reference dimension, as this is the definition of the template matching process. Thus, there are more possible configurations for larger reference sizes. We see that increasing the reference size does not have a large effect on the resource usage. The use of LUTs varies somewhat with reference and template size, while the use of Flip Flops is only dependent on template size. Even though the usage is not constant for the different configurations, they are all relatively close. The amount of Flip Flops increases with template size, but the system uses at most 259.

The HDL wrapper including the AXI DMA IP, AXI Inerconnects and Zynq Processing system had a constant resource usage with 4231 LUTs, 7234 Flip Flops and 3.5 BRAMs.

Table 5.2: Table of the used resources for different configurations of reference and template. The utilization is only for the SAD accelerator, excluding the Zynq processor, AXI Interconnects and the DMA.

Reference dimension	Template dimension	LUTs	Flip Flops
256	32	293	255
	64	294	257
	128	304	259
512	32	293	255
	64	294	257
	128	297	259
	256	304	259
1024	32	309	255
	64	307	257
	128	297	259
	256	297	259
	512	297	259

5.3 Simulation results

Simulations were done for smaller reference and template sizes first, in order to verify operation of the accelerator and confirm Equation 3.6. Both objectives were done, and the accelerator operated as desired and using the amount of clock cycles estimated by the equation. The simulation was done in Vivado Design Suite using the built in simulator, which crashed for larger reference sizes. Therefore, the results presented are shown in Table 5.3, which are theoretical results based on Equation 3.6 shown in column four. In column three of Table 5.3 are execution times for a high level model running on the PYNQ framework on the FPGA. The estimations are done with $W = 8$ pixels calculated in parallel, as this was the bandwidth of the bus between the accelerator and the PS RAM.

We see notable speed-up already with only few pixels calculated in parallel, with the largest speed-ups for small template sizes. Of course, the estimations do not account for memory latency, and assumes that the pixel data is a continuous stream of data. The data input follows the memory organization described in Section 4.3.4.

Table 5.3: Execution times for high level model in PYNQ Jupyter Notebook on the FPGA in third column, and the theoretical execution time of the SAD accelerator in fourth column. Theoretical executions are estimated for $W = 8$.

Reference dimension	Template dimension	High level [s]	Theoretical exec. time SAD accelerator [s]	Speed-up
256	32	4.6	0.04	115.0
	64	4.9	0.10	49.0
	128	4.9	0.19	25.79
512	32	21	0.16	131.25
	64	26	0.57	45.61
	128	45	1.68	26.79
	256	87	2.98	29.19
1024	32	89	0.70	127.14
	64	122	2.6	46.92
	128	279	9.13	30.56
	256	784	26.84	29.21
	512	1291	47.72	27.05

5.4 Experimental results

The system was not implemented correctly in hardware, and there are therefore no complete experimental results to present. However, we can based on other tests estimate how the system would perform in an experimental setup, as described in Section 4.3.4. We can time time how long the PS takes to place and transfer different sizes of sub-image and template. We can then compare this with the time for the high level model and simulation times. Table 5.4 show the measured time for the PS to store and initiate DMA transfer of sub-images and template images for different dimensions. The storage time of one template and sub-image relies only on the template dimensions. The complete template matching process is dependent on the amount of sub-images to compute, which change based on the relationship between the reference and template dimensions.

Table 5.4: Time to store sub-image and template in PS RAM as shown in Figure 4.4

Template dimension	Storage time on template and sub-image [s]
32	0.026
64	0.109
128	0.429
256	1.719
512	6.863

Similarly to Equation 3.5, we can find the number of sub-images based on the reference and template dimensions. The number of sub-images are $(M-m)(N-n)$, where the reference dimensions are $M \times N$ and the template dimensions are $m \times n$. If we multiply the number of sub-images with the corresponding time for storing the sub-images in Table 5.4 we find an estimate for how fast the system would be able to perform a template matching process. Considering that the number of sub-images are in the order of thousands, even the shortest storage time would be way too slow. Even a reference as small as 64×64 with a template of 32×32 would use 26 s to complete. This is a reduction in speed of five times compared

to the high level model running on the FPGA for a reference size 16 times as large, which is obviously very inefficient.

Chapter 6

Discussions and future work

In this chapter, some thoughts on the systems complexity and development time will be given. Then, the focus will shift towards discussing the findings from Chapter 5. The results regarding resource usage and performance will be evaluated and it will be discussed why the results turned out the way they did. The performance results will be compared to the alternative of a high level model, and some areas of improvement will be provided. Lastly, avenues for further investigation and development will be provided as future work.

6.1 Development

The design is a relatively simple system to implement. The control logic consists of three counters each of which signals a few different parts of the system. The communication with PS is done through the AXI protocol which is a relatively complex protocol and can have a long implementation time. However, the SAD Accelerator communicates through the much simpler AXI Stream protocol which is translated to AXI by Xilinx IP's, meaning only the AXI Stream protocol needs to be implemented by the developer.

The memory handling was the most challenging part during the implementation. The system is designed for a constant stream of pixel data in order to be the most efficient. A bottle neck in data flow is therefore detrimental to performance. This is reflected in the estimations based on the PYNQ tests presented in Section 5.4 where the writing of data to memory before transfer through DMA makes the system many times slower than the theory suggests. If the developer knows how to utilize the other memory resources that the FPGA has to offer, the development time should be short and performance greatly increased.

6.2 Accuracy

The high level model shows that it is possible to use SAD to perform template matching. The algorithm is however susceptible to noise, especially for small template sizes. For larger template and reference sizes, the high level model performed well even with the noise. The results show that for the larger template sizes, the algorithm is relatively accurate. There are few errors, and these few errors are small. For the small template sizes, the amount of noise is too great, and the errors are many and not insignificant. This implies that some form of normalizing element is needed for noisy conditions and when using small template sizes.

6.3 Resource usage

The system is not greedy on resources, with highest amount of each resource utiilized being only 309 LUTs and 259 Flip Flops, yielding a total usage of 4540 LUTs, 7493 Flip Flops and 3.5 BRAMs including the HDL wrapper. There is also little variation in resource usage between different template sizes. This is a system that could be implemented on much smaller FPGAs than the ZCU104, with no restriction on template and reference size. The template and reference size is however restricted to memory capacity, where the ZCU104 has 1.4 MB of BRAM which is enough for 1024×1024 reference and a 512×512 template. An FPGA with less BRAM would have to utilize other types of memory like external RAM in order to operate on images of this size.

6.4 Performance

The use of the memory organization described in Section 4.3.3 is a huge bottleneck and was a poor design choice. The choice limits the SAD Accelerator to a width of only 8 pixels in parallel, since the 16 byte communication bus transfers 8 template pixels and 8 reference pixels. In addition to this, the PS cannot interleave and transfer the data fast enough. The accelerator ends up spending most of its time waiting for new data. The resulting system is therefore significantly slower than using the high level model on PS using PYNQ. In order to get adequate performance and a speed-up relative to the high level model, the memory organization must be improved.

The interleaving of the template and sub-image is the bulk of the memory preparing process for the PS. Thus, for example only storing the template in BRAM would remove the interleaving part and should contribute to increasing the efficiency greatly. The reference could also be stored in BRAM close to the accelerator, however when references get big, smaller FPGAs might not have enough resources. One could then either use external RAM or load a partition of the reference into BRAM. The partitions would have to be bigger than a single sub-image in order for this to be more efficient.

The simulation results agree with the theoretical efficiency presented in Section 3.2.1. From Table 5.3 one can see that if a memory organization capable of supplying a continuous pixel stream is implemented, the accelerator has a significant speed-up from alternative high level method. If this memory organization also has a higher bandwidth, the SAD Accelerator could have an increased width W, which would proportionally decrease the number of clock cycles as stated in Equation 3.6

It is also interesting to note that the highest speed up are for smaller template sizes, so a trade-off for speed against accuracy is relevant.

The maximum clock frequency is estimated by the implementation tool in Vivado, and varies for the different implementations. The implementations had an average maximum frequency of around 180 MHz. The maximum frequency should however be expected to change for implementations with higher SAD width W, as the critical path through the adders becomes longer. The frequency can be improved through pipelining, which can easily be done in the SAD architecture by placing intermediary registers between each layer of adders.

6.5 Further work

The most important avenue for further work is to create a system that outperforms the alternative high level model, since there is no reason to have a hardware accelerator if it does

not improve performance. In order to do this, the memory organization must be improved, as this is the main bottleneck in the system as discussed in Section 6.4. This can be done by using different kinds of memory on the FPGA which has lower read latency and higher bandwidth. The system then has fewer memory accesses and performance is drastically increased.

Improvements to the algorithm in order to increase accuracy in noisy conditions should also be done. One can for example study the other algorithms presented in Section 2. A normalizing element can be implemented in parallel with the similarity calculating unit with insignificant penalties [2].

Further work also consists of optimizing the algorithm in order to increase the efficiency and decrease power consumption. This can be done as described in Section 3.6, with partial sums and improving adder-architecture.

Chapter 7

Conclusions

The objectives of this thesis was to implement a template matching algorithm on an FPGA which accelerated the process relative to an alternative high level model. The system which has been designed was not operational on the FPGA, and estimations of the efficiency if it was operational was worse than that of the high level alternative. This is because of a poor design choice in regards to memory organization. The interleaving of pixel values, as well as low bandwidth are the main problems of the organization. If an organization is implemented which improves these areas and can supply the design with a continuous stream of pixel data, the system has significant speed-up relative to the high level alternative, with higher speed-ups for smaller template sizes. The speed-ups range from around 25 times to around 130 times compared to a high level model. The speed-up should increase proportionally with the amount of pixels computed in parallel. Other optimizations are also available which can further increase acceleration relative to the high level model.

Accuracy of the algorithm was studied using a high level model. For template images with a width of half the width of the reference, the algorithm has a 100 % accuracy. If the template width is one-quarter of the reference, the accuracy decreases to between 82 % to 94 %, with small errors. The maximum errors for this template and reference size was 1 pixel from the correct location. For template widths smaller than one-quarter reference width, the accuracy is further decreased. For a template width of an eighth of the reference, the highest accuracy was 54 %, with the highest error being 191 pixels off from the correct location in a 256 pixel wide reference.

Appendix A

Speckle image script

Listing A.1: speckleScript.m

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

%% Ref: Katrin Philipp?? Technische Universitat 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; % padding
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
```

Bibliography

[1] Q. Zhou, L. Yang, and H. Cao, "A Configurable Circuit for Cross-Correlation in Real-Time Image Matching," *Journal of Computer Science and Technology*, vol. 32, no. 6, pp. 1305–1318, Nov. 2017. [Online]. Available: http://link.springer.com/10.1007/s11390-017-1765-4

[2] A. Lindoso and L. Entrena, "High performance FPGA-based image correlation," *Journal of Real-Time Image Processing*, vol. 2, no. 4, pp. 223–233, Dec. 2007. [Online]. Available: http://link.springer.com/10.1007/s11554-007-0066-5

[3] W. Ahmad and I. Hamzaoglu, "An Efficient Approximate Sum of Absolute Differences Hardware for FPGAs," in *2021 IEEE International Conference on Consumer Electronics (ICCE)*, Las Vegas, NV, USA, Jan. 2021, pp. 1–5, iSSN: 2158-4001.

[4] P. Johri, D. M.J., R. Khanam, M. Marciszack, and A. Will, *Trends and Advancements of Image Processing and Its Applications*, 1st ed., ser. EAI/ Springer Innavations in Communication and Computing. Springer Cham, 2022.

[5] J. P. Lewis, "Fast Normalized Cross-Correlation," *Ind. Light & Magic*, 10 2001.

[6] D. K. Karna, S. Agarwal, and S. Nikam, "Normalized cross-correlation based fingerprint matching," in *2008 Fifth International Conference on Computer Graphics, Imaging and Visualisation*, 2008, pp. 229–232.

[7] B. Pan, K. Qian, H. Xie, and A. Asundi, "Two-dimensional digital image correlation for in-plane displacement and strain measurement: a review," *Measurement Science and Technology*, vol. 20, no. 6, p. 062001, Jun. 2009. [Online]. Available: https://iopscience.iop.org/article/10.1088/0957-0233/20/6/062001

[8] G. Slettemoen, Personal communication, 2022.

[9] *"ZCU104 Board User Guide"*, Xilinx, 10 2018, "UG1267".

[10] L. Di Stefano, S. Mattoccia, and M. Mola, "An efficient algorithm for exhaustive template matching based on normalized cross correlation," in *12th International Conference on Image Analysis and Processing, 2003.Proceedings.*, Mantova, Italy, 2003, pp. 322–327.

[11] E. Fykse, "Performance Comparison of GPU, DSP and FPGA implementations of image processing and computer vision algorithms in embedded systems," p. 76, 2013, NTNU Master's thesis.

[12] J. G. Proakis and D. G. Manolakis, *Digital signal processing - principles, algorithms and applications*, 4th ed. Massachusetts: Pearson, April 2006, ISBN 0131873741.

[13] MathWorks, “Normalized 2-d cross-correlation,” https://se.mathworks.com/help/images/ref/normxcorr2.html, accessed: 05/06-2023.

[14] B. Friemel, L. Bohs, and G. Trahey, “Relative performance of two-dimensional speckle-tracking techniques: normalized correlation, non-normalized correlation and sum-absolute-difference,” in *1995 IEEE Ultrasonics Symposium. Proceedings. An International Symposium*, vol. 2, Nov. 1995, pp. 1481–1484 vol.2, iSSN: 1051-0117.

[15] H. Niitsuma and T. Maruyama, “Sum of Absolute Difference Implementations for Image Processing on FPGAs,” Milan, Italy, Aug. 2010, pp. 167–170, iSSN: 1946-1488.

[16] K. R. Rao, D. N. Kim, and J. J. Hwang, “Two-Dimensional Discrete Fourier Transform,” in *Fast Fourier Transform - Algorithms and Applications*. Dordrecht: Springer Netherlands, 2010. [Online]. Available: https://doi.org/10.1007/978-1-4020-6629-0_5

[17] D. D. Duncan, S. J. Kirkpatrick, and R. K. Wang, “Statistics of local speckle contrast,” *Journal of the Optical Society of America A*, vol. 25, no. 1, p. 9, Jan. 2008. [Online]. Available: https://opg.optica.org/abstract.cfm?URI=josaa-25-1-9

[18] J. R. Janesick, *Photon transfer*. Washington: SPIE–The International Society for Optical Engineering, august 2007, ISBN 978-0-8194-6722-5.

[19] AMD, “Pynq introduction,” https://pynq.readthedocs.io/en/latest/, accessed: 05/06-2023.

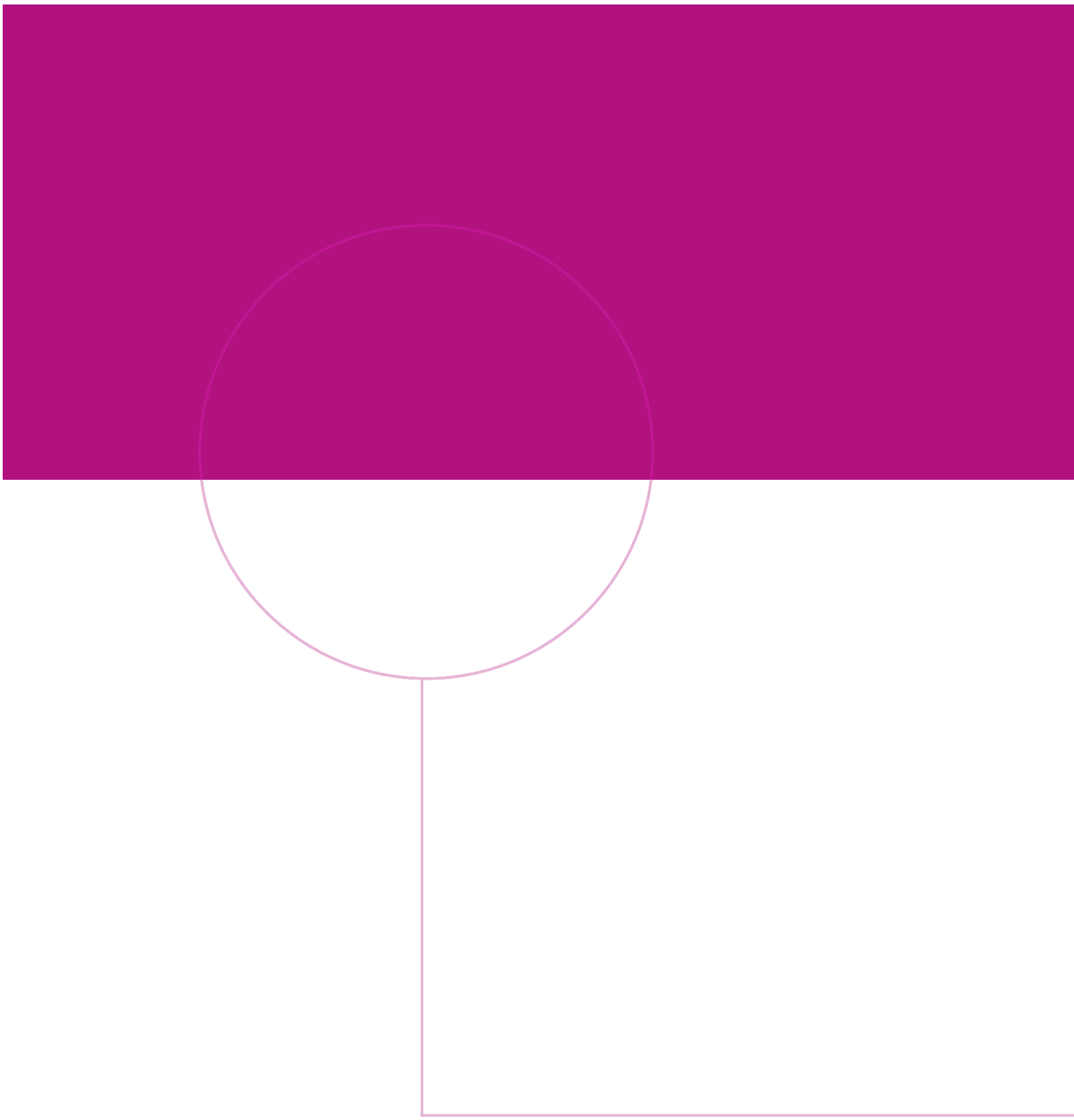
[20] S. Wong, B. Stougie, and S. Cotofana, “Alternatives in FPGA-based SAD implementations,” in *2002 IEEE International Conference on Field-Programmable Technology, 2002. (FPT). Proceedings.*, Hong Kong, China, Dec. 2002, pp. 449–452.

[21] T. Adiono, M. D. Adhinata, N. Prihatiningrum, R. Disastra, R. V. W. Putra, and A. H. Salman, “An architecture design of SAD based template matching for fast queue counter in FPGA,” in *2016 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS)*, Oct. 2016, pp. 1–4.

[22] N. N. Shah, K. R. Agarwal, and H. M. Singapuri, “Implementation of sum of absolute difference using optimized partial summation term reduction,” in *2013 International Conference on Advanced Electronic Systems (ICAES)*, 2013, pp. 192–196.

[23] *"UltraScale Architecture Memory Resources"*, Xilinx, 9 2021, "UG573".

[24] ARM, “AMBA AXI and ACE Protocol Specification AXI3, AXI4, and AXI4-Lite ACE and ACE-Lite,” June 2004, issue D.



NTNU

Norwegian University of
Science and Technology