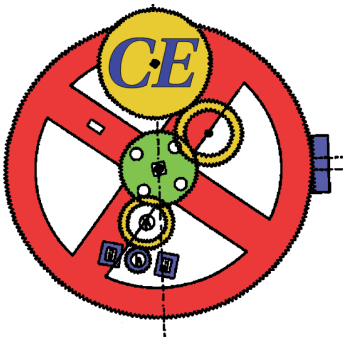


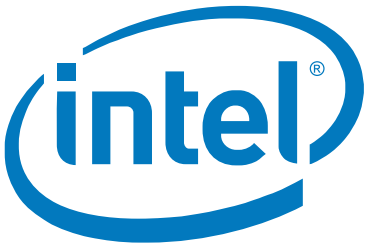
Computer Engineering
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ce.et.tudelft.nl/

2015

MSc THESIS

Flexible hardware accelerator for 2D Convolution and Census Transform

Jeffrey van den Bor



CE-MS-2015-08

Abstract

Image Processing is an emerging field: every year new applications are introduced, and these are pushing the hardware requirements. This thesis looks at the design of a hardware accelerator to accelerate several filters used in Image Processing: 2D Convolution, Census Transform, and Local Binary Patterns. At Intel, these filters are used for Convolutional Neural Networks, Gaussian Blur, Stereo Vision, and Face Detection applications. The new hardware accelerator is based on an existing Intel accelerator (the Block Matching and Bilateral Filter accelerator). The new accelerator reuses some components from this accelerator such as multipliers, adder trees, and subtraction units. This allows for a considerable reduction of the area overhead. Furthermore, the new accelerator is more flexible than the existing one because it accelerates both the new filters and the original filters and has a variable window size of 5x5, 7x7 and 11x11 that is realized by combining the results of the smallest window together. In order to analyze the performance of the accelerator implemented in this work, we compare the new accelerator with the original one in terms of speed, processor utilization, throughput, and area. We demonstrate that the average speedup for the 2D Convolution and the 5x5 Census Transform is 1,57x and 1,50x respectively. Note that higher speedups are possible when using multiple instances of the new accelerator; i.e. the maximum speedup for the 2D Convolution is 7,86x, and the maximum speedup for the Census Transform is 4,50x. In addition, the processor utilization is lowered by 12,61x on average for the 2D Convolution and 4,00x for the 5x5 Census Transform. This improvement allows the processor to perform other operations in the background or to reduce the dynamic power consumption. Throughput is obtained by considering real-time video processing. The 2D Convolution is capable of processing 4K video, and the 5x5 Census Transform can handle 8K videos. Finally, the area of the new accelerator increases with 42% compared to the original accelerator, but at system level it results in a total area increase of only 6%. Then, varying the number of accelerators provides a trade-off between speedup, processor utilization, and area usage.

TU Delft
Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

Flexible hardware accelerator for 2D Convolution and Census Transform

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Jeffrey van den Bor
born in Voorburg, Netherlands

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Flexible hardware accelerator for 2D Convolution and Census Transform

by Jeffrey van den Bor

Abstract

Image Processing is an emerging field: every year new applications are introduced, and these are pushing the hardware requirements. This thesis looks at the design of a hardware accelerator to accelerate several filters used in Image Processing: 2D Convolution, Census Transform, and Local Binary Patterns. At Intel, these filters are used for Convolutional Neural Networks, Gaussian Blur, Stereo Vision, and Face Detection applications. The new hardware accelerator is based on an existing Intel accelerator (the Block Matching and Bilateral Filter accelerator). The new accelerator reuses some components from this accelerator such as multipliers, adder trees, and subtraction units. This allows for a considerable reduction of the area overhead. Furthermore, the new accelerator is more flexible than the existing one because it accelerates both the new filters and the original filters and has a variable window size of 5x5, 7x7 and 11x11 that is realized by combining the results of the smallest window together. In order to analyze the performance of the accelerator implemented in this work, we compare the new accelerator with the original one in terms of speed, processor utilization, throughput, and area. We demonstrate that the average speedup for the 2D Convolution and the 5x5 Census Transform is 1,57x and 1,50x respectively. Note that higher speedups are possible when using multiple instances of the new accelerator; i.e. the maximum speedup for the 2D Convolution is 7,86x, and the maximum speedup for the Census Transform is 4,50x. In addition, the processor utilization is lowered by 12,61x on average for the 2D Convolution and 4,00x for the 5x5 Census Transform. This improvement allows the processor to perform other operations in the background or to reduce the dynamic power consumption. Throughput is obtained by considering real-time video processing. The 2D Convolution is capable of processing 4K video, and the 5x5 Census Transform can handle 8K videos. Finally, the area of the new accelerator increases with 42% compared to the original accelerator, but at system level it results in a total area increase of only 6%. Then, varying the number of accelerators provides a trade-off between speedup, processor utilization, and area usage.

Laboratory : Computer Engineering
Codenumber : CE-MS-2015-08

Committee Members :

Chairperson and Advisor:	Prof. dr. K.L.M. Bertels, CE, TU Delft
Member:	Assistant prof. dr. ir. Z. Al-Ars, CE, TU Delft
Member:	E. van Daalen, Intel Corporation
Member:	Associate Prof. dr.ir. R. van Leuken, CAS, TU Delft

Dedicated to my family and friends

Contents

List of Figures

List of Tables

List of Acronyms

1080p Full-HD video with resolution of 1920x1080 pixels
4K Ultra HD video with resolution of 4096x2160 pixels
8K Full Ultra HD video with resolution of 7680x4320 pixels
ANR Advanced Noise Reduction
ASIC Application-Specific Integrated Circuit
BFA Bilateral Filter Accelerator
BMA Block Matching Accelerator
CHDL Configurable Hardware Description Language
CNN Convolutional Neural Network
CT Census Transform
DMA Direct Memory Access
FPGA Field-Programmable Gate Array
FPS Frames per Second
FU Functional Unit
GPGPU General-Purpose computing on Graphics Processing Units
GPU Graphics Processing Unit
ICG Imaging and Camera Technologies Group
IPU Image Processing Unit
ISP Image Signal Processor
LBP Local Binary Patterns
LUT Look-Up Table
MCT Modified Census Transform
mLBP modified Local Binary Patterns
SAD Sum of Absolute Differences
SNR Signal-to-Noise Ratio
SUBABS Subtraction followed by Absolute Value
TIM The Incredible Machine
VLIW Very Long Instruction Word

Acknowledgements

I would like to thank a number of people for supporting me during this project. First of all, I would like to thank Prof. Dr. Koen Bertels for providing the opportunity to work on this project at Intel and for providing guidance and advice during the project. Secondly, I want to thank Dr. Ir. Carmina G. Almudéver for her advice, her support in writing the thesis, and for reading my thesis multiple times.

I would also like to thank several people who I met at Intel. Thank you Frank Bouwens for providing information about the current accelerator, explaining the approach, supporting me whenever I encountered problems, and for reading my thesis. Edwin van Dalen, thank you for assisting me on the project, providing the necessary information about the processor and giving feedback on my thesis. Also I would like to thank Rik-Jan Zwartenkot for having regular discussions and giving me more insight in the company. Further acknowledgements go to my friends and colleagues Dan and Rahul for sharing the office and the conversations we had every day.

A special thank you goes to my family, for providing support and advice at good times and hard times.

Jeffrey van den Bor
Delft, The Netherlands
September 4, 2015

1 Introduction

Image processing is an emerging field, with many new interesting applications. Some of these applications are shown in Figure 1.1. The cameras in smartphones and tablets are increasingly being used to capture the memorable moments of our lives. The quality is becoming more important because people are less likely to buy a device that is known to shoots bad pictures. Additionally, the megapixel race is still ongoing. Every new device gets more megapixels compared to the previous generation. Currently, it is still unclear how long the trend of increasing megapixels will last. Furthermore, there is an increase in the number of frames per second; this is mostly used for capturing slow-motion video. Most devices already have two cameras (front and back), but now there is also the trend of 3D (stereo) imaging which requires two side-by-side cameras.

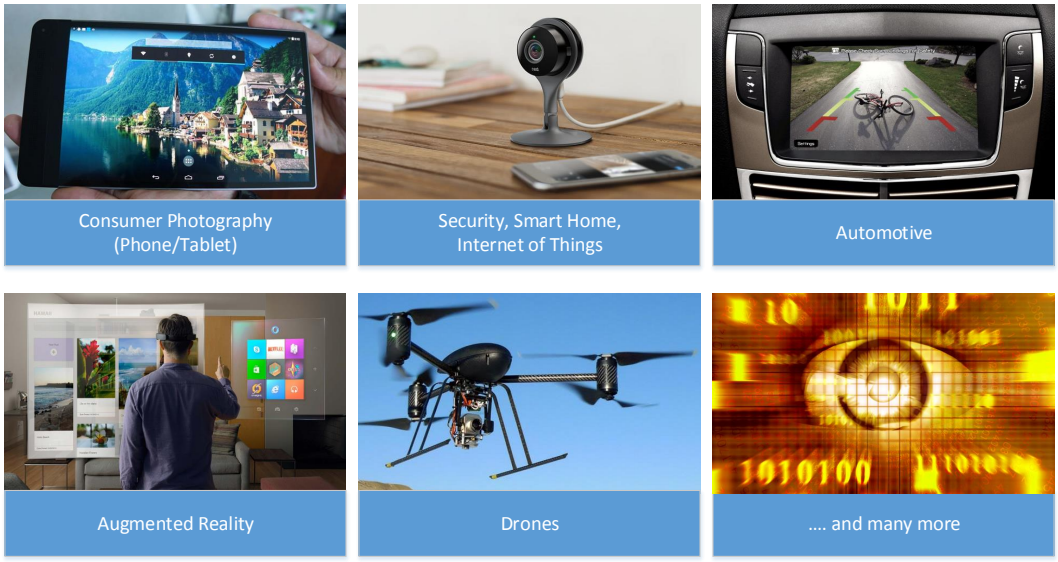


Figure 1.1: Many different applications for Image Processing.

The second application is to use cameras inside homes and buildings, for security purposes (camera connected to the cloud) and Internet of Things applications. Automotive is also an application where cameras are increasingly used. It started with rear-view cameras to assist the user in parking. Now dashboard cameras are also becoming more popular to help capture evidence in the case of an accident. Luxury cars already use cameras for accident prevention and assisting the driver to stay in the lane. For autonomous vehicles, it is even more important to use cameras since there is no human interaction involved. The next application is augmented reality, where digital elements are virtually projected on the real world. For example, it is possible to project a big

TV-screen on the empty wall. In this case, cameras are required to detect the locations of where to put the digital content (e.g. empty walls). Another application is drones, to capture videos and photos from another perspective. Finally, there are potentially many more applications coming in the next few years!

All these developments require better algorithms and an increase in computing capacity. This increase in computing capacity is why accelerators are becoming more important. Custom accelerators allow for faster processing by assisting the main processor with specific operations. This can lead to faster execution time, and it can lead to energy saving because the accelerator can be optimized for that purpose.

1.1 Intel Imaging and Camera Technologies Group (ICG)

The Imaging and Camera Technologies Group (ICG) develops complete solutions for mobile imaging and video applications. These solutions consist of both fixed and flexible parts. Fixed functions are used for algorithms that are considered a standard in the industry and can be deeply optimized for power and performance. Flexibility, on the other hand, is required for new and future image processing applications. Important here is that the hardware is produced years before release, and this flexibility allows for additions and updates to the algorithm later on.

Image processing is extremely compute demanding, and it is bandwidth and memory intensive. The required computing power is increasing from hundreds of GOPS (Giga-Operations Per Second) towards TeraOPS. It requires a bandwidth of tens of Gigabytes per second which is also increasing, and memory with hundreds of Megabytes going towards Gigabytes. It is important to have hardware acceleration to handle the increase in computing capacity. A hardware accelerator offers high performance and throughput while saving power and area. In some cases, the accelerator might not be able to process the data any faster, but it still offloads the main processor so it can process other tasks instead.

1.2 Thesis Topic

Design of a hardware accelerator to accelerate 2D Convolution and the Census Transform.

Image Processing consists of many steps; first the sensor captures the light and converts it into a digital signal. The captured image then has to be processed, including several filters to enhance the quality and many other tasks like the recognition of faces. This thesis looks at the construction of a hardware accelerator for new filter functions. The main focus of this thesis will be on accelerating the 2D Convolution and the Census Transform. 2D Convolution is used for Convolutional Neural Networks(CNN), to recognition objects in an image. Furthermore, the Convolution will be used for Gaussian Blur (also known as 2D-FIR). The Census Transform will be used for stereo vision applications to extract depth information. In this thesis, a filter known as Local Binary Patterns (LBP) will also be explored since it is very similar to the Census Transform. Its main application is Face Detection. It is worth noting that an accelerator for the

LBP has not been implemented, but instead a possible design is presented. An overview of the previously mentioned accelerators and the corresponding applications is shown in Figure 1.2.

The current Intel Image Processing Unit (IPU) features a Block Matching Accelerator (BMA) and Bilateral Filter Accelerator (BFA) unit, which share some similarities with the new filters. In this thesis, it is investigated if hardware reuse is possible by extending this accelerator. Other options, including the design of a completely new unit, are also analyzed. Design space exploration is an important contribution of the thesis that will include an analysis both performance and area impact.

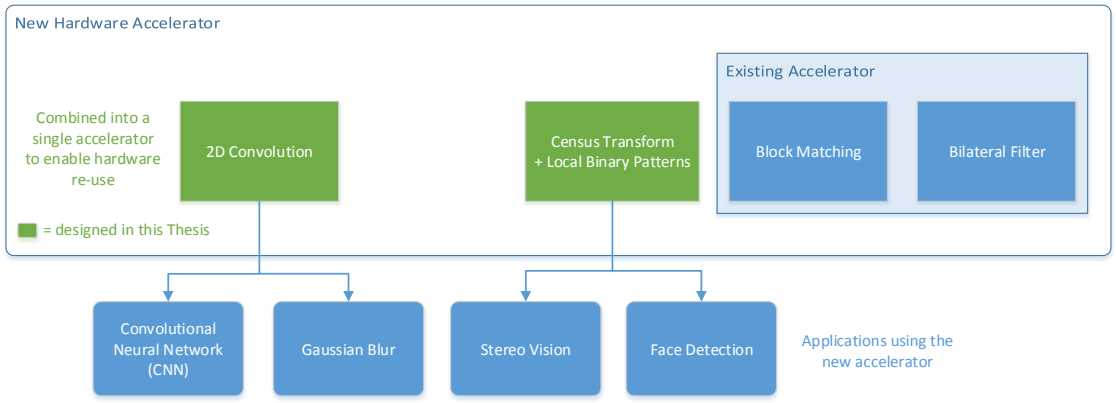


Figure 1.2: Overview of the hardware accelerator which will be designed in this thesis and its applications.

1.3 Thesis Organization

The rest of this thesis is organized as follows. In Chapter 2, the different filters (2D Convolution, Census Transform and Local Binary Patterns) are examined, including an analysis of papers and scientific publications relevant to these topics. A description of how these filters will be used at the Intel Image Processing Unit is presented in Chapter 3, containing speedup calculations and requirements for the accelerator. Chapter 4 explains the current architecture of the Intel Image Processing Unit. It covers both the general architecture and the current implementation of the BMA/BFA accelerator that can be extended. Extending an existing accelerator allows for the reuse of existing hardware and thus limiting additional area for the new accelerator. Additionally it describes the approach and tools used at Intel. The actual design and implementation are described in Chapter 5, which also explains what hardware is reused from the current accelerator. In Chapter 6, the simulation results of important signals in the design are presented. Additionally the achieved speedup, processor utilization, throughput and area usage are given. Finally, Chapter 7 presents the conclusion and possible future work.

2 State of the Art

This chapter describes the different filters that will be accelerated. These filters are the 2D Convolution, the Census Transform, and Local Binary Patterns. Each filter is discussed separately to analyze what it can be used for and what operations it performs exactly. The chapter ends with an overview of several acceleration techniques and technologies that have already been explored in literature.

2.1 2D Convolution

2D Convolution is a mathematical operation that combines two matrices together (by performing multiplication). Generally these are of different sizes; for example, the source image and a 5x5 pixel filter (called the kernel). The convolution is calculated by sliding the kernel over the entire image, at each position the values inside the kernel are multiplied with the pixels in the image. The results after this multiplication are then added together to obtain a single value. By changing the values in the matrix, several image processing algorithms can be described. For example blur, sharpen, emboss, lighten, darken, and edge detection. In Figure 2.1, an example of 2D Convolution is shown. In this case, the emboss function is performed. Other functions are possible by changing the value's and/or size of the kernel. The example of Figure 2.1 shows a 3x3 kernel, other popular sizes include 5x5, 7x7, and 11x11.

As discussed previously, the kernel slides over the image and performs the convolution at each position. However, there is a problem on the edges of the image since this requires information from pixels outside the picture. There are multiple solutions for this problem; the pixels can be regarded as black, white, or the value of the nearest pixel. It is also possible to wrap around the image and use the pixel values on the other side of the image. Another option is to crop the picture, ignoring the entire edge. Since this thesis focuses on the acceleration of filters, it can simply be left to the user by employing the cropping technique. The user can now decide what to do at the edges and add the desired values to the edge of the original picture, temporally increasing the size of the image.

Also, correlation and convolution because these two concepts are closely related to each other[2]. Correlation uses the same technique as Convolution (sliding the kernel across the picture), but with a filter rotated by 180°. Note that if the filter is symmetric, the convolution and correlation give the same results. It is not always clear in literature and manuals if the author means correlation or convolution. However, the difference between them is small, and it can be left to the user of the hardware accelerator to provide the correct kernel.

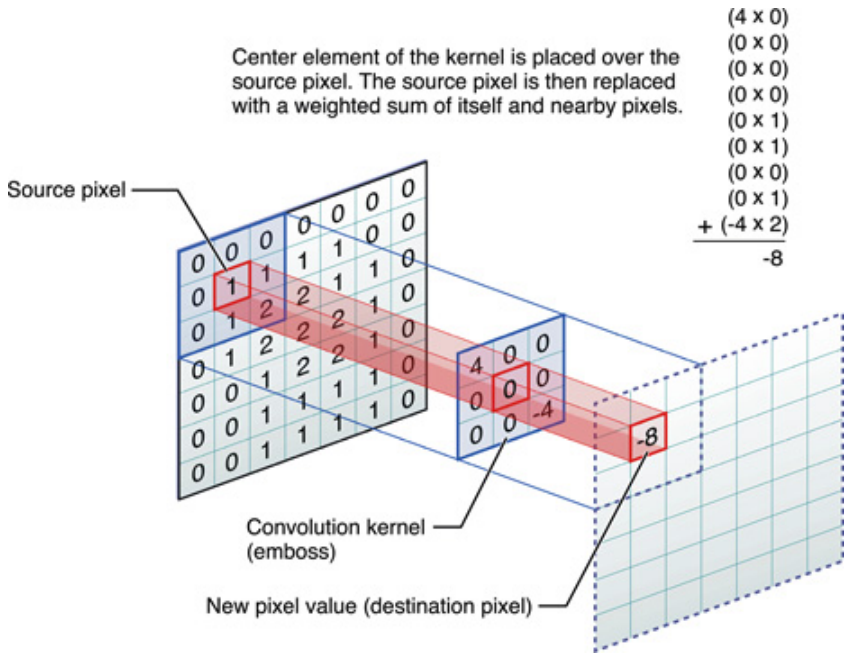


Figure 2.1: Example of a 2D Convolution. [1]

2.1.1 Convolutional Neural-Networks

The Intel Image Signal Processor (ISP) will primarily use the 2D Convolution for Convolutional Neural-Networks (CNN). These networks are used to recognize objects or persons in an image (for example character recognition or in the Automotive industry to recognize objects in front of the car or the signs on the road). In [3] the CNN is employed to detect and identify a person; for example the faces can be matched with a large database from the police to detect criminal activity. For mobile phones and tablets, this can also give interesting applications. For example, the ability to unlock the phone when the owner is in front of the camera or adjust the exposure time of the photo based on the detected face. Facial image processing is further described in [4], which is aimed at embedded systems and uses the CNN for tasks like the automatic focus in cameras and video conferencing. This work presents a set of high-level optimizations to allow the usage of Convolutional Neural Networks on embedded processors. The first optimization is an automatic fractional transformation, which is a tool to generate a fixed-point network from the floating point description. The next optimization is the combination of convolution with sub-sampling. The authors show that any NxN convolution followed by a sub-sampling operation is equal to a single $(N+1)$x$(N+1)$ convolution with steps of two pixels. The improvement in performance of this change is 65% for a 5x5 convolution. The final optimization is related to the memory, by changing it to a line by line based solution. The neural network is implemented, and all of the improvements together resulted in a speedup of up to 700x. However, this is a somewhat unfair comparison. They manage to process 12.8 faces per second and yet they also mention an implementation from someone else who achieves 2.5 faces per second. The speedup compared to that

solution is only 5.1, which is more realistic.

The general architecture of a Convolution Neural Network is shown in Figure 2.2. A Convolutional Neural Network starts with several convolution layers and sub-sampling layers, this is followed by some fully connected layers. The convolutional layer has many filters (kernels) that are convolved with the image; producing the "feature maps". So the actual 2D Convolution is happening in the Convolution layers. In total, the 2D Convolution is applied many times, each time with a different kernel. The contents of these kernels and the fully connected layers allow the network to be trained to recognize objects and faces. Simple features (like edges) are extracted at a higher resolution, and more complex features at a coarser resolution by sub-sampling the previous layer.

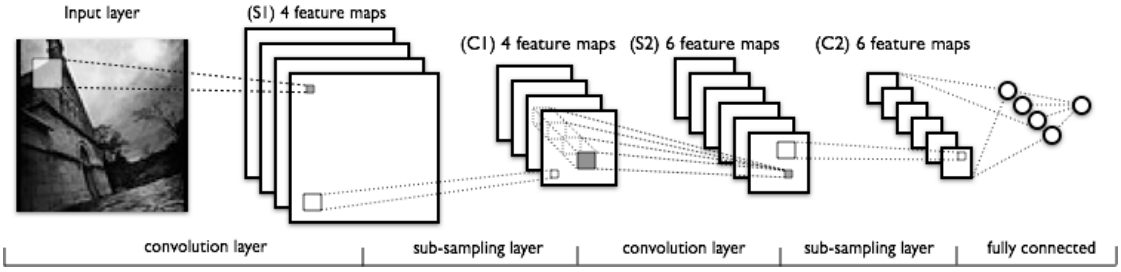


Figure 2.2: A typical structure of a convolution neural network. [5]

2.1.2 Gaussian Blur

Another application for the 2D Convolution is the Gaussian Blur filter, which uses the 2D Convolution to blur the image and reduces the noise levels of a picture. Additionally, these filters are used in edge detection algorithms. Gaussian smoothing is also used in other applications like mosaicing (stitching images together) and for obtaining High Dynamic Range (HDR) images.

The basic equation for a 2D Gaussian distribution is shown in Equation 2.1. The standard deviation σ is the main parameter to control the amount of averaging between the center pixel and its neighbors. On image processing applications, this distribution has to be approximated with the convolution kernel. This approximation results in a second parameter that can be changed: the kernel size. A larger kernel size corresponds to a larger convolution kernel, this allows the storage of more values and will result in a better approximation of the Gaussian distribution. Determining the kernel size is a trade-off between the amount of noise reduction (quality), speed and hardware resources. An example of several 7x7 Gaussian kernels with a different standard deviation σ is given in Figure 2.3, which is generated by MATLAB. The Gaussian Blur can thus be implemented by using the previously described 2D Convolution when providing the right values to the kernel.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.1}$$

The effect of applying a Gaussian blur is investigated in [6]. When capturing a photo, noise is always present. In certain applications, this noise can influence important de-

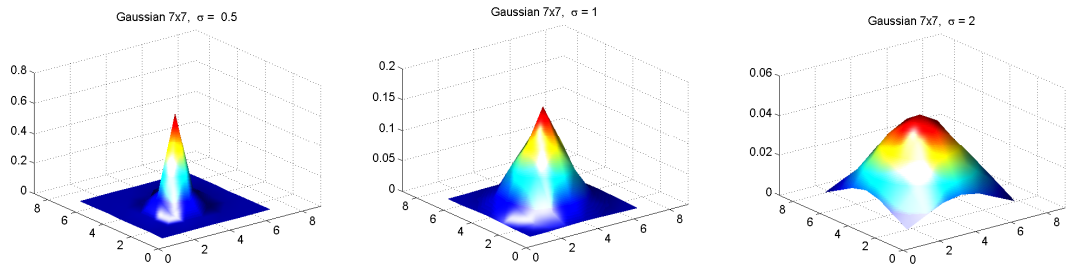


Figure 2.3: Gaussian kernels with size 7x7 and standard deviation $\sigma = 0.5$, 1.0 and 2.0.

cisions, like with medical imaging where the noise can hide a small tumor. However, applying too much filtering to the noise can also influence the diagnosis (the small tumor disappears). The authors compare the filtered image with the original image using a quality factor. This quality factor includes the covariance between the pictures and the distortion in both luminance and contrast. The noise level is measured by the Signal-to-Noise Ratio (SNR). The authors perform some calculations on an image containing noise. They conclude that, when compared to the noise variance, the variance of the Gaussian function has a larger influence on quality. The Gaussian filter should preferably be used on images containing a lot of noise and with a small variance (standard deviation). Besides reducing noise, the Gaussian function can also be applied for segmentation (extracting parts of the image). In this case, a region from the histogram is isolated and apply the Gaussian filter to generate a mask. For segmentation, a large variance is preferred.

A useful overview of Gaussian smoothing is presented in [7], primarily aimed at a fixed-point platform. On fixed-point systems, the filter coefficients have to be rounded, approximating the original distribution. This rounding leads to two errors: quantization errors and average intensity errors (not being able to obtain a unity sum). This work presents a new way to approximate the coefficients. First, the values are rounded and then the center pixel is correct to achieve a unity sum. The results show that it manages to provide a higher Signal-to-Noise Ratio than the traditional method, and it does not require additional resources. Especially when approximating with low available bits, the SNR is significantly better than the rounded version.

The Gaussian coefficients can even be approximated by a power-of-two value[8]. This way no multipliers are required; the Gaussian smoothing can be performed with only shifting and addition. The results are quite good by being able to process a 512x512 grayscale image at 747 FPS. However, by approximating the values, it will probably result in a loss of image quality. The impact on quality has not been adequately described; it would be nice to have some sample images and compare the approximation with an ideal version.

2.2 Census Transform

Besides the 2D Convolution, research was done in the Census Transform since it can potentially share hardware with the 2D Convolution and accelerate more applications.

The Census Transform is often used for Stereo Vision (3D-imaging), to estimate the depth by creating a disparity map. The Census Transform was introduced in 1994 [9]. It works by comparing the intensity of the center pixel with the intensity values of its neighbors inside a window. See Figure 2.4 for an example of the Census Transform. If the intensity of the neighbor pixel is larger or equal than the center pixel, a '1' is stored; otherwise a '0' is stored. The result is a binary vector for each pixel of interest.

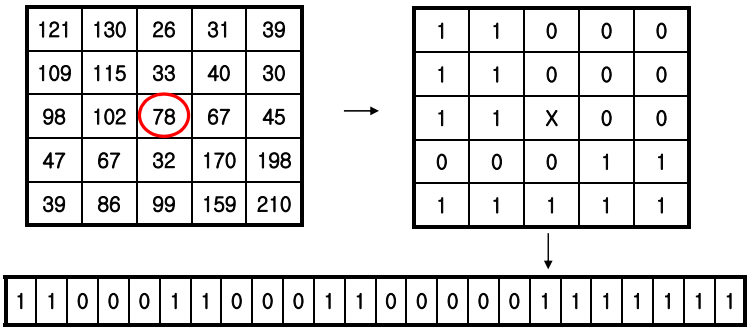


Figure 2.4: Computing the Census Transform by comparing each neighboring pixel with the center pixel. [10]

An overview of several modifications to the original Census Transform is given in Table 2.1. A major weakness of the Census Transform is that it relies a lot on the center pixel, making it sensitive to noise problems. Besides noise problems, the Census Transform can not capture all possible kernel values since the center pixel itself is not used in the comparison[11]. For example, in a 3x3 neighborhood it can only result in $2^8 = 256$ different kernels instead of $2^9 - 1 = 511$ different kernels. To solve this, an improved version known as the Modified Census Transform (MCT) is introduced[11], mainly used for Face Detection. Essentially the modification to the original Census Transform is that now each pixel in the window, including the center pixel, is used. Additionally these are compared to the average value (also known as the mean) of the window instead of comparing them to the center pixel. Several other papers also try to solve the problem of relying on the center pixel[12][13][14]. A small color census transform is presented in [12], by extending the previously described MCT with color information. It uses the color distance and the average of the color distances instead of the regular pixel and the average of the pixel values respectively. The small color census transform compares 6 pixels inside the window with the center pixel by calculating the color distance; each color distance is measured using the Manhattan distance. Comparing fewer pixels than possible in the entire window is known as a Sparse Census Transform. The Modified Census Transform is also used in [13], but here it outputs two bits for each comparison. These outputs are based on the comparison of pixels with both the original center pixel and the average; giving four different outcomes. The Modified Census Transform is compared to the original for Stereo Vision applications, and it shows an improvement up to 10% on matching quality. To get a robust Census Transform, [14] employs a 16x16 Sparse Census Transform combined with the Modified Census Transform. According to the authors, a large Sparse Census performs better than a small regular Census

Transform while the time required is equal. After using the MCT, the authors noticed it is sensitive to small changes of in the window. To solve this, an offset of 1, 2 or 3 is added to the average value.

2.2.1 Stereo Vision

As mentioned before, the main application of the Census Transform is Stereo Vision. Stereo Vision algorithms usually consist of these four steps:

1. Matching cost computation
2. Cost aggregation
3. Disparity computation
4. Disparity refinement

The matching cost computation calculates the similarity of pixels, for example by taking the absolute difference between pixel intensities. Cost aggregation combines these results over a certain support window around each pixel. The disparity computation calculates the difference between the x-coordinates of a particular location in both images. When this difference is large, it results in a high disparity and indicates a point closer to the camera. Finally, the refinement improves the outcome with various post-processing techniques. The matching cost process is the step where the Census Transform can be used to transform both source images (step 1).

An example of the Census Transform in the context of Stereo Vision is shown in Figure 2.5. The original image (top-left) and the resulting disparity map after finishing all steps (top-right) are from the Middlebury Stereo Datasets [15]. The Census transformed images (bottom) were made with MATLAB using the technique described before. The size of the kernel can be changed depending on the application, the example of Figure 2.4 shows a 5x5 Census Transform. The examples in Figure 2.5 are created with a size of 3x3 and 7x7.

An example of a Stereo Vision application is given in [16], which describes a stereo vision system for embedded systems. The framework is essentially a big pipeline, consisting of a LoG (Laplacian of Gaussian) filter and the stereo processing itself. This stereo processing is done with a Census Transform having an 11x11 window. An Altera Cyclone III FPGA is employed for implementation, and the results show that the frame rate is limited by the sensors instead of the processing hardware. The work also compares the performance with other designs. It becomes clear the authors achieve a high performance (both in terms of resolution and frames per second). However, the area and power results are not given.

Table 2.1: Overview of several Census Transform algorithms.

Paper	Target Application	Census Method	Census size	Results	Remarks
[9] (1994)	Stereo Matching	Original	7x7	Large improvement over normalized correlation and Rank transform	First introduction of Census Transform
[11] (2004)	Face Detection	Modified (MCT)	3x3	Detection rates over 90% and false positive rate of 10^{-7} %	First introduction of Modified Census Transform
[12] (2011)	Stereo Matching	Small-color	5x5 sparse	Not yet tested	Compares 6 pixels with center and uses color information
[13] (2013)	Stereo Matching	MCT + Neighborhood	Multiple	Better matching at depth discontinuities	Compares to both center and average value
[14] (2012)	Stereo Matching	MCT + small offset	16x16 sparse	Good, compared to dense census and SAD	
[17] (2010)	Face Detection	MCT + color	3x3	Improved detection rate + fewer false alarms in low resolution images	Adds mean from RGB layers

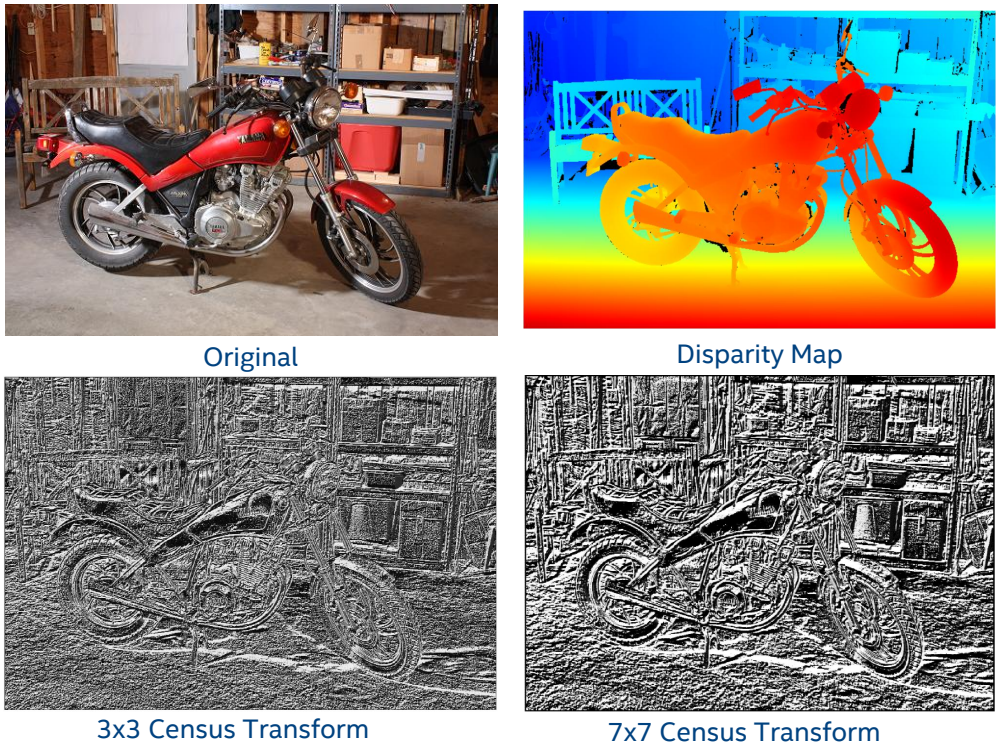


Figure 2.5: Example of the Census Transform using different kernel sizes. Original (top-left) and final Disparity Map (top-right) images are from [15].

Another example of an algorithm that uses the Census Transform for stereo matching is given in [18]. This work specifically looks at the use of global optimization techniques to optimize the matching quality of local matching. The proposed algorithm consists of a Sparse Census Transform followed by a modified semi-global matching. Semi-global matching is used to minimize the energy in multiple directions (horizontal, vertical and diagonal). The semi-global matching is modified by assuming that information from a piece of the picture is enough instead of the whole image. The authors also look at textureless areas (e.g. a large white wall), where reliable matching is difficult. To improve this, the authors first calculate a confidence value and texture value for each pixel to determine if that pixel can be used in the disparity map. Next, segmentation and plane fitting are used to optimize textureless areas and pixels with a low confidence value. The algorithm is tested against the Middlebury datasets to compare the different improvements. The results show that the (modified) semi-global matching improves the quality. The best results are achieved by combining the Census Transform, semi-global matching and plane fitting.

It is worth noting that there are other methods to obtain a Stereo image. In [19] the Sum of Absolute Differences (SAD) algorithm is compared to the Sum of Hamming Distances combined with the Census Transform. The SAD subtracts pixels in a window between two images (reference and target image). After this subtraction, the absolute values are added together. When salt-and-pepper noise is added, the SAD does not

manage to differentiate the objects anymore (information is lost). According to the authors, even after filtering with a median filter the final disparity map remains a lot worse compared to the situation without noise. The Sum of Hamming Distances (SHD) is often used after the Census Transform. The SHD performs a bitwise XOR operation on the Census transformed values from the left and right image. This is usually followed by a bit-counting step to count the number of results with a '1' (indicating a difference between the two strings). When the same noise is added as before, the SHD manages to obtain a way better disparity map compared to the SAD situations. The authors show that applying the median filter does not result in a significant improvement.

2.3 Local Binary Patterns

Finally, Local Binary Patterns (LBP) are explored, which is a filter similar to the Census Transform and often used in facial image analysis. It can be used for tasks like face detection and face recognition. Like the Census Transform, it compares each pixel in a window with the center pixel and is known for its tolerance of illumination changes. An example of the LBP is shown in Figure 2.6. When comparing the original 3x3 Census transform with the original 3x3 Local Binary Patterns, only the output order is different. The LBP uses a clockwise order while the Census transform stores the result in a line-by-line based matter. Because this LBP operation shows many similarities with the Census transform, it is worth investigating its potential since it might also be possible to accelerate this type of operation.

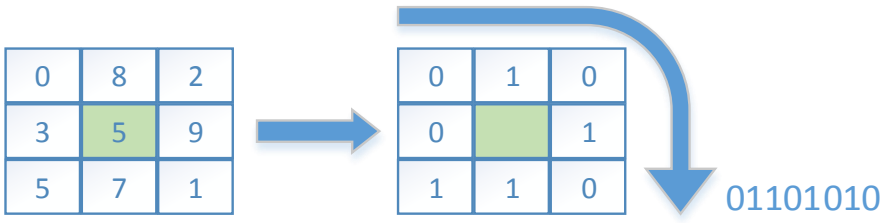


Figure 2.6: Example of the Local Binary Patterns; pixels are compared to the center pixel and the result is stored in a clock-wise matter.

In [20] a large number of papers regarding the latest LBP trends are compared to each other. The original LBP works on a 3x3 window. To handle textures and faces at different window sizes, the operator has been modified to also handle larger windows. This can be done by employing a circle from the center pixel, with points distributed among the circle. Sometimes not all points are located directly on a corresponding pixel, so interpolation is required to use the circle correctly. One of the recent trends is to improve the discriminative capability, by encoding more information. For example by comparing all pixels, including the center pixel, with the mean value of the pixels inside the window (a method known as Improved or Modified LBP and similar to the Modified Census Transform). Another improvement is called Extended LBP, which also encodes the exact gray value differences. The second trend is to improve robustness since the original LBP is sensitive to noise. To solve this, a method called local ternary patterns

is introduced; this method extends the LBP by employing 3-value results where the pixels are compared to the center pixel and a user-specified threshold. The third trend is to improve the choice of the neighborhood. This includes the number of points to use on the circle, the size of the circle (3x3, 5x5, 7x7), and the distribution of the points. The main reason to use this circular approach is because of the rotation invariance property. However, that property is not needed for all applications. Other shapes are also possible; for example, the Elongated LBP uses points on an eclipse that improves the capturing of anisotropic features (like facial images). Furthermore, it is possible to capture information from larger structures by averaging blocks and applying the LBP on these larger blocks. The fourth trend is to extend the LBP to a 3D version, but this is not relevant to the topic of this thesis.

2.3.1 Face Detection and Face Recognition

Facial image analysis is also covered in [20], which compares many papers using LBP for several applications (face detection, recognition, and expression analysis). Specifically it contains a performance comparison of 11 different Face Recognition methods based on LBP, which are all executed on the same database. The reported accuracy depends on the chosen subset of data. For example, illumination changes can be tested with a specific database (known as the `fc` database). All methods score good (above 90%) on the regular Facial Expression database. Some methods have problems with illumination, the normal LBP, and Multi-Scale LBP are both achieving around 70% accuracy, but the other methods perform the same as with the Facial Expressions database. The accuracy is lowest with the Duplicate database, which is used to test the aging of subjects. Almost all methods score below 80% here. In [21] the Improved LBP is used, which compares all pixels (including the center pixel) with the average. The results show a detection of above 90% and a false positive rate of $2.99 * 10^{-7}$. Several LBP variants are compared in [22], it demonstrates that the Center Symmetric Local Binary Pattern achieves the best results when compared to the original LBP, Multivariate LBP, and LBP Variance. The Center Symmetric LBP works by comparing the pixel values in pairs that are opposite to each other instead of comparing it to the center pixel. The recognition rate is compared for different window sizes, and it can be concluded that the recognition rate increases on a bigger window size; up to 87% for the Center Symmetric LBP on a 30x30 window. Age variation is a difficult task for face recognition systems, a problem that [23] attempts to solve. It uses the Multi-Scale LBP as part of the feature extraction, which describes the image at different scales (window sizes). After the feature extraction, two separate methods of classification are compared, but these are not relevant to the LBP. The final performance is a 70% recognition rate on images with age variation and occlusion (part of the face covered). Face detection for mobile phones is covered in [24], which uses two LBP methods. First it uses original LBP to extracts features and is followed by a co-occurrence of adjacent LBP. This effectively combines multiple LBP results in a 2D histogram. The results show a true positive rate of 97.2% and a false positive rate of $5.76 * 10^{-6}$. A big advantage is that it requires fewer operations compared to other face detection methods, so it can operate faster on weaker hardware like a mobile phone.

2.4 Acceleration technologies and the latest trends

There are several ways to accelerate applications. A Graphics Processing Unit (GPU) is often used for acceleration because GPUs feature many (more than 1000) small cores to support parallel computation. Another option is to use a Field-Programmable Gate Array (FPGA). FPGAs feature programmable logic blocks; allowing the user to create a custom solution that can still be modified later. An Application-Specific Integrated Circuit (ASIC) is used to build a circuit that is optimized for a particular usage, the hardware is fixed and cannot be changed later. Finally, it is also possible to combine these solutions together and create a hybrid accelerator.

In [25], GPUs and FPGAs for accelerating compute intensive applications such as Gaussian Elimination, Data Encryption Standard, and Needleman-Wunsch, are studied. The authors consider both performance and programmability. The results show that FPGAs are the fastest in the tested applications. The GPU requires a high utilization to benefit from the latency hiding design. Programming the FPGA with VHDL is more complex compared to GPU (using CUDA) and CPU, this is measured by looking at the length of the code. The comparison between the different platform can be improved by performing more tests and also include metrics like chip area and power consumption. Also, the integration and combination of the various acceleration technologies together is not discussed.

In [26] the possible developments of single-chip heterogeneous computing is investigated. Optimizing energy efficiency plays a significant role in these developments since the power does not scale the same as the increase in transistor density. Additionally, the required bandwidth is increasing in both data rate and the number of external pins. For these reasons, the authors investigate the combination of traditional processors with unconventional cores (custom logic, FPGAs or GPGPUs). According to the results, unconventional cores can only show a significant performance gain (above 10x) when a large portion of the code can be executed in parallel (according to the authors more than 90%). Custom logic provides the best performance for all tested applications, but less efficient solutions were also able to reach the bandwidth-limited performance in e.g. the Fast Fourier Transform. More research should be spent on incorporating varying degrees of parallelism in an application. Also, FPGAs integrated with conventional multicores should be investigated. The challenge to solve first is the memory bandwidth limitations.

Acceleration specifically aimed to image processing is discussed in [27]. This work compares FPGA, GPU and CPU in the following applications: two-dimensional filters, stereo-vision, and k-means clustering. Although FPGAs have a low operational frequency, they can achieve a very high performance in many applications thanks to its flexibility.

GPUs and FPGAs are compared for Image Convolution Processing in [28]. The GPU version (using CUDA) achieves the best speedup of 200x compared to a plain C version. A speedup of up to 70x is obtained when compared to MATLAB and 20x when compared to the FPGA implementation. However, the FPGA version lacks a true parallel approach due to resource limitations of the used FPGA. Also, the chosen FPGA does not feature dedicated multiplier blocks that could result in a larger speedup and maybe even achieve a higher speedup than the CUDA version. Besides this, the authors only compare the

execution time and number of clock cycles for the different solution, other metrics like power and area are not measured.

Intel already has accelerators in the current processor known as the Block Matching Accelerator and Bilateral Filter Accelerator (BMA/BFA). These are further described in Chapter 4. Comparing it to the previously mentioned techniques, this is a custom logic (ASIC) solution.

2.4.1 Acceleration of 2D Convolution

In this section, we will discuss the different acceleration options for 2D convolution and its applications.

An implementation of Convolutional Networks on a low-end DSP-oriented Field Programmable Gate Array (FPGA) is presented in [29]. The implementation uses a soft processor on the FPGA and a new Vector Arithmetic and Logic Unit (VALU). The operations that are used in the network are implemented on hardware and provided as new macroinstructions. The resulting system is used for face detection, and when running on an image with 512x384 pixels, it takes 100ms to process (resulting in 10 frames per second).

In [30] the acceleration of CNN-based algorithms on hardware is investigated. The application of the CNN-algorithm is the detection of astronomical objects from a telescope image. Hardware acceleration is performed on a High-Performance Reconfigurable Computer (HPRC), combining general purpose standard microprocessors with custom hardware accelerators based on FPGAs. The results show a speedup of 13 times compared to a standard personal computer.

Acceleration of Neural Networks, and specifically the convolution phase, is presented in [31]. The work presents a hardware accelerator for accelerating the HMAX model (this models the visual cortex). Specifically, the S2 stage is accelerated. This stage involves the convolution with a lot of different kernels. The accelerator is implemented on an FPGA, and the results show speedups ranging from 5 to 11 compared to a Tesla GPU.

A coprocessor for Convolutional Neural Networks (CNNs) is presented in [32]. The implementation involves a coprocessor that is combined with off-chip memory, this memory is implemented using several memory banks. Secondly, data precision is reduced, and multiple data words are packed into each memory operation. According to the authors, this is the first time research is done at accelerating the entire CNN with a programmable coprocessor. The coprocessor is designed by grouping vector processing elements together.

A Convolutional Face Finder (CFF) algorithm implemented on an FPGA is presented in [33]. The structure of the CFF is quite similar to the Convolutional Neural Network since it involves multiple convolutions, sub-sampling and fully connected layers. This work uses a tool called SynDEx for design space exploration. Using a ring architecture, data parallelisms of the Convolutional Face Finder can optimized in an efficient way, and results in a low transfer bandwidth. The ring architecture connects each processing elements to its two nearest neighbors by a direct link, and each processing element has its own local memory.

Convolution involves many multiplications and accumulations. These accumulations

are also explored in more detail. In [34] several addition structures for FPGAs are compared; these are carry propagated adder (CPA), Carry Save Adder (CSA) and a combination. Although it is aimed at FPGAs, it might still be worthwhile investigating this as well for the acceleration.

Memory potentially plays a major role in accelerating the Convolutional Neural Networks. In [35] an accelerator is investigated with a flexible memory hierarchy. The accelerator aims to maximize the memory usage by scheduling for data locality. The resources on the FPGA can be reduced by 13x while maintaining the same performance; if the same amount of resources is used the accelerator achieves an 11x speedup. The size of the kernel is not fixed; this is done by employing a cluster of Processing Elements. These are used to perform the Multiply Accumulate operation; the convolution can be mapped to this by iteratively applying a kernel value and image value to each Processing Element. The memory subsystem provides flexibility and an increased bandwidth. A reduction in communication is achieved when successive computations use overlapping values.

Hardware designs for the implementation of several image filters are presented in [36]. These filters include (weighted) median and Gaussian filters. The architectures are focused on speed and area usage and implemented for an ASIC with 65nm technology. The authors implement a pipelined sliding window, by using registers and FIFOs. The different filters are implemented in hardware as separate units and compared to the software solution (MATLAB).

In [37] a reconfigurable hardware implementation of a median filter is presented. It contains programmable window sizes, allowing to use sizes from 3x3 to 7x7. The median filter is used to remove noise and is quite similar to the other filters described before. It uses neighboring pixels to calculate the new pixel value, by sorting them on value and take the median. The authors test the implementation using various images, starting at 4x4 pixels and up to 128x128 pixels. They compared the results with an MATLAB/C++ implementation. The hardware (FPGA) implementation is faster at small image sizes. However, increases in the image size result in an increase in the processing time. This even resulted in a situation when the MATLAB/C++ version performed faster than the FPGA version. This might be improved by increasing the clock frequency that is currently running at 50 MHz.

2.4.2 Acceleration of Census transform and Local Binary Patterns

Since the Census Transform and Local Binary Patterns are very similar, the acceleration options (implementations on GPU, FPGA or ASIC) can be expected to be very similar as well.

The Census Transform is discussed in [38], where it is used on an FPGA to perform low-cost stereo vision. The Census Transform has a highly parallel structure, making it a perfect candidate for acceleration. The authors concluded that a 13x13 pixel window size combined with a search disparity of 20 pixels resulted in a good trade-off between performance and computational complexity. Frame buffers are no longer necessary because calculations are performed real-time. The results show that it is possible to achieve 40 frames per second on a 320x240 resolution. The total design uses 57% of the FPGA

resources and can run up to 26MHz.

The Census Transform is also used in [39]; in this case, it is used to perform face detection. The proposed face detection method is aimed at the detection of faces in variable illumination conditions, by employing the Modified Census Transform (MCT). The MCT uses a moving 3x3 window and instead of comparing pixels with the center pixel, it now compares the pixels to the average value. Besides the MCT, the hardware consists of noise reduction, image scalar, candidate detector and a few more modules. The final system has a 99.76 % detection rate in various illumination situations. The system is tested on an FPGA with a 320x240 pixels camera. It can achieve 149 frames per second and detect up to 32 faces concurrently. Additionally, the authors developed an ASIC solution that achieves real-time (30 frames per second) performance on a 13.5 MHz clock frequency. The power consumption is 226 mW.

In [40] an FPGA is used to implement a high-quality Stereo Vision system capable of processing a 1024x768 image at 30 frames per second. It is based on the AD-Census algorithm that is currently ranked at the 6th position of the Middlebury benchmark, but was ranked second when the authors published their work. The AD-Census works by combining information from both the absolute differences (AD) and the census transform. Unfortunately, the authors do not specify how the Census Transform itself is implemented.

A processor for local binary patterns is introduced in [41]. The proposed processor is programmable allowing the software to be modified, something that is not possible on dedicated hardware. The processor is capable of performing two different types of a 3x3 LBP; with bilinear interpolation and without interpolation. Normally this interpolation would require multiplications, but to save hardware it is performed by shift and addition operations. This approximates the normal interpolation and thereby introduces an error. The processor is tested on an FPGA, and it turns out that only 3% of the Logic Cells are used. This is because most of the resources are used by the interconnect. Furthermore, it is synthesized on 180 nm technology and results in an area of 15825 NAND-gate equivalents. The processor is compared to a VLIW DSP from Texas Instruments and a workstation implementation. The latency compared to the DSP is equal, comparing power is more difficult because of different implementation platforms. The authors claim an improvement in latency of 17,5 compared to the workstation and it is only 2,0 worse than ASIC, but it is not explained how these numbers are obtained.

Local binary patterns are also used in [42] to detect the heads and shoulders of humans in real-time using an FPGA. Using only the upper body has an advantage over using the entire body because of possible occlusions when used in surveillance systems. The system uses a non-redundant LBP by considering a pattern and its complement the same. Additionally it only includes uniform patterns that have at most two bitwise transitions between ‘0’ and ‘1’. The authors compared several variants of LBP: the normal, with a biased threshold, with average value as threshold, with the mean value as the threshold and a threshold based on the mean and standard deviation. The last one gives the best results in terms of accuracy. However, implementing this LBP version would require a division. They modified the formula, so it performs a division by a fixed value. This still requires a division by nine which is performed as an LUT operation. The final system running on an FPGA is capable of processing a 640x480 pixels image

at 60 frames per second.

Finally [43] proposes a hybrid system containing a Xilinx Zynq (Processor combined with FPGA) and a Nvidia Jetson TK1 (Processor combined with GPU) to perform face recognition. The algorithm is split into two parts; face detection is carried out on the Zync, and the actual recognition is done on the Jetson. The compute intensive tasks of the face detection component are implemented on the FPGA; this includes the LBP. Unfortunately, no information is provided about the actual LBP implementation. The final system is used on images with a resolution of 1280x720 pixels. A speedup of 69 is achieved compared to the situation when running only on the processor. Compared to the Zynq a speedup of 4,8 and compared to the Nvidia platform a speedup of 3,2 is achieved. Furthermore, it is 40% more energy efficient than the sequential version.

2.5 Conclusions

In this chapter, the different filters, and their corresponding applications are investigated. These are the 2D Convolution, Census Transform, and Local Binary Patterns. The 2D Convolution is performed by sliding a kernel over the image, at each position the values from the kernel are multiplied with corresponding values in the image. The results are added together to get a new value at the center of the kernel. The 2D Convolution at Intel will mainly be for Convolutional Neural Networks (CNN) and Gaussian Blurring. The neural networks are used to detect and recognize objects in an image; for example to identify a person. The general structure of these networks consists of several convolution and sub-sampling layers, followed by fully connected layers. Gaussian blur is mainly used for reducing noise present in an image. This can be implemented by using the 2D Convolution if the proper values are used inside the kernel. These values have to be approximated, which is especially challenging when using a fixed point platform.

The second filter is known as the Census Transform and is often used for Stereo Vision applications. Similarly to the 2D Convolution a window slides across the image. At each position, the values of the pixels inside the window are compared to the value of the center pixel. If the value of a pixel is larger or equal than the center pixel, a '1' is stored; otherwise a '0' is stored. According to the literature study, a major weakness of the Census Transform is that it relies too much on the center pixel, making it sensitive to noise. Several modifications are presented, like the Modified Census Transform, which compares all pixels to the average value of the pixels inside the window. The Census Transform can be used in Stereo Vision applications to compute the matching cost. Using the Census Transform a good performance is possible (real-time processing), especially when using the Sparse Census Transform since that version uses fewer pixels. The final quality mainly depends on the steps after the Census Transform. For example, a higher quality can be obtained by combining the Census Transform with global optimization techniques.

This chapter also explores the filter known as Local Binary Patterns (LBP), which is very similar to the Census Transform since it also compares pixels inside a window to the center pixel. It is often used for facial image analysis (face detection and face recognition). When comparing a 3x3 Census Transform with the original 3x3 Local Binary Patterns, only the output order is different. The LBP uses a clock-wise order to store the results

while the Census Transform uses a line-by-line order. Several modifications to the original LBP are investigated. Larger window sizes are handled by 'drawing' a circle in the window and distribute points on that circle. Since not all points will be located exactly above a pixel, linear interpolation is required. The reason for this circular approach is to obtain rotation invariance. Another possible modification is the comparison of pixels with the average value to avoid noise problems, similarly to the Modified Census Transform. Currently, the biggest challenge when using the LBP for face recognition is to recognize persons after an extended period of time (due to aging) and when the face is covered by objects (occlusion). Most variants of applications using the LBP only manage to detect up to 80% of the persons in this situation.

The chapter ends with an overview of different acceleration techniques and technologies in general and specifically for the different filters. The most common way to accelerate algorithms is by using GPUs and FPGAs. The available speedup varies a lot; this depends on the type of FPGA and GPU used and the application itself. Best performance can often be achieved with custom logic (ASIC), which can also lower the energy consumption. More research should be spent on the integration of these platforms together. For example, an FPGA coupled with a multicore or GPU.

Most of the accelerators proposed up to now are aimed to accelerate one particular application such as the convolution phase of a Neural Network or a face detection application. In this work, a new accelerator will be described that is capable of performing multiple different filter operations while sharing the same hardware. The new accelerator offers a high degree of flexibility; it is now possible to execute a 2D Convolution, Census Transform or Local Binary Patterns on different window sizes (5x5, 7x7 or 11x11). Besides this, it is still capable of performing the original Bilateral Filter and Block Matching algorithms from the current hardware accelerator. Since the accelerator will be implemented on ASIC, this flexibility still allows firmware developers in the future to use the accelerator in many different configurations.

Target Applications and Requirements

3

As described before and shown in Figure 1.2, the accelerator will be used for the following applications:

- 2D convolution
  - Convolutional Neural Networks (CNN)
  - Gaussian Blur
- Census Transform and Local Binary Patterns
  - Stereo Vision
  - Face Detection

In this chapter, applications for the different filters are examined in the context of the Intel Image Signal Processor (ISP), this includes obtaining the theoretical speedup using Amdahl's law. This analysis is followed by the specifications, requirements and desired metrics.

3.1 2D Convolution: Convolution Neural Network

The Convolution Neural Network used at Intel has a structure similar to the one shown in Figure 3.1. It consists of several *convolutional* layers containing the 2D Convolution, Non-Linearity mapping, and Max-pooling. These layers are followed by one or more *fully-connected* layers, which consists of a Linear Transformation (Matrix Multiplication) and Non-Linearity mapping. The amount and content of these layers are not fixed and depends on the application and desired results.

The non-linearity mapping performs the function given in Equation 3.1 for each pixel. Pooling is comparable to sub-sampling, but only the maximum value inside the window is passed instead of the average. According to [44], a max pooling operation is vastly superior for capturing invariances in image-like data, compared to a subsampling operation.

$$y = max(0, x) \tag{3.1}$$

The actual 2D Convolution is shown in green since this will be accelerated in this thesis. The convolution is performed using several different kernels; the content (values) of these kernels can be trained for object recognition. Each iteration of the CNN consists of a different amount of inputs and outputs; at the beginning of the process it starts with three inputs (RGB) but that increases after each convolution step. The kernel size and the number of kernels vary for each stage in the process.

In Figure 3.2 an example of the data flow of the CNN used at Intel is shown. This data flow diagram will be used for the speed up calculations. It starts with an input

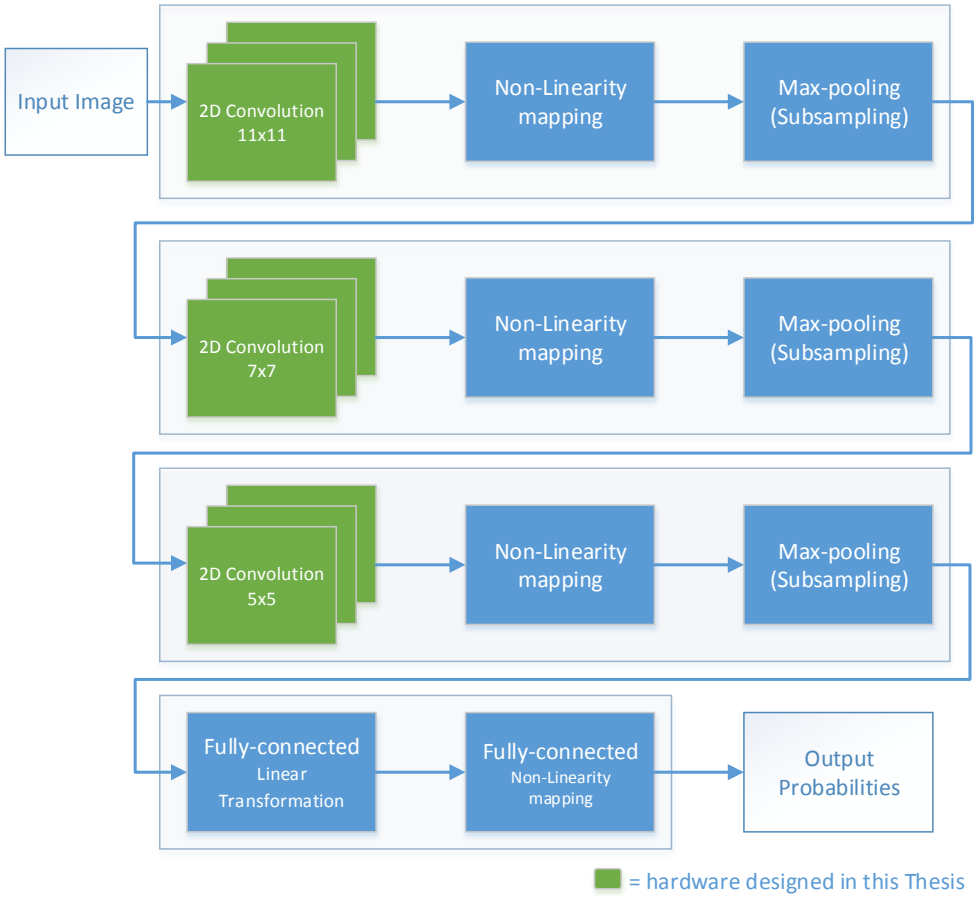


Figure 3.1: Fundamental structure of the Convolutional Neural Network.

image of 214x214 pixels. The 2D Convolution is then applied with an 11x11 filter and stride 3 to skip certain pixels, the result after the first convolution is 64 outputs. The output of Layer 1 goes to Layer 2 as input, which features a 7x7 filter with stride 1 resulting in 128 outputs. This is followed by two 5x5 filters with stride 1 resulting in 256 and 512 outputs. Finally, two fully connected layers are used to obtain the final 1x100 output.

3.1.1 Theoretical Speedup

Using the example application shown in Figure 3.2, an analysis can be performed to see the best acceleration options. The number of operations for each layer have been calculated and is given in Table 3.1. The number of operations for the convolution step is based on the size of the output image in each step, which is determined by the kernel size and the stride. For each output pixel it requires $Size * Size$ operations ($Size$ is the window size; 11x11, 7x7 or 5x5), this is multiplied by the number of output layers and

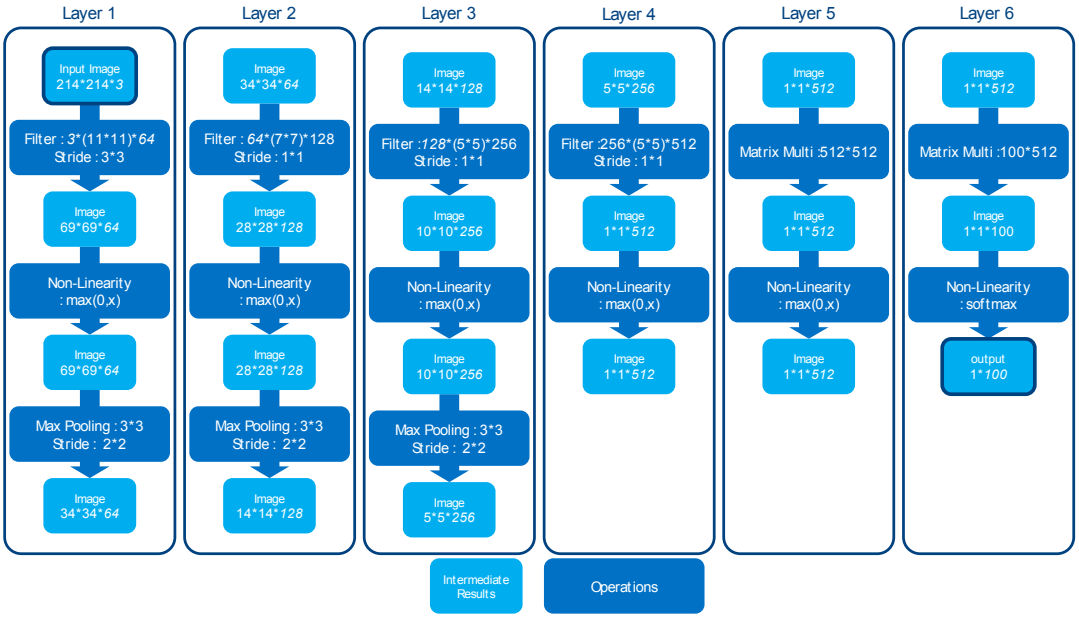


Figure 3.2: Example of the Convolutional Neural Network used at Intel. [45]

Table 3.1: Analysis of the example CNN application: number of operations required for the different steps.

Stage	Convolution	[%]Total	Non-Linear	[%]Total
1	214849536	18,26%	295936	0,03%
2	629407744	53,50%	100352	0,01%
3	163840000	13,93%	25600	0,00%
4	6553600	0,56%	512	0,00%
5	0	0,00%	0	0,00%
6	0	0,00%	0	0,00%
		86,25%		0,04%

Stage	Max-Pooling	[%]Total	Fully-Connected	[%]Total
1	665856	0,06%	0	0,00%
2	225792	0,02%	0	0,00%
3	57600	0,00%	0	0,00%
4	0	0,00%	0	0,00%
5	0	0,00%	134218240	11,41%
6	0	0,00%	26214912	2,23%
		0,08%		13,64%

input layers to get the total operations in that step. The non-linearity step performs one operation for each pixel. Moreover, the Max-Pooling step performs $3 * 3$ operations for each output pixel (each pixel in the 3x3 window need to be processed so the maximum can be found). Finally, the fully connected layers perform a matrix multiplication followed by a non-linearity step. From the calculations, it becomes clear that the 2D Convolution requires the most operations (86% of the total operations). Also, the 512x512 matrix multiplication is computational intensive since it has a complexity $O(n^3)$.

Using Amdahl's Law the maximal speedup can be calculated. See Equation 3.2 for a description of Amdahl's Law, here P represents the parallel fraction of the program that can be accelerated and N represents the number of processors available. If we take the number of processors as infinite, the right side of the equation is obtained which indicated the maximum theoretical speedup.

$$Speedup = \frac{1}{(1-P) + \frac{P}{N}} \leq \frac{1}{(1-P)} \tag{3.2}$$

Using the values from Table 3.1, the highest speedup of accelerating the 5x5 convolution is 1,17x. Accelerating the 7x7 convolution results in a 2,15x speedup and with the 11x11 convolution a speedup of 1,22x is possible. So it becomes clear that accelerating only one convolution size does not result in a huge speedup. However, if we accelerate all convolution steps (with different kernel sizes), the speedup is a lot larger. In total the convolution is used for 86%; this results in a theoretical speedup of 7,27x for the entire application. So accelerating the 2D Convolution shows a solid potential to speedup the whole Convolutional Neural Network. Of course this is only a theoretical speedup, in practice the speedup will be lower because the model only looks at computational operations and does not include communication and memory operations.

3.2 2D Convolution: Gaussian Blur

Gaussian blur (or Gaussian smoothing) is a direct application of the 2D Convolution since it involves convolving the image with a Gaussian kernel. Besides the convolution, there are no other calculations that need to be performed. This means that the theoretical speedup depends entirely on the speed of the 2D Convolution, making it a perfect candidate for acceleration. When an 11x11 Convolution is required, it can theoretically do 121 multiplications in parallel. These have to be added together; this requires 7 stages when using an adder tree (requires $O(log\ n)$ time). On a sequential implementation, it requires the same amount of operations as the amount of numbers to add together (121 for the 11x11). The results are shown in Table 3.2. A maximum speedup of 30,3x is theoretically possible when using an 11x11 Convolution, a 7x7 has a maximum speedup of 14,0x, and the 5x5 convolution has a maximum speedup of 8,3x. On average, the speedup is 17,5x.

Table 3.2: Required number of operations for the Gaussian Blur.

Size	Sequential	Parallel	Speedup
5x5	50	6	8,3x
7x7	98	7	14,0x
11x11	242	8	30,3x

3.3 Census Transform: Stereo Vision

The Stereo Vision application consists of the flow shown in Figure 3.3. The green elements are the blocks that will be accelerated in this thesis. Note that these steps correspond to the steps explained before in Section 2.2.1. Region aggregation can be done by performing a 2D Convolution, so this is why it can also be accelerated. The minimum filter (resulting in the technique known as shiftable windows aggregation) and median filter are added to obtain a higher quality result. Quality can be further improved by using other methods like a different form of aggregation (cross-region) and adding scanline optimization. However, the focus of this thesis is mainly on accelerating the 2D Convolution and Census Transform and not about improving Stereo Vision in general. For this reason, the data flow is shown in Figure 3.3 will be used as the reference.

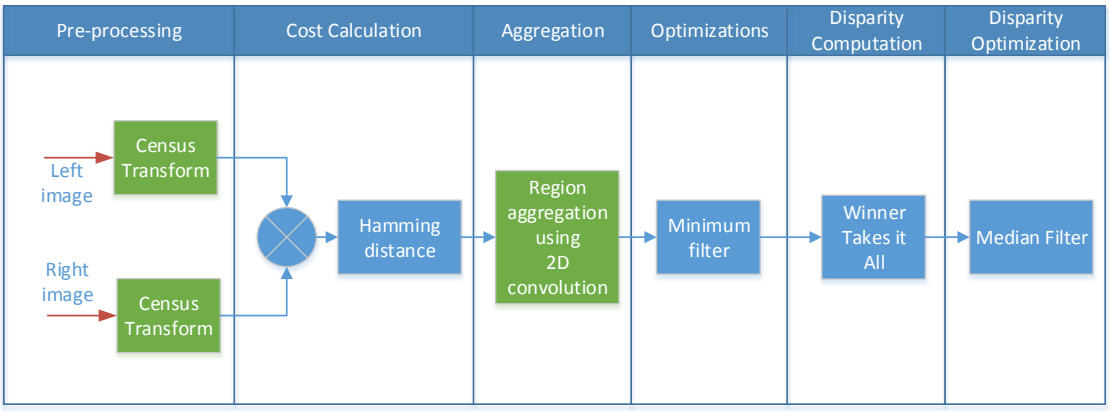


Figure 3.3: Data flow of the Stereo Vision application.

As shown before in the literature survey, there are many variants for the Census Transform. Currently, the application uses a 7x7 Sparse Census Transform. The advantage of this is that it compares 24 pixels with the center pixel instead of 48 which a normal Census Transform would do. Also, this results in an output of 24 bits instead of 48 bits; reducing the communication time and the computation effort of the steps after the Census Transform. Although modification of the application is not really the goal of this thesis, changing the Census Transform to another variant might result in a better quality. Especially the Modified Census Transform, which uses the average value instead of the center pixel, shows big potential according to the literature survey. Note that calculating the average value can also be done with a 2D Convolution. Furthermore,

it is worth investigating if the Local Binary Patterns can be used for Stereo Vision as well since it is very similar to the Census Transform. The major difference is that it only compares the pixels lying on a circle; reducing the computation effort even further. For now we assume the 7x7 Sparse Census Transform since it was part of the original application; the structure of this transform is shown in Figure 3.4. Here the checkerboard pattern is clearly visible; the blue locations are the pixels that are compared to the center pixel (shown in orange).

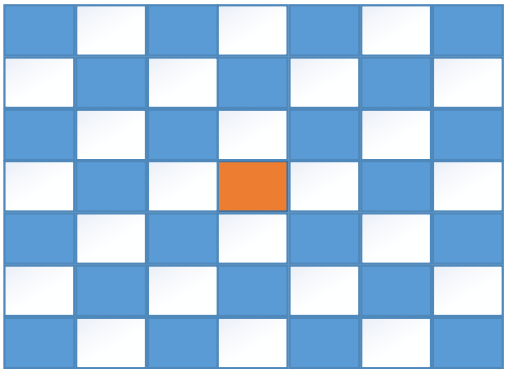


Figure 3.4: Structure of the 7x7 Sparse Census Transform.

3.3.1 Theoretical Speedup

To calculate the possible speedup, the number of operations are determined for each step shown in Figure 3.3. The results are given in Table 3.3. This table assumes a sequential implementation where each operation is equal (e.g. multiplication and addition are both considered as a single operation). Four different implementations of the Census Transform are compared; Sparse Census, Normal Census, Sparse Modified Census and Normal Modified Census (modified Census compares pixels to the average value). Note that the optimization steps are not included here; this is not a fundamental component for a Stereo Vision application and highly depends on the desired results. This table assumes a 7x7 window for both the Census Transform and the Aggregation and shows the results for the processing of a Full-HD 1080p image (1920x1080 pixels). In this case, the Sparse (modified) Census Transform compares 24 pixels to the center, and the normal (modified) Census Transform compares 48 pixels. Furthermore, the Modified Census first calculates the average of these pixels, requiring 25 multiplications and additions for the Sparse version and 49 for the normal version. This explains the increase in the number of operations. The second step; Hamming Distance; compares the results obtained from the Census Transform. It does this by taking the `XOR` of both results and counting the number of ones in the binary outcome. The sparse 7x7 results in a 24-bit result and the normal 7x7 gives 48-bit, independent of using the modified Census or normal Census. The third step is the aggregation of the results by using a fixed 7x7 window, basically performing an averaging. The number of results after the Hamming distance does not depend on the used Census Transform, so this is a constant value in the table. Finally, the “Winner Takes it All” computes the minimum of the achieved results. Again, this

is independent on the used Census Transform.

Table 3.3: Number of operations (in thousands) required for Stereo Vision.

Census Variant	Census		Hamming		Aggregation		Winner	
Sparse Census	99533	2%	1492992	29%	3048192	60%	466560	9%
Normal Census	199066	3%	2985984	45%	3048192	45%	466560	7%
Sparse MCT	306893	6%	1492992	28%	3048192	57%	466560	9%
Normal MCT	605491	9%	2985984	42%	3048192	43%	466560	7%

These values are also shown in Figure 3.5, which shows the percentage of the total operations that the particular function is used. Clearly the Region Aggregation is the most intensive step of the Stereo Vision application, but this one can already be accelerated by using the 2D Convolution accelerator. Figure 3.6 shows the potential speedup when accelerating parts of the application. This speedup is based on the average number of operations for the different Census Transforms. Here it becomes clear that accelerating the Census Transform by itself does not result in a huge speedup, but when both the Census and Region Aggregation are accelerated a speedup of 2,33x is possible. Note that accelerating the Hamming distance would also be a good option. However, the reason it has a lot of operations is because it takes a massive amount of data from the Census Transform and performs a simple `XOR` and addition on it. The ratio of operations per output is therefore expected to be not that large, and the communication bandwidth is expected to be the more dominant factor.

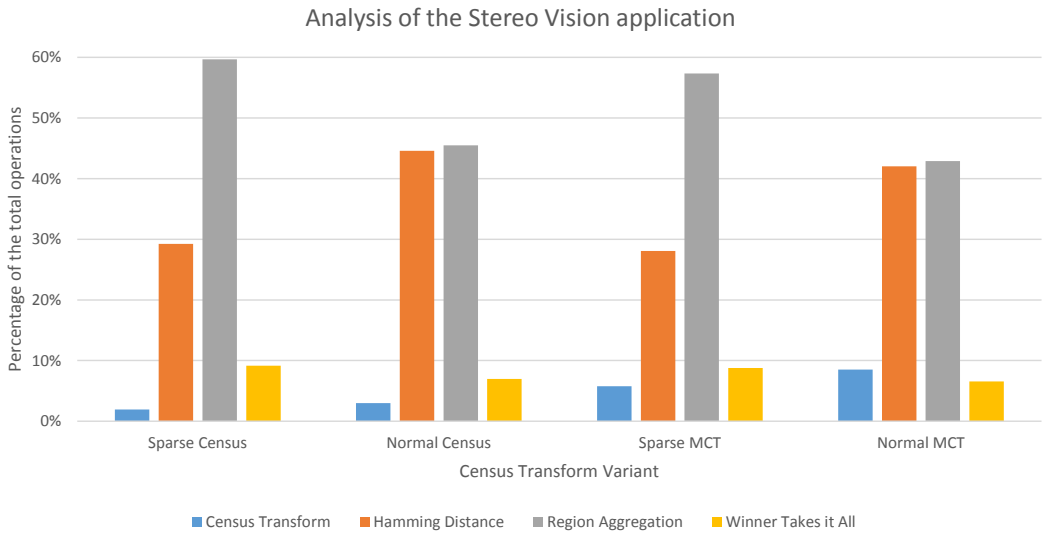


Figure 3.5: Analysis of the different components involved with Stereo Vision.

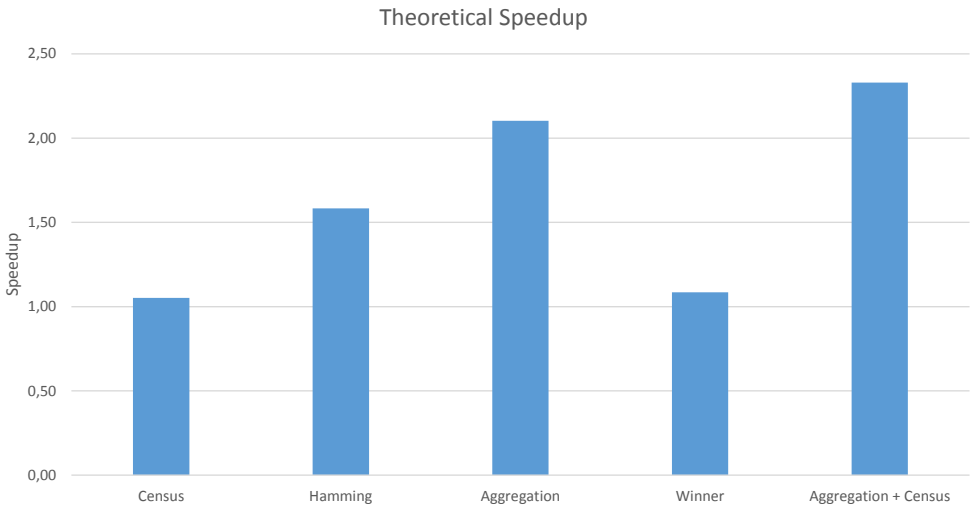


Figure 3.6: Possible speedup when accelerating Stereo Vision components.

3.4 Local Binary Patterns: Face Detection/Recognition

Face Recognition applications using Local Binary Patterns usually perform the following steps:

1. Preprocessing to clean the image.
2. Computing the LBP for each pixel.
3. Extracting local features by computing histograms of LBP; the image is divided into regions, and a histogram is constructed for each region.
4. Classification; comparing the result against a trained set of images.

The LBP used at Intel is the modified LBP (mLBP) with size 5x5; the modified LBP compares the pixels to the average value instead of the actual center pixel. Normally a window larger than 3x3 would require linear interpolation to map the points of the circle to corresponding pixels. However, the application used at Intel simply takes the pixels at the edge, as shown in Figure 3.7. Since the pixels are not directly on the circle, the rotation invariant property of LBP will probably be influenced. However, the advantage is that it does not require linear interpolation, saving computation effort. Besides this, it results in 8 pixels to be compared with the center. Otherwise, it would require more pixels because of the interpolation.

3.4.1 Theoretical Speedup

For the speedup calculation, it is better to not consider the preprocessing steps since it varies a lot on the desired results and the quality of the supplied image. For example, Gaussian Smoothing (convolution) is used to remove noise but this might not be required when the noise is already removed in earlier applications. The number of operations required for the remaining steps are given in Table 3.4, which is calculated for both the original 5x5 LBP and the modified 5x5 LBP. The distance calculating is part of

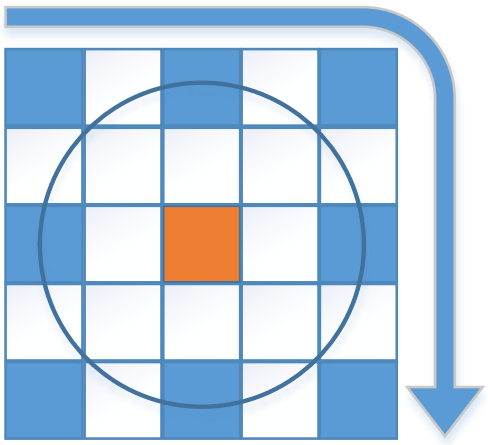


Figure 3.7: Structure of the 5x5 Local Binary Patterns.

the Classification step, but it is included separately because it greatly determines the required amount of processing.

Table 3.4: Operations required (in thousands) for Face Recognition

	LBP	Histogram	Distance	Classification
LBP 5x5	16589	2074	7552	250
mLBP 5x5	49766	2074	7552	250

The values in Table 3.4 are based on an image with a Full-HD resolution (1920x1080 pixels). For each pixel, eight neighboring pixels are compared to the center pixel. With the mLBP, the number of operations is three times higher because it requires one multiplication and one addition for each of the 8 pixels to get the average value. The image is split into 8x8 blocks (resulting in 240x135 pixels for each block), and the histogram is calculated for each block. The number of operations to construct this histogram is equal to the number of pixels since the LBP value of each pixel is used. Because the LBP returns an 8-bit value, the histogram would have values between 0 and 255. However, an optimization is applied to only store uniform values, defined by a pattern containing at most two '0' to '1' or '1' to '0' transitions. For 8-bit values, only 58 patterns are uniform. All the remaining results that do not have this property end up in an additional histogram value. So in total each histogram outputs 59 values. According to [46], these uniform patterns contribute for 90.9% on the tested images, so little information is lost. The histograms are concatenated together to create a large vector. With the 8x8 distribution of blocks, this results in a vector having $8 * 8 * 59 = 3776$ elements. These vectors are now compared with a database of existing images to determine the best match (known as classification). First the distances between the newly constructed vector and the vectors inside the database have to be calculated. Calculating the distance is often done with a "Chi-squared distance" as shown in [47] and Equation 3.3. D is the number of elements in the vector, in this example it is 3776. X is the new vector

which should be compared with a vector Y from the database.

$$\chi^2(x,y) = \sum_{i=1}^{D} \frac{(x_i - y_i)^2}{(x_i + y_i)} \tag{3.3}$$

The total number of operations for the distance measurement highly depends on the number of images in the database since this distance has to be calculated for each vector in the database. The results in Table 3.4 assume a database of 500 images. The size of the database depends on the desired application. For example, in a smartphone it would require only a few images to detect the user. On the other hand when used in automotive it would require the storage of many images to detect objects like cars, persons, and dogs. The final step is the actual classification using a nearest-neighbor classifier. Finding the k nearest neighbors involves sorting, which often has a worst-case complexity of $O(n^2)$. Again, this highly depends on the number of images in the database. The mLBP in Table 3.4 is used for 83,44% of the entire application. This results in a maximum speedup of 6,04x using Amdahl's law.

3.5 Specification and Requirements

Based on the speedup calculations for the Convolutional Neural Network, the 2D Convolution should at least support a kernel size of 7x7 since that is the most computational intensive filter. Ideally a design with a variable kernel (5x5,7x7 and 11x11) should be used to enable an even higher speedup. The Sparse Census Transform and modified Local Binary Patterns should at least support a kernel size of 5x5. The required precision for the 2D Convolution is 16-bit for the image/data plane and 8-bit for the kernel. The resulting output should be 16-bit. The Census Transform also works on a 16-bit input and produces a 16-bit output for the 5x5 window by applying a Sparse Census Transform. Finally the mLBP also works on a 16-bit input and output, since the vector processor uses 16-bit elements in each vector.

The clock frequency in the design depends on applied voltage. Since the accelerator should work in all operation modes, a frequency of 500 MHz should be obtainable. Note that for reliability purposes, the flip-flops will be replaced by Scan flip-flops, potentially adding an extra component in the critical path.

An additional requirement is to explore the current accelerators (Block Matching Accelerator and Bilateral Filter Accelerator) and see if the hardware can be re-used from those accelerators. This way, area and power can be saved as new functionality is added at a reduced cost.

3.6 Metrics

The parameters that we will use to analyze the impact of the new accelerator are speedup, processor utilization, throughput, and area. The speedup is calculated based on the number of cycles that are required to calculate one output pixel, making it independent of the used image resolution. A baseline system without accelerator is used to determine the amount of cycles that are required to perform the operations without acceleration. The

system is then changed to include the accelerator, and the number of cycles per pixel can be obtained. The processor utilization determines how much resources of the processor are used; a low utilization allows the processor to perform other tasks or save power by clock-gating unused resources. The processor utilization is obtained by inspecting how operations are mapped on the available resources. Throughput is obtained by examining the behavior on real-time video processing, using different resolutions. Measuring the area is important because it increases the cost of manufacturing and the size of the chip. Area measurement is obtained by synthesizing the design. Two syntheses are required for this, one without acceleration and one with the accelerator. Synthesis will be performed using 14 *nm* technology. We know that power is also an important metric because the accelerator could be used in mobile devices running on a battery. However, currently the measurement of dynamic power consumption is not possible due to organizational reasons; the tools for measuring the dynamic power consumption are not functional at the moment. Static power (leakage) can be obtained by synthesis, but this is not relevant for the thesis because it is related to the area and does not represent the actual power usage of the accelerator (no switching activity). The design methodology is further described in Chapter 4.

3.7 Conclusions

In this chapter the different filters (2D Convolution, Census Transform and Local Binary Patterns) and their corresponding applications used at Intel are explored. Using Amdahl's law, it is possible to obtain a theoretical speedup when accelerating these filters. The Convolutional Neural Network is dominated by the 2D Convolution; it accounts for 86% of the operations when looking at example data. So accelerating the 2D Convolution for this application makes sense since it is possible to achieve a speedup of 7,3x for the entire application. The second application using 2D Convolution is Gaussian Blur. This application only uses the 2D Convolution and does not require any further processing. The theoretical speedup depends on the window size (5x5, 7x7 or 11x11); the maximum speedup is 30x for the 11x11 Gaussian Blur. For this reason, accelerating the 2D Convolution is worth doing.

The next application is Stereo Vision, which can use both the 2D Convolution and Census Transform. Looking at the number of operations that are required for an example application, it turns out the region aggregation is the most compute intensive. This aggregation can be done with a 2D Convolution, which could result in a speedup of 2,1x for the entire application. Although the Census Transform is not the 'critical' part of the application, it is possible to achieve a total speedup of 2,3x when accelerating both the Census Transform and 2D Convolution. Besides this, it will offload the processor to do other work in the background. Finally Local Binary Patterns are explored, which can be used in Face Detection and Face Recognition. Again, an example application is examined, and it turns out the Local Binary Patterns account for the majority of the operations (83%), this results in a maximum speedup of 6,0x.

All these potential speedups lead to several requirements for the accelerator. The 2D Convolution should at least support a kernel of 7x7, but the highest speedup is possible when also the 5x5 and 11x11 are included. The Census Transform uses a 5x5 sparse

method by skipping certain pixels, reducing compute and communication demands. The LBP uses a modified 5x5 variant that compares 8 pixels with the average value. An additional requirement is that it should synthesize on 500 MHz, and the area overhead can probably be limited by investigating the reuse of existing accelerators. Finally, the metrics that we will use to analyze the impact of the new accelerator are presented. These are speedup, processor utilization, throughput, and area overhead.

Current Architecture and Approach

4

This chapter describes the current architecture, including the available hardware accelerators that might be available for reuse. The chapter ends by describing the approach; including the tools and languages used.

4.1 Intel Image Processing Unit (IPU)

The Image Processing Unit (IPU) contains multiple Hardware and Firmware (Software) components. It is a complex system-on-chip containing programmable processors, accelerators, camera interface components, Memory Management Units (MMU) and many more components. An overview of the current IPU is given in Figure 4.1. The IPU contains an Image Signal Processor (ISP) Processing System, which includes the Vector Processors. The ISP is also the part where the current accelerators and the accelerator designed in this thesis can be found.

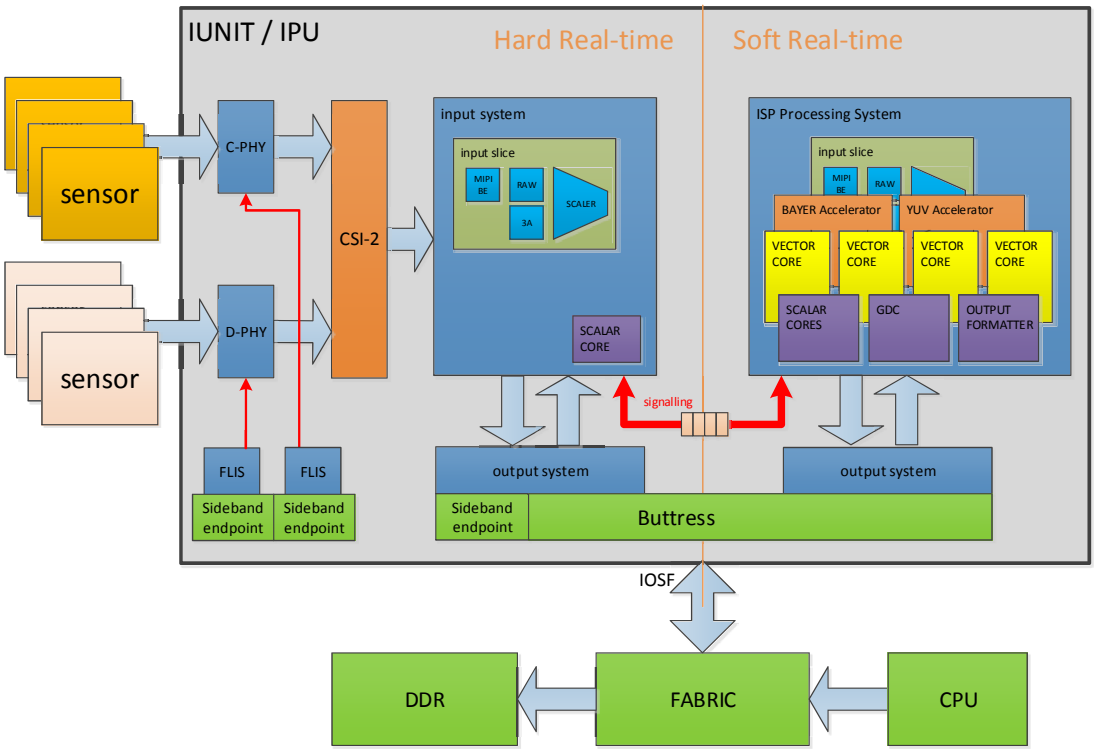


Figure 4.1: Image Processing Unit (IPU) from Intel. [48]

4.2 Available hardware designs

The current architecture of the ISP contains an accelerator called the Block Matching Accelerator and Bilateral Filter Accelerator (BMA / BFA). This accelerator features a 'sliding window' technique by moving a square window across the image and performing a certain calculation at each position. This is similar to the concept of 2D Convolution and Census Transform, so the design of the new accelerator could potentially be based on this unit. The accelerator is designed to speed up the computation for image quality improvement without sacrificing power consumption and performance. The current BMA is mostly used for Advanced Noise Reduction (ANR) and motion tracking in a video while the BFA completes the BMA for noise reduction and other image quality improvements. The BMA and the BFA are quite similar, so the functions are combined into a single accelerator for maximum reuse of hardware.

4.2.1 Block Matching

The primary function when using the accelerator for Block Matching is to compare a block of 8x8 pixels with the image, finding the best match. Each comparison goes through a Sum of Absolute Differences (SAD) and is stored in an output element for the user to analyze. The region of interest has a size of 16x16 pixels (which is part of the image). Since each vector has 32 elements, this consists of eight vectors containing 8x4 pixels. The 'search area' can be used to take a portion of this region of interest to select 14x14 pixels or the whole 16x16 region of interest. This search area will be matched to an 8x8 pixels reference block (consisting of two vectors/blocks). The reference block shifts around the Search Area and at each position it calculates the difference to find the best-matching part of the search area. The possible number of candidates are:

$$\text{Number of Candidates} = \left(\frac{\text{Search Area width} - 8}{\text{Pixel Step}} + 1\right)^2 \tag{4.1}$$

Pixel step is used to skip pixels and reduce the number of candidates; this will improve the speed at the cost of a decrease in accuracy. This results in a big difference between the fast 14x14 search area with a Pixel Step of 2 and the full 16x16 Search Area with a Pixel Step of 1. The BMA unit also contains an option for Masking; this influences the shape of the search area. The elements of interest are selected with a binary vector. This masking can be used to, for example, obtain a diamond shape or a checkerboard pattern as Reference element.

4.2.2 Bilateral Filter

The accelerator is also capable to accelerate bilateral filter operations. A bilateral filter is a non-linear, edge-preserving and noise-reducing smoothing filter [49]. The idea is that each pixel in the resulting image is calculated by a weighted average of values from nearby pixels of the original image. The weight can be a Gaussian distribution and can be changed for each filter. Sharp edges are preserved since the weight also depends on differences like color intensity and the depth distance.

4.2.3 Current implementation

Both the BMA and BFA operate on a large amount of data and apply similar computations on the elements. For this reason, the architecture is designed to combine the BMA and BFA together and reuse hardware when possible. See Figure 4.2 for an overview of the architecture.

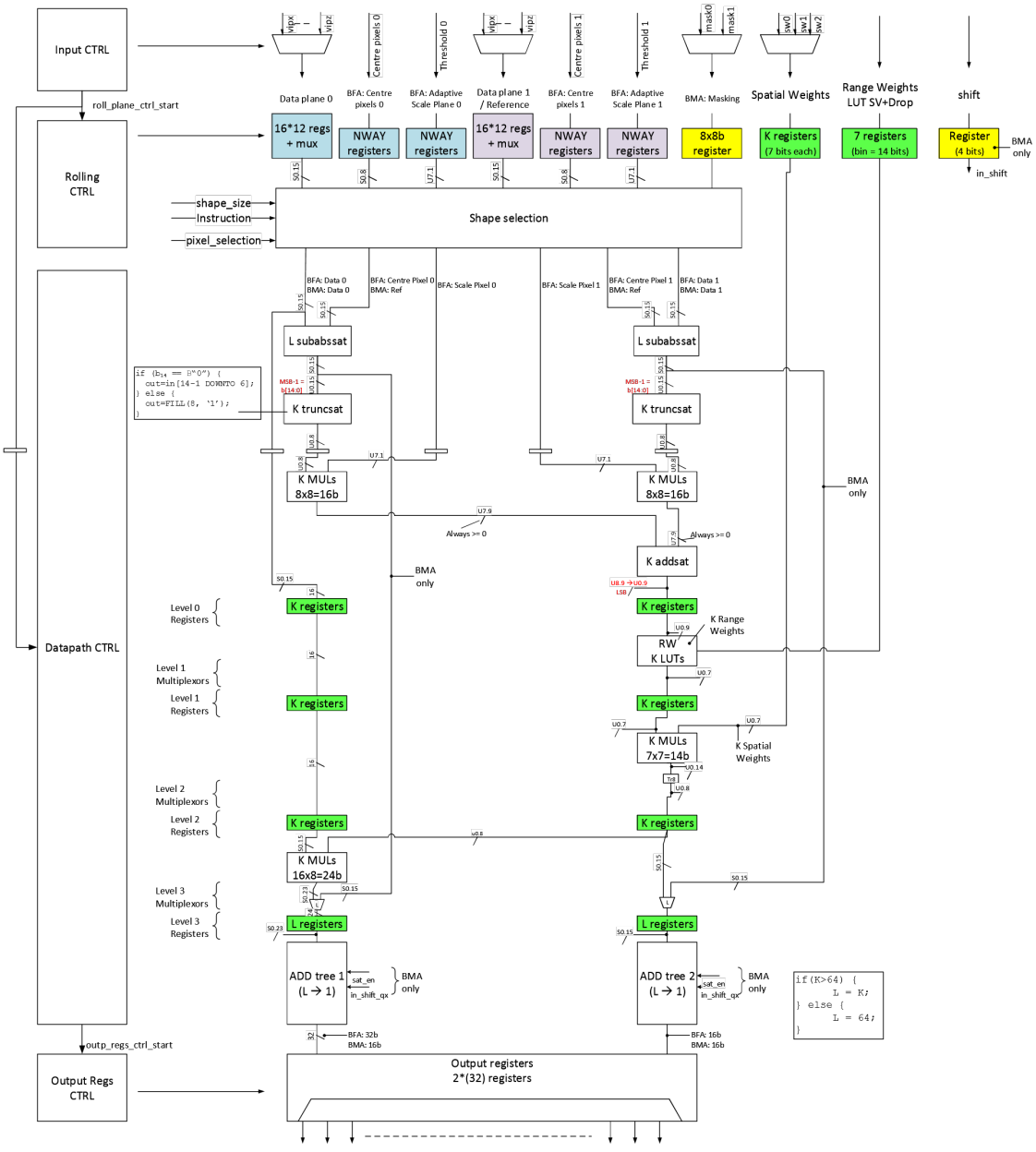


Figure 4.2: Overview of the current BMA / BFA architecture from Intel. [50]

The architecture consists of four controllers:

1. Input controller
2. Rolling Plane controller
3. Datapath controller
4. Output register controller

The input controller loads the data from the input ports (vectors and scalar data) to the appropriate registers. The input requires multiple vectors to load a 12x16 plane; each vector contains 32 elements of 16 bits. The rolling plane controller moves the data around, by shifting the data horizontally or vertically. This means that the plane itself moves while the position of the grid is fixed. The rolling controller also contains a Shape Selection component to select data from the top-left corner of the plane, change it if necessary (selecting the relevant pixels) and forward it to the data path. The data path controller controls the arithmetic operations and data path. This data path has two slices that can run in parallel, for BMA this calculates two candidates in parallel and skips certain arithmetic parts that are only used by the BFA. The output registers are used to store data for the output vectors. The number of components shown in Figure 4.2 depends on the parameters K and L. In the current implementation K = 61 and L = 64.

For the possible hardware reuse, it might be worth investigating the available registers. The registers used in the BMA/BFA are summarized in Table 4.1. Clearly there are many registers available with different sizes.

Table 4.1: Registers available in the BMA/BFA.

Name	Number of Registers	# Vectors	Precision [bits]
Data Plane 0	12*16=192	6	16
BFA: Center pixels 0	32	1	8
BFA: Adaptive Scale Plan 0	32	1	8
Data Plane 1	12*16=192	6	16
BFA: Center pixels 1	32	1	8
BFA: Adaptive Scale Plan 1	32	1	8
BMA: Masking	8*8=64	2	1
Spatial Weights	61	-	7
Range Weights LUT SV+Drop	7	-	14
Shift	1	-	4
Computation Level0+1+2 Left	183	-	16
Computation Level3 Left	64	-	24
Computation Level0 Right	61	-	10
Computation Level1 Right	61	-	8
Computation Level2 Right	61	-	9
Computation Level3 Right	64	-	16
Output	2*32=64	2	16

The calculations performed in the data path depend on the type of operation (block matching or bilateral filtering). The BMA only performs the `subabssat` operation; this consists of a subtraction, absolute value, and saturation. The accelerator contains 128 of these units, each supporting signed 16-bit values on the input. After this step, the results are added together in an adder tree. There are two adder trees in the current implementation; the left one adds 64 elements of 24-bit together, and the right adder tree is designed to add 64 numbers of 16-bit together. The BFA operation also performs `truncsat` operations (truncation followed by saturation), addition, look-up and multiplication. Especially the multiplication is interesting since this is heavily used in the 2D Convolution. The available multipliers are summarized in Table 5.5 and will be further described in Section 5.3.2.

4.3 Approach

Finally, it is worth mentioning the methodology used at Intel for building a new accelerator. It includes a proprietary Hardware Description Language called Configurable Hardware Description Language (CHDL) and a language called The Incredible Machine (TIM). TIM is a Machine Description Language and specifies how the processor should be constructed. Using TIM, it is possible to create entirely different systems specific for a desired application. CHDL has a syntax which is similar to that of synthesizable VHDL; the compiler actually generates VHDL from the CHDL description. There are some differences in the syntax. For example, CHDL is case sensitive, and all the reserved keywords are written in capital letters. CHDL has the following advantages (compared to VHDL):

- Ports (input/output) can be created conditionally
- Guard statements
- External function calling
- Multidimensional Arrays

The big advantage of CHDL is its ability to create highly configurable hardware. Specifically the Guard statement is useful; it allows the user to generate hardware unique to the situation. For example, a product requires a 'lite' version of some filter to save hardware and power usage. This can be passed as a parameter using the TIM language. This is illustrated in the following example:

```
SIGNAL wire_data : BIT;
WITH (p_have_feature) GUARD BLOCK
    -- Some Logic
    wire_data <= '1';
END BLOCK;
WITH NOT(p_have_feature) GUARD BLOCK
    -- Some Logic
    wire_data <= '0';
END BLOCK;
```

When `p_have_feature` is passed to the CHDL code, it will generate the statements within the Guard statement and assign the value '1' to the signal. Otherwise, it will assign the value '0'. Note that this is different from using a normal `IF/ELSE` since it determines if the logic is generated in hardware or not. A normal `IF/ELSE` would result in a multiplexer being constructed to select the correct data.

It is also possible to directly work with multidimensional arrays, as shown in the following example:

```
CONSTANT c_add_bits : D2<INTEGER>;
FOR row IN 0 TO (12 - 1) GENERATE
    FOR col IN 0 TO (16 - 1) GENERATE
        c_add_bits{row}{col} := 16;     -- Create 16-bit array
    END GENERATE;
END GENERATE;
SIGNAL wire_add : D2<BITVECTOR>(c_add_bits);
```

This creates a 2D matrix with size 12x16, and each element is 16-bit. Now the array can be used with a simple index (e.g. `wire_add{0}{0}` will access the first element).

The TIM language has two usages; specification of what a functional unit should do (known as the semantics) and the building of a core by connecting these functional units together with Issue Slots. When creating a new Functional Unit the TIM language is also used to define a time shape, this specifies the data input and output for each cycle. The semantics is used for simulations and to see if the RTL matches the functionality defined in the semantics. A simple example of the semantics:

```
OP my_mul <signed nway>(svecN<nway> A, B) -> (svecN<nway> R) {
    <DOC>
        <SHORT> Signed 16-bit vector multiplication</SHORT>
        <SEM> R[i] = A[i]*B[i] </SEM>
        <DESCRP> Performs a simple multiplication.</DESCRP>
    </DOC>;

    SEM R <nway>(A,B) = {
        for i [(nway - 1), 0] {
            R[i] = (signed <Width := 16>) A[i] * (signed <Width := 16>) B[i];
        }
```

```
    };
};
```

In this example, a new operation called `my_mul` is defined with two vector inputs and one vector output. The number of elements in a vector is defined by `nway` and can be passed as a parameter when using the operation. The `DOC` is used for documentation purposes (e.g. automatic manual generation) and uses a XML style of coding. This is purely for documentation and does not define the actual functionality; this is done with the `SEM` code. In this case it specifies how the output R should get a value based on the input, it is possible to have multiple outputs and specify the behavior for each output individually. Now the new operation has to be included in a Functional Unit:

```
FU my_fu_unit <signed nway>(Port ~ vip0 , Port ~ vip1) -> (Port ~ vop0) {
    // Time shape
    cycle[0] = {vip0, vip1};
    cycle[1] = {vop0};

    // Available operations
    my_mul : vop0 = my_mul(vip0, vip01);
};
```

Finally, this Functional Unit can be included inside a core and the entire system.

4.3.1 Test method

Testing if the accelerator provides the correct result will initially be done by visual inspection. Additionally, the result can be compared to the output obtained with a C-version of the program and see if each pixel value matches exactly. More tests can be performed like a full white or black image. Finally, it is possible to generate a random test input and let that be executed by the semantics defined in TIM as well as the accelerated version and compare the output. The tools at Intel support this last method. Ideally the final design should be tested on an FPGA to verify the correctness in the real world. Unfortunately, due to organizational reasons it is not possible to perform this test during the time of this thesis work. The FPGAs are located in a different country, and they are normally only used at the last stage when integrating the entire system. However, the test method described here ensures that the design should work in all use cases, and the FPGA is only required for a final check. When synthesizing the design, equivalence checking is used to check the functionality of the synthesized hardware against the desired RTL functionality.

4.4 Conclusions

In this chapter, the architecture of the Intel Image Processing Unit is explored. This system contains the Image Signal Processor, including the vector processing and accelerators. The existing BMA and BFA accelerator is explored in more detail since several of their components could be reused for the new accelerator. This analysis includes the

loading of input data, handling the input data and performing the actual computations. The chapter ends with an overview of the methodology used at Intel. Several languages are introduced: TIM and CHDL. TIM is used for describing the expected behavior of a functional unit and for connecting components in a core. CHDL is similar to VHDL but is aimed to provide configurability. This is achieved with the introduction of guard statements.

5 Design

This chapter describes the design of the new accelerator. There are two options for the accelerator: building a new one or reusing parts from the existing accelerators. These two options are described in Section 5.1. The chapter then presents an overview of the design for the 2D Convolution. After this, Section 5.3 describes the 2D Convolution in more detail; specifically how it is implemented and what parts of the current accelerators are reused. Then in Section 5.4 an overview of the Sparse Census Transform is provided and Section 5.5 describes the actual implementation. Finally, Section 5.6 describes how the Modified Local Binary Patterns can be implemented by combining the Census Transform and the 2D Convolution.

5.1 New accelerator or extending the current accelerator

The 2D Convolution involves many multiplications. To ensure the additional area is limited, it is worth investigating the reuse of the multipliers inside the current accelerator. However, reusing a multiplier will require a multiplexer to switch the inputs. For this reason, first the difference in area between adding a new multiplier or a multiplexer should be investigated. The results are given in Table 5.1; the numbers are given as 'standard cells'. The 121 component column is to indicate the size when it used for an 11x11 Convolution, which requires 121 multipliers. Clearly using a multiplexer instead of a multiplier will be better in terms of area. Note that adding a multiplexer in the critical path could potential influence the clock speed. For this reason, the new accelerator will be an extension of the current Block Matching Accelerator and Bilateral Filter Accelerator (BMA/BFA).

Table 5.1: Area difference between different components.

	One Component	121 comp.	Increase
Multiplier 16-bit	184	22264	46x
Adder 16-bit	33	3993	8x
Register 16-bit	29	3509	7x
Mux 2:1 16-bit	4	484	1x

Of course, there are more aspects when making this decision. For example, creating a new accelerator provides more freedom to implement the filters. Additionally a new accelerator could achieve a higher speedup since you are not bound to the implementation of the multipliers and adders. However, the area difference between a multiplier and multiplexer is significant; creating a new accelerator only makes sense when the functionality of the original accelerator is no longer required. So to summarize; the idea

is to design an accelerator that supports the original BMA/BFA functionality and is also capable of performing the 2D Convolution, Sparse Census Transform, and the Modified Local Binary Patterns. This way, the new accelerator has a lot more functionality while the hardware increase is limited.

5.2 Overview of the 2D Convolution

The greatest speedup for the Convolutional Neural Network can be obtained when a variable window size for the 2D Convolution is employed (supporting both 5x5, 7x7 and 11x11). This variable window can be constructed by combining the results from smaller windows. An example of this is shown in Equation 5.1, where "I" is the input image and K is the kernel. Here a 7x7 convolution (which requires 49 multiplications and additions) is split into two "5x5" units. The first unit computes 25 of the 49 elements, and the second unit computes the remaining 24 elements where the last input is put to zero to fill the input to 25 elements. For this reason, the initial design features several "5x5" windows that can be combined together to form a larger convolution kernel. See Figure 5.1 for the corresponding structure, where two 5x5 kernels can be combined to one 7x7 and an 11x11 can be formed by using multiple 5x5 units.

$$\sum_{i=0}^{48} I_i K_i = \sum_{i=0}^{24} I_i K_i + \sum_{i=25}^{48} I_i K_i \tag{5.1}$$

As shown in the previous example, the number of multiplications and additions are not always a multiple of 25. To still achieve the correct result, the unused inputs should be set to zero. This means that not all available hardware is being used. The utilization is calculated in Table 5.2 for each kernel size. The sizes between parentheses are not necessary for the accelerator but are included in the table for reference. As can be expected, the 5x5 has a hardware utilization of 100% because the design is based around this window size. It can be observed that the utilization is above 97% for all kernel sizes except the 9x9 kernel size that has a utilization of 81 %. Currently, the 9x9 is not required for the Convolutional Neural Network (see the example of Figure 3.2). Splitting the kernels to multiple "5x5" units looks like a good choice since it offers the flexibility to change the kernel size while sharing the same hardware.

Table 5.2: Utilization when constructing large windows from multiple 5x5 windows.

Size	Nr. 5x5 Required	Utilization
5x5	1	100%
7x7	2	98%
(9x9)	4	81%
11x11	5	97%
(13x13)	7	97%
(15x15)	9	100%

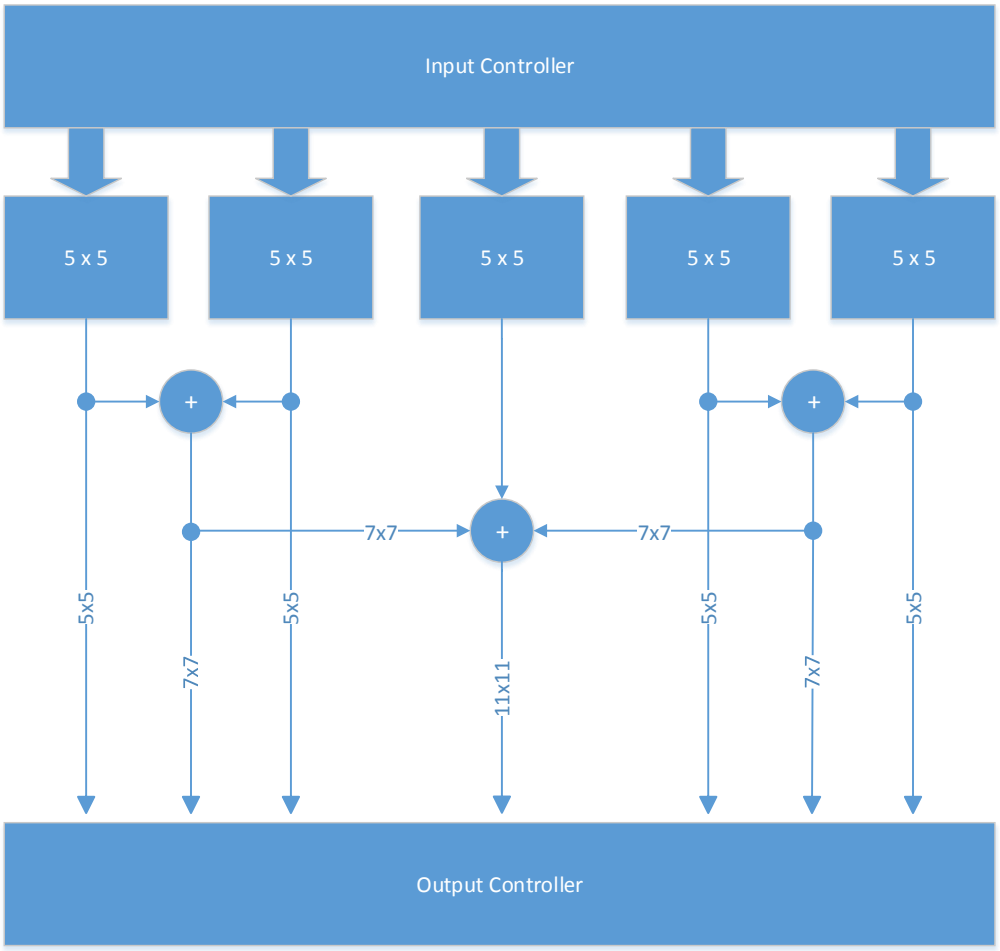


Figure 5.1: Initial design of the 2D Convolution accelerator, showing multiple "5x5" windows that can be combined to form larger kernel sizes.

5.2.1 Input/Output size

The current accelerators (BMA/BFA) work on a 12x16 pixels input. Since reuse of components is important to reduce added area, the same size is used for the 2D Convolution. See Table 5.3 for an overview of the different convolution operations. Note that the 5x5 operation works on a 10x12 pixels input since using a 12x16 pixels input would result in more than two output vectors. This situation should be avoided in order to limit the extra area that would be required for the extra interconnect (the current BMA/BFA supports two output vectors). Additionally, it would require an extra cycle to transfer the data. The 7x7 operation can be used on both the 10x12 and 12x16 pixels input, this allows the user to make a trade-off between speed (latency) and the amount of results that are available. Loading the 10x12 input requires four vectors while the 12x16 input requires six vectors; so loading data is faster on the small version. Besides this, the 10x12 input would require fewer cycles to perform the actual computation since there are fewer outputs to be calculated. The option of a 7x7 on a 12x16 input is currently

not implemented, but this could be added in the feature. For this reason, it is shown with parentheses in the table.

Table 5.3: Different operation modes for the 2D convolution

Kernel Size	Input [dimension]	Output [dimension]	Output [vectors]
5x5	10x12	6x8	2
7x7	10x12	4x6	1
(7x7)	12x16	6x10	2
11x11	12x16	2x6	1

5.2.2 Fixed Point representation

The vectors inside the processor use 16-bit to represent numbers. Fixed-point representation is required to handle fractional numbers. A fixed point number $Qm.n$ consists of m bits for the integer part and n bits for the fraction. When multiplying two fixed point numbers the resulting output has the following format:

$$Qm.n * Qx.y = Q(m + x).(n + y) \tag{5.2}$$

Similarly for addition the output has the following format:

$$Qm.n + Qm.n = Q(m + 1).n \tag{5.3}$$

Note that for addition the two input numbers must be in the same format. To achieve this, the operand with the lowest number of fraction bits needs to be shifted right before performing the addition. Also note the increase of 1 for the integer part, this is because otherwise the addition can result in an overflow.

The fixed point representation used for the 2D Convolution is shown in Table 5.4. The input data is in Signed Q15.0 (16 bits) and the kernel is in Signed Q1.6 (8 bits). After multiplying one pixel from the data with one value from the kernel it results in a Signed Q16.6 number. For the 11x11 convolution 121 of these values have to be added together; when using an adder tree this requires:

$$\frac{log(11 * 11)}{log(2)} = 6.92 = 7 \text{ stages} \tag{5.4}$$

For this reason, the result after the addition has seven additional bits for the integer part, resulting in Signed Q23.6. However, the output vectors are 16-bit, so this will not fit. For the output to be reused immediately as a new input to the accelerator, the fixed point representation of the output is chosen to be Signed Q15.0. Note that this reduction from Q23.6 to Q15.0 could result in an 'overflow' if the original value required more than 15 bits for the integer part. To solve this, the output can be saturated to the maximum value if this occurs. Furthermore rounding can be applied to handle the removal of the fractional part. Currently, these options are not implemented and left as future work, note that both these options will slightly increase the area. The value needs

to be saturated when the OR function of the bits above bit 15 results in a '1'. Rounding up is needed when the first bit of the fractional part is '1'; indicating a 0.5 or higher. This round is done by adding the value 1 to the result, requiring an adder. This can also result in a carry moving all the way through the number, which could create an overflow. So it is important first to apply the rounding and then the saturation.

Table 5.4: Fixed Point representation for 2D Convolution

	Signed	Bits integer	Bits fraction
Input Data	Yes	15	0
Input Kernel	Yes	1	6
(*Multiplication*)	Yes	16	6
(*Addition*)	Yes	23	6
Output Data	Yes	15	0

5.3 Implementation of 2D Convolution

As mentioned before, the reuse of hardware that is already inside the current accelerator is important to limit the additional area that is required. In this section, the (potential) reuse of the input controller and its selection of data from this input is explored. Furthermore, the reuse of multipliers and adders from the current accelerator is explored. Finally, the output storage is described.

5.3.1 Input Controller and selection of data

The rolling plane from the BMA/BFA (described in Chapter 4) is used to provide new values for the shape selection, which selects a certain window of pixels (e.g. 7x7) from the top-left corner of the rolling plane. An alternative would be to select directly the pixels from the 10x12 or 12x16 input. However, the reason for the rolling plane in the current accelerator is because otherwise it would require huge multiplexers to select the input pixel to use. This would give problems with timing and also an increase in area, so for this reason the reuse of the rolling plane is logical. The original rolling plane works on an input of 12x16 (requiring six input vectors), but the 2D Convolution should also support a 10x12 input (requiring four input vectors). See Figure 5.2 for the difference between the construction of a 10x12 input and 12x16 input using vectors. Note that the 12x16 input stores the data in blocks of 4x8 pixels while the 10x12 input uses blocks of 10x3 pixels. There are two ways to correctly reuse the rolling plane; on the host the 10x12 plane could be expanded into a 12x16 by adding zeros. This way the hardware can be reused entirely, but it requires the transmission of six vectors instead of four. The other possibility is to transmit the 10x12 plane and let the hardware feed the input to the 12x16 rolling plane. The structure of vectors is different (like shown in Figure 5.2), so this solution would require some additional registers to temporally store the 10x12 plane until it is completely received. The transmission of two extra vectors will increase the number of cycles that the accelerator requires, lowering its potential speedup. For this

reason, it has been implemented on hardware where the received 10x12 plane is given as input to the 12x16 Rolling Plane. The area will slightly increase by using registers to store the four vectors of 16-bit elements.

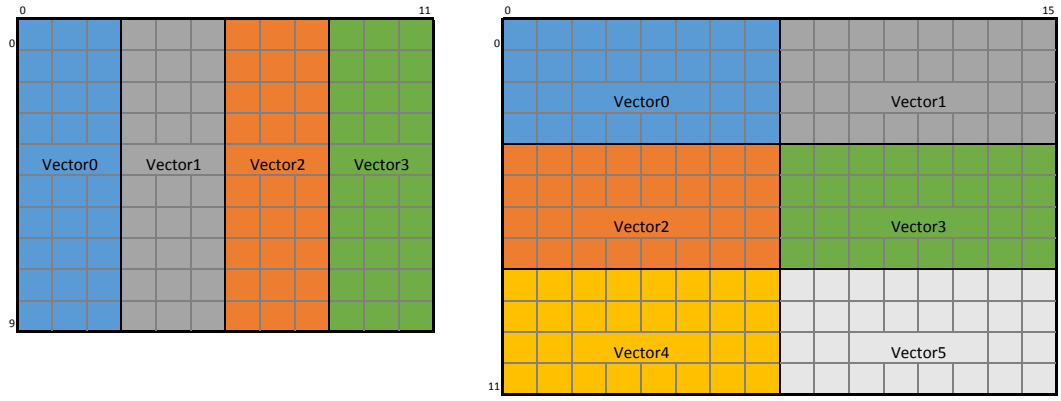


Figure 5.2: Loading a 10x12 input or a 12x16 input, using vectors of 32 elements.

The next step is to roll the plane in the correct way, so the shape selection receives the proper data. This involves shifting the input in a certain direction with a certain step size, depending on the desired operation (5x5, 7x7 or 11x11). The different options are shown in Figure 5.3. Note that four 5x5 convolutions or two 7x7 convolutions can be done in parallel, this is represented with the different colored arrows. Because of this, the rolling plane for the 5x5 convolution has to move down first, and then jump with a step of 4 and move up again. Similarly for the 7x7 convolution it jumps with a step of 2.

The top-left corner of the rolling plane is used to select either four 5x5 planes, two 7x7 planes or one 11x11 plane, depending on the desired operation. The BMA/BFA currently selects a single 7x7 or 9x9 plane, so the other options have been added to select multiple windows in parallel.

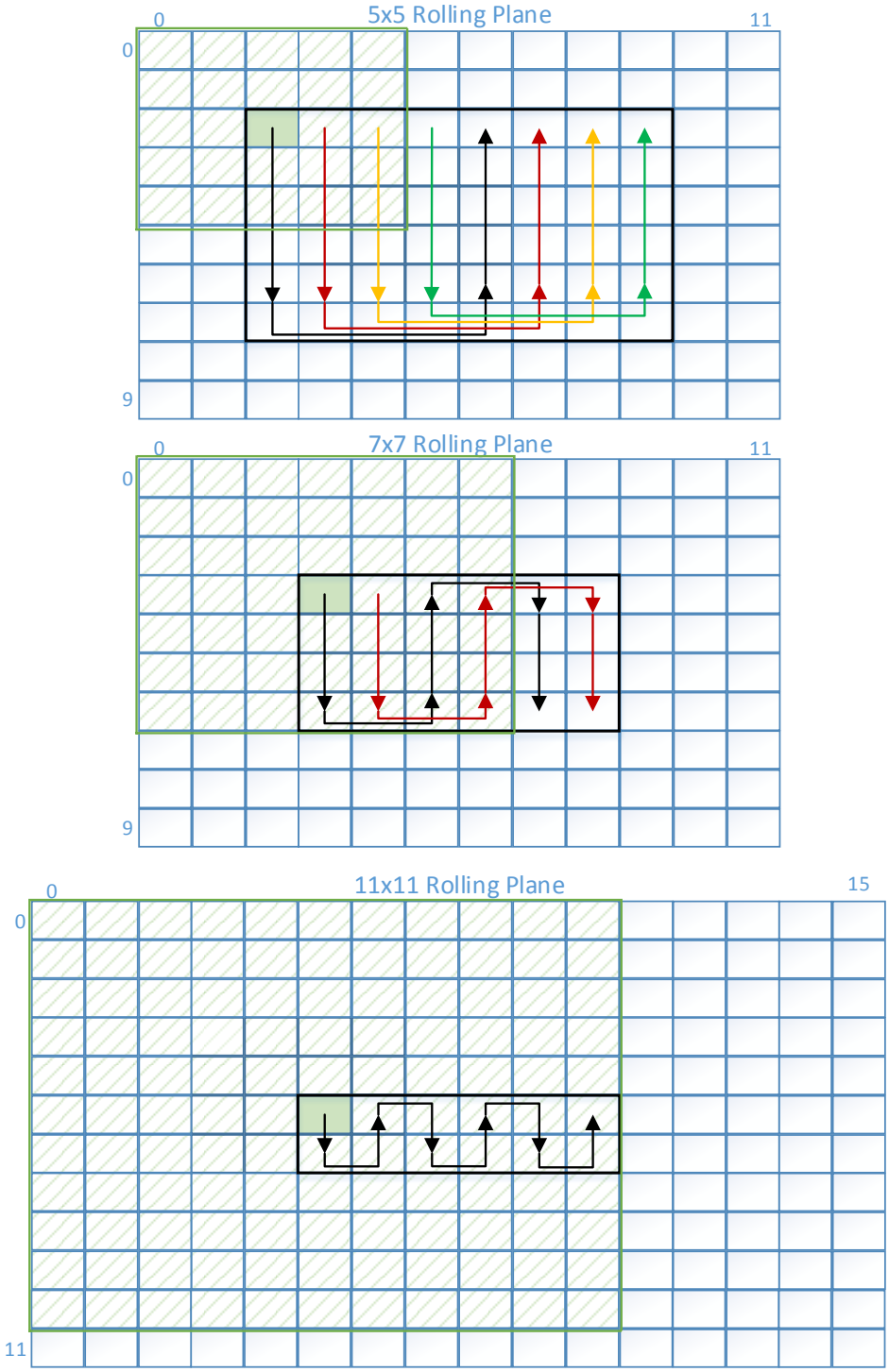


Figure 5.3: Rolling plane movement for the 5x5, 7x7 and 11x11 2D Convolution. Parallel arrows indicate parallel computation.

5.3.2 Multiplication

As explained in Section 5.1, it is worth reusing the multipliers in the current accelerator since the area difference between a multiplier and multiplexer is around 46. See Table 5.5 for an overview of the multipliers in the BMA/BFA. To support a kernel size of 11x11, 121 signed multipliers of 16x8 bits are required since the data is 16-bit, and the kernel is 8-bit. Half of those are directly available in the BMA/BFA, but to reuse the rest of the multipliers will require some modifications.

Table 5.5: Multipliers inside BMA/BFA

Amount	Signed/Unsigned	Bits In1	Bits In2	Bits Out
122	Unsigned	8	8	16
61	Signed	16	8	24
61	Unsigned	7	7	14

Building a wide multiplier from narrower ones is done with a Divide-and-Conquer technique described in [51]. The basic idea is to multiply a part of the number (e.g. 8 of the 16 bits) and add the results together. The construction of a 16x8 bits multiplier from two 8x8 bits multipliers is shown in Figure 5.4. Here the 8-bit number is multiplied with both the upper-half and lower-half of the 16-bit number and these are finally added together. Note in theory the addition can result in a carry to the outside, but because the input is 16-bit multiplied with 8-bit it can never exceed 24-bit.

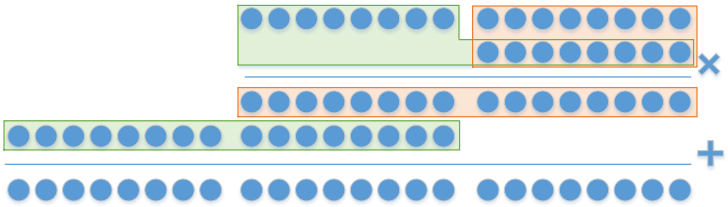


Figure 5.4: 16x8 bits multiplication using two 8x8 bits multipliers.

The next challenge is to support signed numbers. As shown in Table 5.5 the 8x8 bit multipliers are unsigned while the 16x8 bit multiplier requires signed numbers.

There are two ways to handle signed numbers; changing the multipliers or changing the input and output. The first way is to change the 8x8 bits unsigned multipliers in signed 9x9 bits multipliers; increasing the size of each multiplier. The increase to 9x9 bits is needed to support the same unsigned numbers as in the 8x8 bit multiplier. An example of building a 4x2 bits signed multiplier from two 2x2 bits multipliers is shown in Figure 5.5. Notice that the one of the smaller multipliers performs a multiplication between an unsigned number and a signed number. The other multiplier performs multiplication between two signed numbers.

The second way to handle signed numbers is by performing a 2's complement conversion of the input number if it is negative (determined by the sign bit). By taking the 2's complement, the input will become positive, and the original multipliers can be used

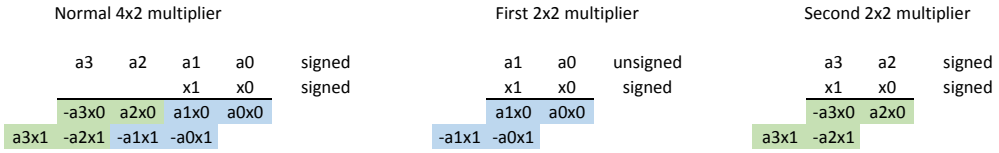


Figure 5.5: Large signed multiplier using smaller multipliers.

without modification. The output might need to be converted as well if the output is expected to be negative. This can be determined by taking the `XOR` of the input sign bits. If the `XOR` is '1' (1 of the inputs is negative), the output needs to be converted by using a 2's complement. When the `XOR` is '0' (the inputs are both positive, or they are both negative), the result of multiplication does not require correction. Even recent papers like [52] use the technique of converting the 2's complement of the input and converting the output if necessary.

Note that the conversion to (and from) 2's complement requires an addition (inverting the bits and adding '1'), this increases the area. So both options would require extra hardware: either by creating larger signed multipliers or because of the 2's complement conversion. The difference between the two options is estimated to be small. The second option where the input and output are converted is chosen, since this guarantees to not break the existing functionality.

In summary; 121 multipliers of 16x8 bits are required, to support this all the 61 Signed 16x8 bits multipliers from the BMA/BFA are fully reused. The remaining 61 will be created by using the unsigned 8x8 bits multipliers, two of those are required for one 16x8 bits multiplication. So all 122 unsigned 8x8 bits multipliers will be used. This leaves 61 unsigned 7x7 bits multipliers from the BMA/BFA that are not reused at all.

5.3.3 Addition

After each pixel in the data plane is multiplied with the corresponding pixel in the kernel, the results have to be added together to obtain a single value. The current accelerator features two adder trees that can be used for this purpose. Both adder trees perform an addition of 64 values. The left adder tree supports 24-bit for the input values while the right adder tree supports 16-bit for the input values. The output of the left tree is 32-bit, and the right tree outputs 16-bit values. Both adder trees support shifting the output and saturating the output.

To use these adder trees for the 2D Convolution, they need to be modified. The result after multiplication is 24 bits, which means the right adder tree needs an extension from 16-bit to 24-bit and the output of the second tree needs to be 32-bit. Besides this modification, both trees need to be modified, so they output the intermediate result after the addition of 32 values. This way, the results from two 5x5 units can be added in parallel on a single adder tree. See Figure 5.6 for the modified adder tree, showing the extra outputs to add the intermediate 32 values together in the color green. Note these extra outputs do not support the shifting and saturation options since in the original adder tree these are only applied after all 64 numbers are added.

To support the 11x11 kernel size using the initial design from Section 5.2, it would

require an additional adder tree for the last "5x5" result. However, the current adder trees are already capable of adding 128 values together (while 11x11 requires 121 values to be added together). So it is better just to map the 121 values directly on the adder trees and add the result of the two adder trees together to obtain the 11x11 value. This final addition takes place inside the output controller (the green adder in Figure 5.6).

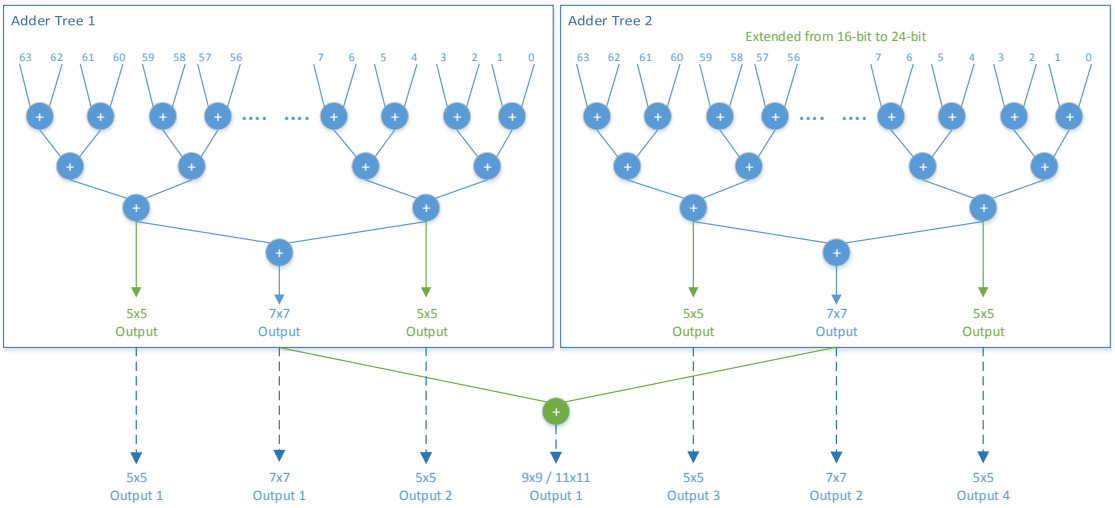


Figure 5.6: Modification of the Adder Tree.

5.3.4 Output

To obtain the correct output, the results after addition need to be stored at the correct location. The output follows the movement of the Rolling Plane shown in Figure 5.3. For the 5x5, four values are available each cycle. The 7x7 gets two values each cycle and the 11x11 processes one cycle. These values need to be stored temporarily until all values are calculated, and the output vector can be created. The 7x7 can be implemented almost identical to the BMA/BFA since the values come directly from the two adder trees. To support the 5x5 and 11x11, the output controller has to be modified. The adder trees now produces four intermediate outputs to support the four 5x5 results; these need to be added as extra input to the output controller. When the 5x5 instruction is executed, it has to select these inputs instead of the regular adder tree. Besides this, it has to store four values to the registers instead of two. The final output is stored as two vectors since there are 48 values available (one vector supports 32 values). The left half is stored in the first vector while the right half is stored in the second vector; splitting the 6x8 output to two 6x4 outputs. The 11x11 requires a modification to store the final result since the outputs from the two adder trees are still stored separately. These two outputs are added together, and the final output is created.

Finally, it is worth mentioning that the output from the adder tree has more bits than the output of a vector (described in Table 5.4). To enable direct usage of the output as a new input to the next convolution, the 16 bits from the integer part are taken. Currently, this is done by directly taking these bits, without applying any rounding, saturation or

clipping. A future improvement could be to clip the output. When the value is larger than 15-bit, it should check if it is negative or positive (sign bit) and adjust the output to either the maximum or minimum value.

5.4 Overview of Sparse Census Transform

Similarly to the 2D Convolution, support for 5x5, 7x7 and 11x11 is investigated. As mentioned before in Section 3.3, the main reason to choose a Sparse Census Transform is to reduce the amount of outputs. Especially when using a larger window size like 7x7 or 11x11, the output would require many bits (48-bit and 120-bit for the 7x7 and 11x11 respectively). These outputs need to be further processed by the processor, which will be slower than when using a sparse census transform. Furthermore, the accelerator would be forced to work on a (much) smaller input to reduce the number of output values.

For the Census Transform, it is also possible to build a large version out of smaller versions, but instead of an addition it needs to 'stitch' the bits together. For example, a sparse census transform of 7x7 requires 24 comparisons with the center pixel and a sparse 5x5 requires 12 comparisons. The 7x7 can thus be split into two 5x5 units when providing the same center pixel to both.

See Table 5.6 for the different operation modes for the Sparse Census Transform. The same dimensions for the input and output are used as the 2D Convolution; for consistency reasons and to limit the additional area that is required to handle other sizes. A sparse 5x5 requires 12 bits, but the data is represented using a 16-bit number, so it requires one output element. Similarly, the 7x7 requires 24 bits and two output elements of 16-bit; the 11x11 requires 60 bits and four output elements of 16-bit. When multiplied with the resulting output dimension, every operation results in 48 outputs elements of 16-bit. So every operation requires two vectors to store the results.

Table 5.6: Sparse Census Transform operation modes.

	Input [size]	Output[Bits]	Outputs[16b]	Output[size]	Output[16b]
5x5	10x12	12	1	6x8	48
7x7	10x12	24	2	4x6	48
11x11	12x16	60	4	2x6	48

5.5 Implementation of Sparse Census Transform

This section describes the actual implementation of the Sparse Census Transform. Similarly to the 2D Convolution, it is possible to implement a 5x5, 7x7 and 11x11 version. Currently, only the 5x5 version is implemented in CHDL. The advantage of implementing this version over the others is because it can be used for the Local Binary Patterns described in Section 5.6. However, the design of the 7x7 and 11x11 are included for future reference.

5.5.1 Input Controller and selection of data

The 10x12 input designed for the 2D Convolution and described in Section 5.3 can also be used for the Census Transform. The rolling plane is slightly different and shown in Figure 5.7; the reason is because it is now possible to process more values at a time. The main calculation the Census Transform has to perform is to compare the value of a pixel with the value of the center pixel. To perform this comparison, the `SUBABS` units inside the BMA/BFA can be reused. These units perform a subtraction followed by the absolute value. Two values A and B are subtracted; A is smaller than B if the result is negative. So to use these for the Census Transform, the sign bit of the result after subtraction has to be stored; bypassing the absolute value calculation. There are 128 `SUBABS` units available, each supporting signed 16-bit values on the input. The 5x5 requires 12 values to be compared, the 7x7 requires 24 values, and the 11x11 requires 60 values to be compared. So in theory ten 5x5 results could be computed in parallel, but using all these would influence the hardware for the rolling plane, shape selection, and the output controller. This is because it would not be possible to shift the plane in the same manner as shown in the first image of Figure 5.7. Data from the row below the current row would already be computed since it has eight columns instead of 10. For this reason, the 5x5 will compute eight results in parallel; making the rolling plane a simple movement upwards each cycle. The 7x7 requires 24 comparisons, so 5 of these could be done in parallel using the available hardware. However, since the 5x5 performs 8 in parallel, only four 7x7 can be done when using the modular approach (two 5x5 combined to get one 7x7). So this is why there are four arrows in parallel shown in Figure 5.7. Note that it does not matter for the final execution time if indeed all five would be computed in parallel since there are six columns. It would still require the computation of the last column (arrow pointing upwards). The 11x11 Census Transform requires 60 values to be computed, so two can be done in parallel.

Besides changing the rolling plane, the shape selection has to be changed. Note that a Sparse Census Transform requires a checkerboard pattern to be selected, and it does not include the center pixel in the selection. To perform a checkerboard pattern, a fully hard-coded solution could be used (e.g. `out[0]=in[0]; out[1]=in[2];`). A more elegant solution is to generate these using a modulo-2 operation. When the position inside the 5x5, 7x7 or 11x11 window (where the position is defined as `row*size+col`) is a multitude of 2, the value should be used. The center pixel should not be included in the output, but it is a multitude of 2. This has been handled by checking if the position (`row*size+col`) is smaller or larger than the position of the center pixel. Unfortunately the CHDL language does not support a modulo operation to generate the desired signals, so this has been implemented with:

$$a \bmod n = a - n * \left\lfloor \frac{a}{n} \right\rfloor) \tag{5.5}$$

The floor function is performed with an integer division, which rounds down the result (discarding the fractional part). It is worth mentioning that this modulo operation only influences the generation of VHDL to select the right elements, no actual hardware is build for this modulo computation. The advantage is that the same code can be used for any window size. Besides the storage of these pixels, also the corresponding center pixels

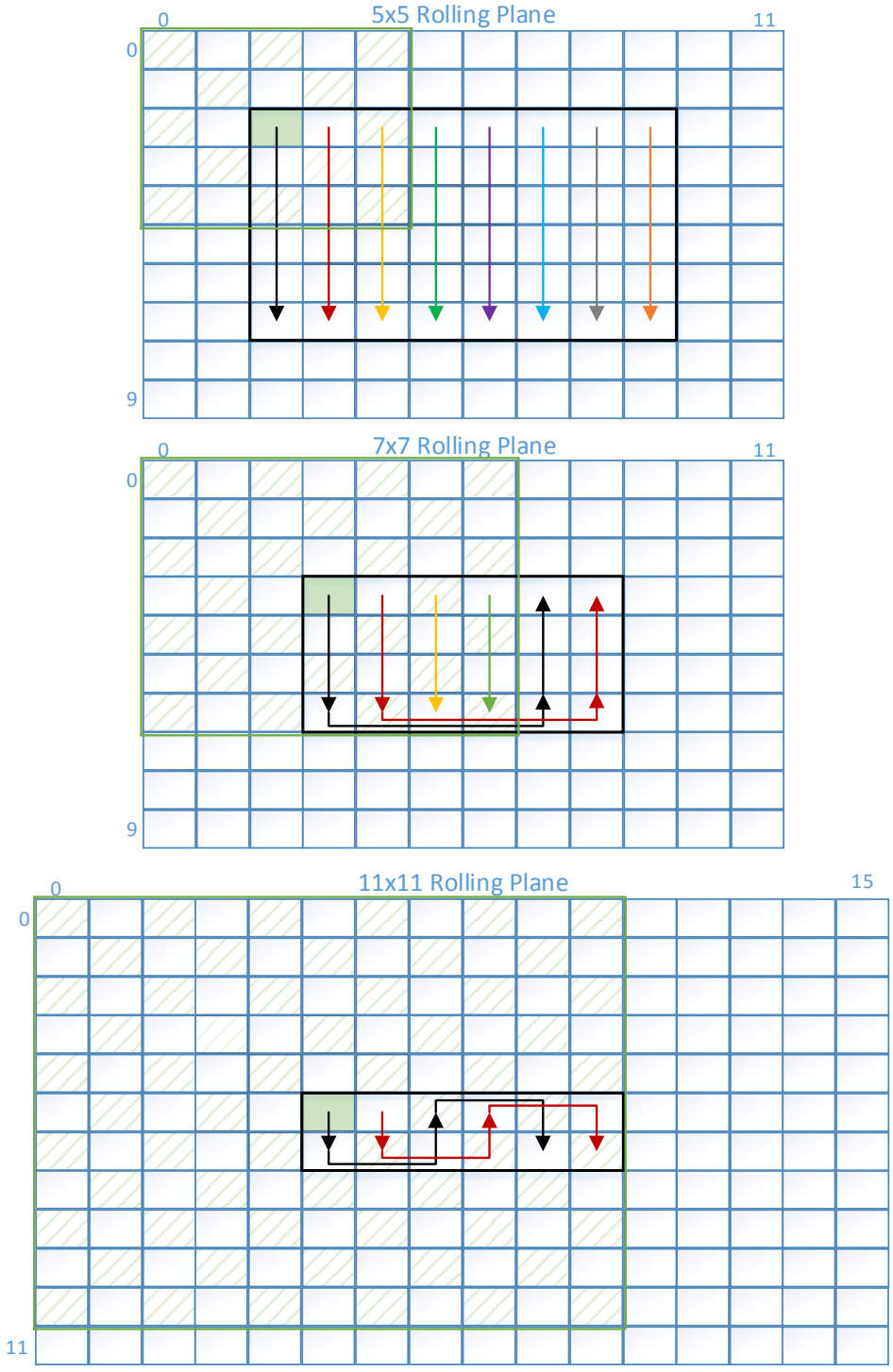


Figure 5.7: Rolling plane movement for the 5x5, 7x7 and 11x11 Census Transform. Parallel arrows indicate parallel computation.

are stored. So the shape selection now outputs the two values that should be compared with each other, this output is already available so it will not increase hardware.

5.5.2 Computation and storage of the output

The shape selection outputs both the values to be compared to the center pixel and the center pixel itself. These are fed into the SUBABS units. As mentioned before, the sign bit of the intermediate result after subtraction is stored (in a signal named `wire_cencus_gr_eq`) to indicate if the value is larger or equal to the center pixel. These values are then combined to one result (`wire_census_out_data`) and extended to 16-bit so they can be stored in the output registers. The output controller is modified again since it now has to handle more results each cycle. More importantly; it has to select the census output instead of the adder tree output. As with the 2D Convolution, the 6x8 output is split to two 6x4 outputs for the 5x5 window.

5.6 Overview of modified Local Binary Patterns

The mLBP requires the calculation of the average value to which the actual values inside the window will be compared to. To support this functionality, different options are available:

1. Compute the average value on the host and transmit this to the accelerator for the comparison.
2. Compute the average and the result on the accelerator, where the 2D convolution is used to compute the average value.

The first option gives the user a bit more flexibility because the user can provide any value as 'average'. For example, you can manually compute the average value on a modified image (e.g. an image with enhanced contrast) and use this average value on the original image with the mLBP. The second option can be a lot faster since everything is done on hardware. See Figure 5.8 for the design of the second option, where the 2D convolution is reused to compute the average value for the center pixel. The parts shown in green color is the added hardware, a multiplexer will select either the average value or the original center pixel. The original center pixel is used for the Census Transform; the average value is used for the modified Local Binary Patterns. Also, it is worth mentioning that as shown in the literature survey, there is a Modified Census Transform which can be added in a similar way since it uses the average value. The option shown in Figure 5.8 is chosen for the implementation on the accelerator since the hardware for the 2D convolution is already available. This will result in a fast mLBP, tightly integrated with the Convolution and Census Transform (all sharing the same hardware). Note the kernel used in Figure 5.8 shows only the value '1' which is required for a classical mLBP, but the user can provide a different kernel if required. For example, the user can use this to compute the average using all pixels inside the 5x5 windows (instead of just the pixels used for mLBP) or give more weight to certain pixels. Although not as flexible as the first option, it still provides sufficient flexibility to change the average calculation.

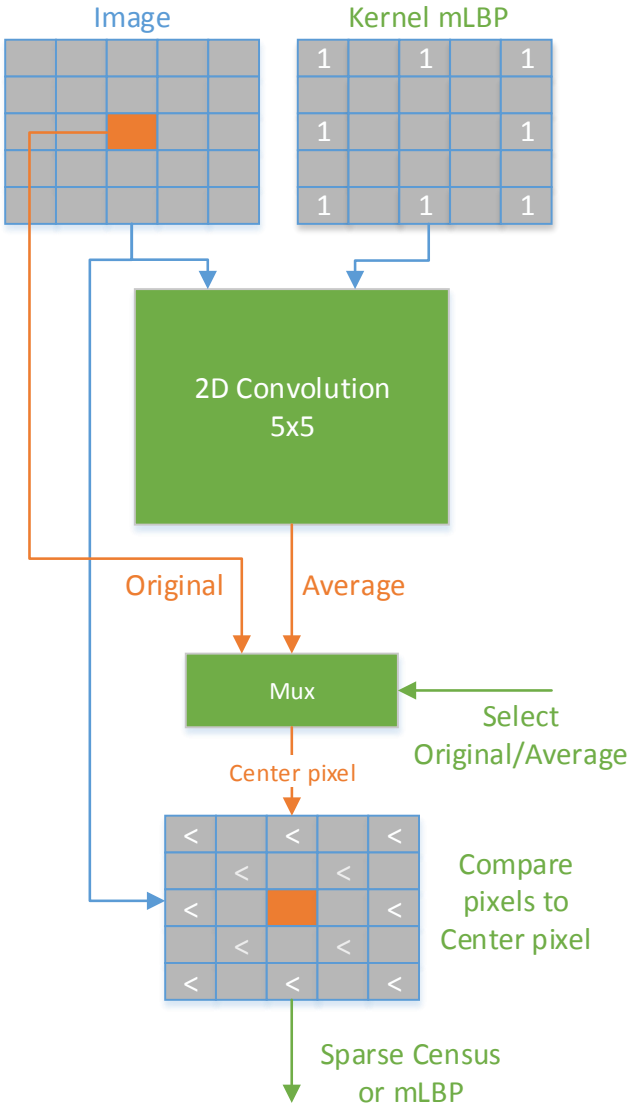


Figure 5.8: Design of the modified Local Binary Patterns acceleration.

Due to the limited time available, priority to 2D Convolution and 5x5 Census Transform is given. For this reason, the actual mLBP is not implemented. However, the concept of combining the already designed 2D Convolution and Census Transform results in an accelerator for mLBP with a limited additional area.

5.7 Conclusions

In this chapter, the design of the new accelerator is presented. It is possible to create a completely new accelerator or base the new accelerator on an existing accelerator, extending the functionality and limiting additional area. Regarding the design of the 2D Convolution, we pointed out that it requires many multiplications. One can implement a new 16-bit multiplier or reuse an existing multiplier plus a 16-bit multiplexer. The area required for a 16-bit multiplier is 46x larger than a 16-bit multiplexer, so an extension of the original accelerator is proposed. As concluded in Chapter 3, a higher speedup can be obtained when accelerating multiple window sizes (5x5, 7x7, and 11x11). Different window sizes have been implemented by combining multiple smaller windows (5x5) together.

We have to modify the original accelerator to support different input planes and different window sizes. The original accelerator supports an input size of 12x16 pixels, which can be reused for the 11x11 window size. However, for the 5x5 option a smaller input size is required. For this reason support for a 10x12 is implemented. The 7x7 could work on both input options, but currently it is implemented for a 10x12 input since it requires fewer data to be transmitted to the accelerator. The 10x12 plane is temporarily stored in registers and provided as a 12x16 plane to the Rolling Plane component. This component moves the data around so that each time new data is available in the top-left corner. The top-left corner is used to select the desired window (5x5, 7x7, or 11x11). The original accelerator only allows to use a single 7x7 window, support for the selection of four 5x5 windows, two 7x7 windows, and one 11x11 window is added. Both the input data and output data feature the same 16-bit Fixed Point representation so multiple 2D Convolutions can be applied in series, that is useful for tasks like the Convolutional Neural Network which requires many successive 2D Convolutions. The input for the kernel data is 8-bit.

Furthermore, the 11x11 option of the 2D convolution requires 121 multipliers. Each multiplier should support the multiplication of a signed 16-bit number with a signed 8-bit kernel value. The original accelerator only has 61 of these multipliers available. Since the area difference between a multiplier and multiplexer is huge, it is worth reusing the smaller multipliers as well. One 16-bit times 8-bit signed multiplier is constructed by combining two 8-bit times 8-bit unsigned multipliers. This is done by performing a 2's complement on the input numbers if they are negative and correct the output if a negative result is expected (determined with the XOR of the two sign bits).

After multiplication, the 24-bit results have to be added together. This is done by reusing the adder trees. The current accelerator features one adder tree for 24-bit data and another for 16-bit data. This second adder tree is extended to 24-bit to support the addition of all required numbers. Additionally, both adder trees are modified to provide an intermediate result so multiple 5x5 convolutions can be done in parallel.

The design for the Census Transform also supports a window size of 5x5, 7x7, and 11x11. However, only the 5x5 version of the Census Transform is currently implemented. This size is the most important since it can also be used for the Local Binary Patterns. The loading of the inputs is the same as with the 2D Convolution (support for the 10x12 plane). Selecting the data from the top-left corner is slightly different since it requires

a checkerboard pattern. The selection of the right pixels is generated with a modulo operator and is scalable to different window sizes, useful for future work. To compare the pixels with the center pixel, the subtraction units are reused.

Finally, the design of the modified Local Binary Patterns (mLBP) is presented. The mLBP requires pixels to be compared with the average value instead of the actual center pixel. This average value can be computed by reusing the newly designed 2D Convolution accelerator. Comparing the pixels to this average value can be done by reusing the Census Transform accelerator. So accelerating the mLBP combines both new accelerators together and extends the functionality even further.

6 Results

This chapter describes the results obtained in this thesis. Section 6.1 describes the different metrics and how these metrics are obtained. The metrics are speedup, processor utilization, throughput, and area. In Section 6.2 the simulation results are presented, containing several figures showing the important signals in the design. Section 6.3 presents the obtained results for the 2D Convolution using the different metrics. Similarly, Section 6.4 presents the results for the Census Transform. Section 6.5 presents the area results for the new accelerator, and Section 6.6 describes the trade-off between using multiple instances of the accelerator.

6.1 Metrics

The two main metrics for this thesis are Speedup and Processor Utilization. The speedup is based on the amount of cycles that are required to calculate a single output pixel. This value is independent of the used clock speed and is useful for calculating the required amount of cycles for an entire image. It is worth mentioning that memory accesses are performed in the background and thus, potentially, do not influence the speedup. The processor uses Direct Memory Access (DMA) to get data from external memory. This data is temporarily stored in the processor for processing. While the processor is performing the actual computations, new data can already be loaded. So, in theory, memory only plays a role at the start and finish of the program. However, when the required amount of cycles for data transfer is larger than the actual computation, it will limit the speedup.

The processor utilization indicates how much of the processor is being used, this can be obtained by inspecting the resources (like the adders and multipliers) that are being utilized. A low utilization allows the processor to use its remaining resources for a different application. Additionally, it can be used to reduce the dynamic power consumption (the values are not changing by employing clock-gating). To calculate the processor utilization for the original processor, it is first required to know how exactly the operation will be mapped on the different Issue Slots (IS) of the Intel Image Processing Unit. An Issue Slot can be described as a one or more Functional Units (e.g. vector addition, and the new accelerator) combined and sharing a local memory (Register File). The different Issue Slots are connected using a bus. The processor is a flexible system where more compute power can easily be added if desired by adding additional Functional Units (which is, of course, a trade-off between speed, area, and power). Additionally, the number of Issue Slots can be changed; currently this is set to six. The processor utilization with acceleration can be obtained by determining the structure of the Issue Slots again. However, this time the processor can process other tasks while waiting for the results of the accelerator. The processor is only needed to load and store the

data; this determines the processor utilization. Data transfers from the external DDR memory to the processor can be done with one vector per cycle, and transfers from the local registers to the accelerator can be done with two vectors per cycle.

It is also worth looking at how the accelerator would perform at real-time processing by calculating the throughput. This can be done by employing the accelerator for video processing. Two parameters are important here: the resolution and the number of frames per second. The resolution is varied between 1080p (1920x1080 pixels), 4K (4096x2160 pixels), and 8K (7680x4320 pixels). Next, the frames per second are obtained. The number of cycles that are required to calculate one output pixel is reused. Now multiplying this with the desired resolution gives the total amount of cycles for one frame of the video. Dividing the total cycles by the clock frequency of 500 MHz results in the time that is required to process a single frame. Now by taking the reciprocal we obtain the number of frames that can be processed in one second (known as Frames per Second (FPS)).

Finally, the area results for the new accelerator are determined. The new accelerator is based on an existing accelerator. The advantage is that functionality is extended while area overhead is limited. Two syntheses are performed to determine the increase in area for the accelerator itself. One synthesis is to determine the area for the original accelerator, and the other is for the new (extended) accelerator. Additionally, the new accelerator is placed in the Image Processing Unit to determine the area increase on a system level.

6.2 Simulation Results

As mentioned before in Section 4.3.1, the design is tested using random numbers. For a consistent test, the results should be the same each time the design is tested. The test input is therefore created by randomly generating a 12x16 plane with 16-bit values and use this as a fixed input. This way, the input contains both negative and positive values. Part of this input is also used for the 10x12 plane. Similarly, the 11x11 kernel is generated with random values of 8-bit and the 5x5 and 7x7 are derived from this.

First, the Rolling Plane is tested to see if it correctly provides new data every cycle. Looking specifically at the 7x7 Convolution, it should follow the movement as shown in Figure 6.1. The arrows first move down for three cycles, followed by a jump right with a step of two, and then move up. The rolling plane moves the data plane itself, so all directions are inverted; when the selection of the cell below the current cell is required, it moves the data upwards.

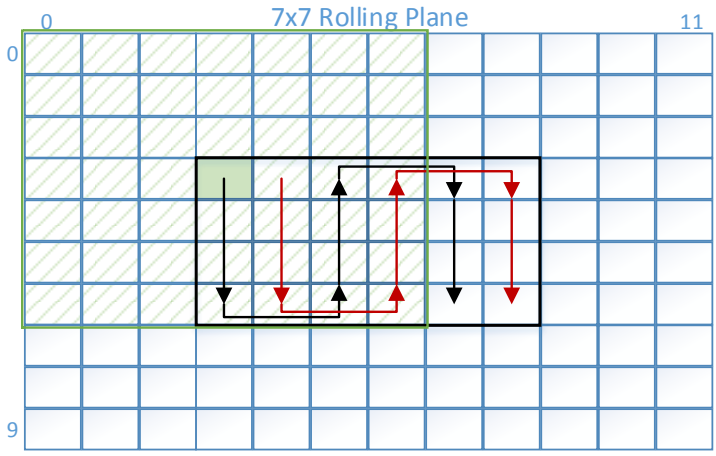


Figure 6.1: Rolling Plane movement for the 7x7 2D Convolution. Image repeated from Chapter 5.

The actual results for the rolling plane can be seen in Figure 6.2, obtained by simulating the design with Cadence SimVision. It shows the results when using the 7x7 2D Convolution, but the 5x5 and 11x11 are similar, and all follow the movement of the arrows as shown before in Figure 5.3. The output plane is a 3-dimensional matrix; the first dimension is to support multiple rolling planes, but this is not used for the 2D Convolution and Census Transform. The second dimension is the vertical axis of the plane and the third dimension of the horizontal axis. As can be seen, data from register `I0_I1_I0` (row 1, column 0) indeed moves to register `I0_I0_I0` (row 0, column 0) one cycle after the `UP` instruction is given and data from `I0_I0_I2` (row 0, column 2) moves to `I0_I0_I0` (row 0, column 0) when the `LEFT` instruction is performed with a step of two. Note that the value of the `in_step_sel` signal shows the value one smaller than the desired step, since a step of 0 is not required and would require an extra bit. Figure 6.2 also shows the 7x7 shape selection at the bottom; it selects a 7x7 block from the top-left corner of the Rolling Plane. The `data_I0_I0` till `data_I0_I48` is filled with these values, the remaining ones (until `data_I0_I63`) are set to zero. Not shown on the figure is the second output, where `data_I1_I0` till `data_I1_I48` is filled with a 7x7 block shifted 1 pixel to the right to perform two convolutions in parallel (the red arrow in Figure 6.1).

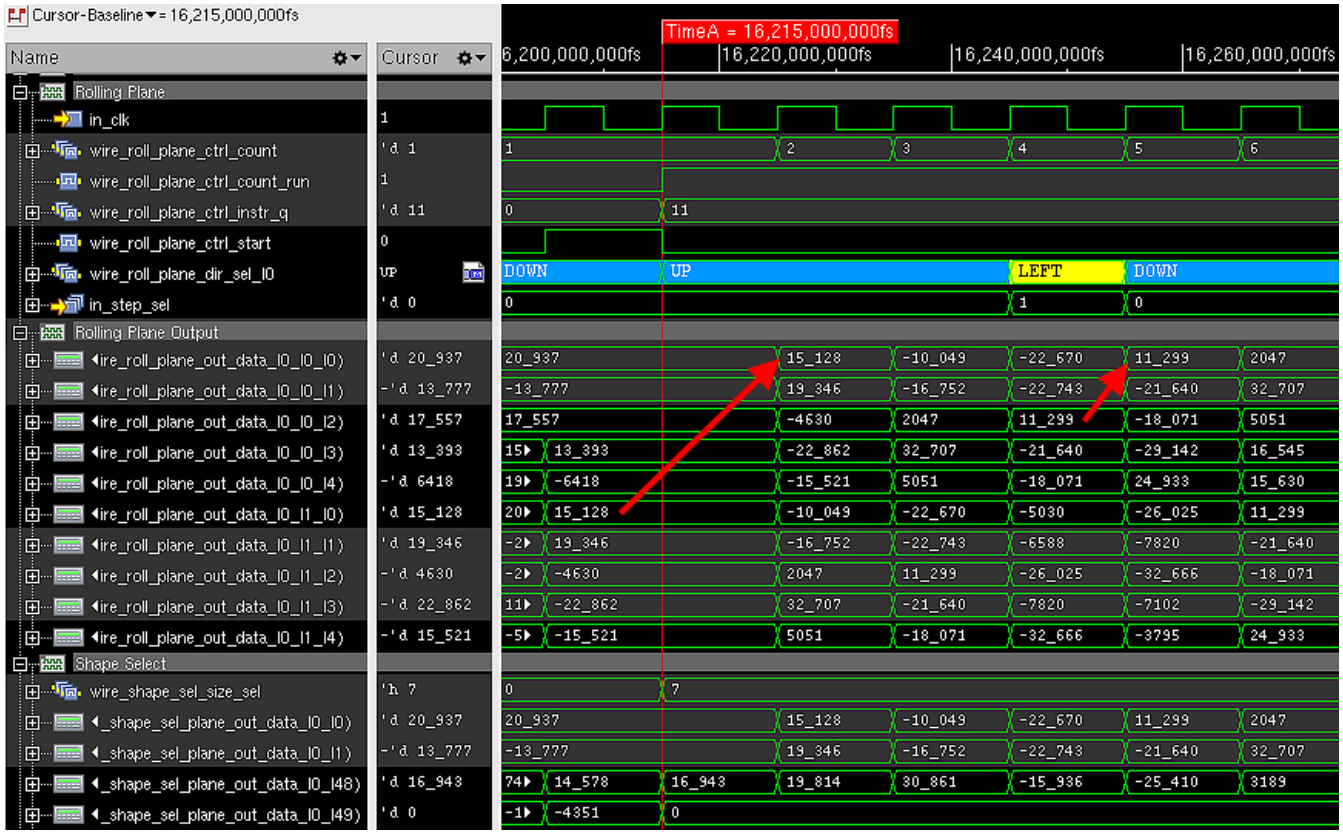


Figure 6.2: Simulation of the 2D Convolution Rolling Plane and Shape Selection (7x7 size).

As explained before in Chapter 5, there are not enough multipliers available to directly perform the 16-bit (input data) times 8-bit (kernel data) multiplication. Therefore, two 8x8-bit multipliers are combined in a single 16x8-bit multiplier. The actual process is explained before in Chapter 5. Figure 6.3 summarizes how the 16x8-bit multiplication can be constructed using two 8x8-bit multipliers.

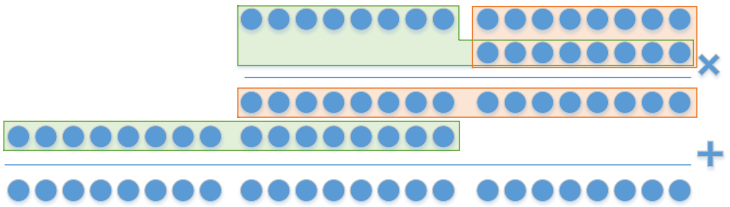


Figure 6.3: 16x8 bits multiplication using two 8x8 bits multipliers. Image repeated from Chapter 5.

The simulation results of this process are shown in Figure 6.4. The top 8 bits and bottom 8 bits are separately given as an input to a multiplier (grouped under input 1) while the 8 bits from the kernel are repeated for both multipliers (grouped as input 2). For example, the first output from the shape selection is `20.937`, which is `01010001 11001001` in binary. When converting the top 8-bit to decimal results in `81`, the bottom half results in `201`. Note the `5x5` idea, described in Chapter 5, is used here. This makes it possible to create a large window from multiple smaller windows. For example, a 7x7 Convolution can be done with two 5x5 Convolutions. The first 25 elements of the `7x7` are processed in the first dimension (`I0_I0` till `I0_I49`) and the second 25 in the second dimension (`I1_I0` till `I1_I49`), each requiring 50 8x8-bit multipliers. Since the `7x7` requires 49 multipliers, the last 8x8-bit multipliers in the second dimension are set to zero. As mentioned before in Chapter 5, handling of signed numbers is done by performing a 2's complement conversion on the inputs to obtain only positive inputs and also convert the output if necessary to obtain the correct result. For example, the first multiplication is $20.937 * -77$, which needs to be converted since it contains a negative input. In the waves of Figure 6.4 this can be seen, since the value -77 is now converted to 77. The result of multiplying the top-half bits (81) with the kernel (77) is 6.237, the bottom-half results in 15.477. This results in the value 1.612.149 when combined. Combining the results together is done with:

$$\text{Result} = Out_0 * 2^8 + Out_1 \tag{6.1}$$

Since one of the inputs was negative, the output needs to be converted as well. This can be seen in Figure 6.4, where the output (`multoutput_I0_I0`) is negative. Since both outputs are 16-bit, the numbers still overlap after the left-shift of 8; requiring an addition for combining the results. The final output is 24-bit because the original multiplication was a 16-bit number with an 8-bit number.

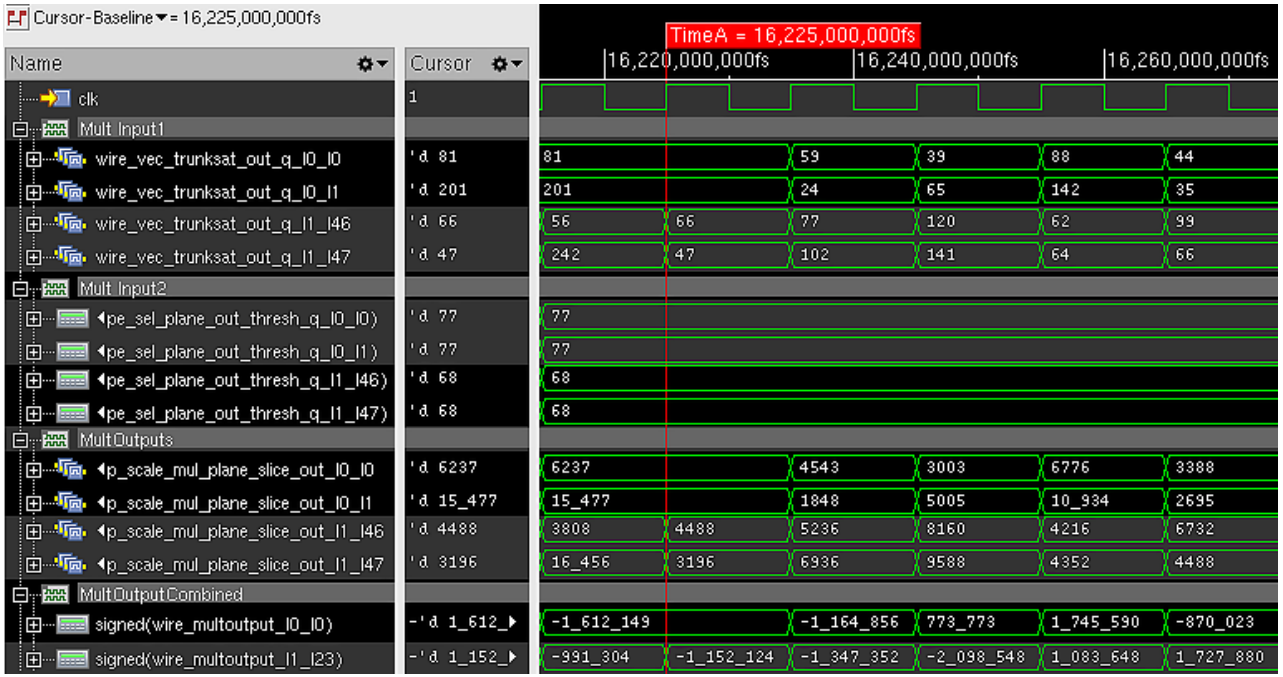


Figure 6.4: Simulation of the 2D Convolution, combining two 8x8 multipliers to create one 16x8 multiplier.

6.3 2D Convolution

The new accelerator is capable of performing both the 2D Convolution and Census Transform. This section describes the results for the 2D Convolution using the different metrics: speedup, processor utilization, and throughput. Finally, we compare the obtained speedup with the theoretical speedup for the different applications.

6.3.1 Speedup

First the cycles per pixel are calculated for the original processor. The original processor will calculate 32 results in parallel; the number of cycles to get these results will depend on the desired window size (5x5, 7x7 or 11x11). 2D Convolution involves multiplication followed by addition; this is known as a Multiply Accumulate (MAC) operation. The Intel Image Processing Unit is capable of calculating 32 of these each cycle when properly pipelined. The amount of MAC operations that are required is calculated by multiplying the number of output pixels with the window size (for example $32 * 5 * 5 = 800$). Since it is possible to calculate 32 MACs every cycle, this result is divided by 32, and we obtain the amount of cycles that are required to get these 32 output pixels. Now by dividing this by the number of outputs (32), the cycles per output pixel are obtained. The results for the original processor are given in Table 6.1.

Table 6.1: Cycles required on the original processor.

	Output Pixels	MAC / cycle	MAC req	Cycles Req	Cycle / Pix
5x5	32	32	800	25	0,78
7x7	32	32	1568	49	1,53
11x11	32	32	3872	121	3,78

The accelerator is capable of giving 48, 24 or 12 results after 24, 25 or 27 cycles respectively; depending on the desired Window Size. This immediately results in the cycles per pixel. The results are given in Table 6.2. Compared to the situation of the original processor, this is a speedup of 1,57x on average.

Table 6.2: Cycles required with the new accelerator.

	Output Pixels	Accelerator Cycles	Cycle / Pix	Speedup
5x5	8x6=48	24	0,50	1,56
7x7	6x4=24	25	1,04	1,47
11x11	6x2=12	27	2,25	1,68

6.3.2 Processor Utilization

Calculating the 2D Convolution without acceleration requires four of the six available Issue Slots to be used continuously; the structure is shown in Figure 6.5. The first Issue Slot is used to transfer data from the external DDR memory to the local memory, and it is also used to store the obtained results back to the DDR memory. This transfer is done using Direct Memory Access (DMA). The second Issue Slot is used to multiply the data vector with the kernel vector. Note that since the vectors have 16-bit elements, the result after multiplication is 32-bit. This is split into two separate vectors, both containing 16-bit of the result (lower 16-bit and higher 16-bit). These are added to the results from the previous iteration, in Issue Slot 3 and 4. This addition can potentially result in a carry from the lower 16-bit, this carry is also given to the addition for the higher 16-bit. Finally, the result is given back to the first Issue Slot to store it in local memory and eventually store the final result in the external DDR memory. In summary: four of the six Issue Slots are used continuously, independent of the window size since that only influences the number of iterations this process is executed. For every Window Size, we obtain a $4/6 = 66,67\%$ Processor Utilization.

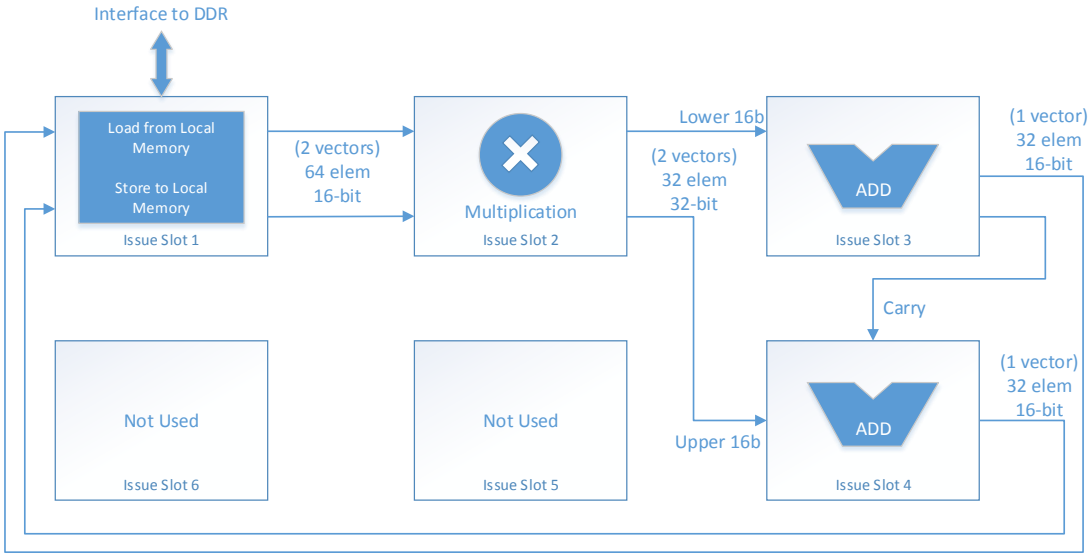


Figure 6.5: Processor Utilization without acceleration.

To calculate the Processor Utilization when using the accelerator, it is again necessary to look at the distribution of the Issue Slots. As shown in Figure 6.6, only two Issue Slots are required this time. One Issue Slot is used to load and store the data, and the other Issue Slot is used for the actual accelerator. However, these Issue Slots are not used continuously. For example, the 5x5 and 7x7 both require four vectors from the external memory to construct the 10x12 plane. The kernel does not change when processing the entire image; it can be loaded only at the beginning and stored in the local memory for the following iterations. Since one vector can be loaded from the local memory each cycle, the first Issue Slot is used only for four cycles. The 11x11 requires a 12x16 plane

with six vectors and thus the first Issue Slot is used for six cycles when processing the 11x11 window. Note that writing the results back can be done simultaneously to reading the next data. With proper pipelining, this does not influence the processor utilization. The number of vectors to write (1 or 2 vectors) is less than the number of vectors to read (4 or 6 vectors).

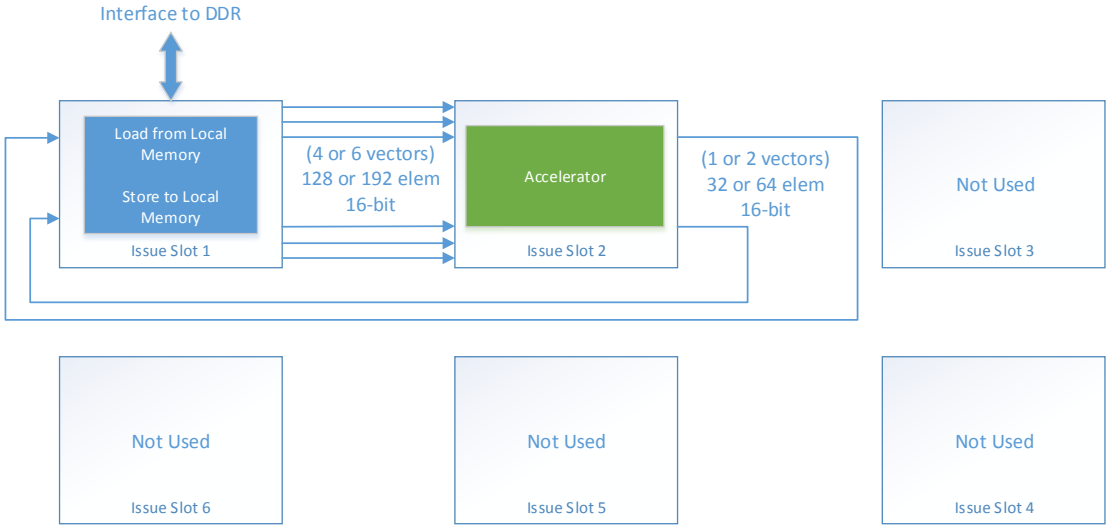


Figure 6.6: Processor Utilization with acceleration.

The second Issue Slot is used for the accelerator, but it can also process other tasks while waiting on the accelerator to finish and output its results. The accelerator needs to read the input data and kernel data from the local register file; this can be done with two vectors per cycle. The 10x12 input requires four vectors, and the 12x16 requires six vectors. The 5x5 kernel requires one vector, the 7x7 kernel requires two vectors, and the 11x11 kernel can be stored in 4 vectors. Similar to reading/writing to the external memory, the reading and writing of the local memory can be overlapped with proper pipelining. Since reading requires more data than writing, it is sufficient only to consider the reading. The second Issue Slot is used for $\lceil (4+1)/2 \rceil = 3$ cycles when using the 5x5. Similarly, it is used for three cycles with the 7x7 and five cycles with the 11x11. The total Processor Utilization can now be calculated with:

$$\text{Processor Utilization} = \frac{\text{Cycles IS1} + \text{Cycles IS2}}{6 * \text{Total Cycles}} * 100\% \tag{6.2}$$

The factor 6 is because of the 6 Issue Slots that could be used in total. The required number of cycles are given in Table 6.2. For example, for the 5x5 the Processor Utilization is calculated with:

$$\text{Processor Utilization}_{5x5} = \frac{4+3}{6*24} * 100\% = 4,86\% \tag{6.3}$$

The processor utilization is also calculated for the 7x7 and 11x11; the results are given in Table 6.3. On average, the processor utilization is 5,44%; that is an improvement of 12,61 times compared to the situation with the original processor. The results

for Speedup and Processor Utilization with the accelerator are shown in Figure 6.7. Additionally the Pixels per Cycle improvement is given in Figure 6.8. It can be observed that the speedup is similar for every window size, but the improvement in processor utilization shows a larger difference. This can be explained by the amount of data that needs to be transferred. The 5x5 and 7x7 are similar because they both use a 10x12 input that requires four vectors. However, the 11x11 requires a 12x16 input that is constructed with six vectors. These vectors need to be loaded from the external memory to the local register file and from the local memory to the accelerator; that requires more cycles.

Table 6.3: Processor Utilization on the original processor and with accelerator.

	Processor Utilization without accelerator	Processor Utilization with accelerator	Improvement
5x5	66,67%	4,86%	13,71
7x7	66,67%	4,67%	14,29
11x11	66,67%	6,79%	9,82

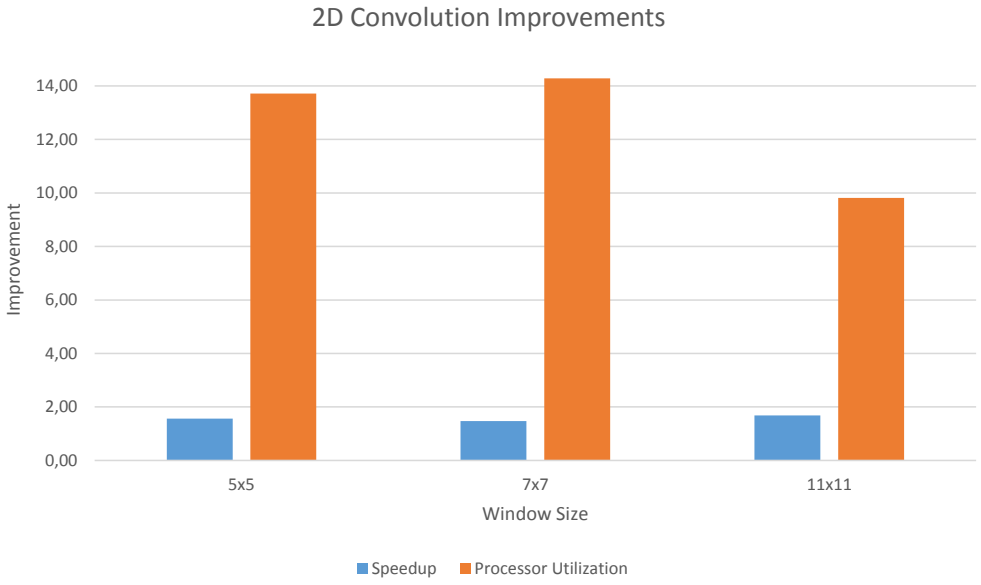


Figure 6.7: Speedup and Processor Utilization improvements for the 2D Convolution.

6.3.3 Throughput

The throughput is calculated by examining a video processing situation. The throughput is based on the number of cycles that are required to obtain one pixel. The results are shown in Figure 6.9. On a resolution of 1080p (1920x1080 pixels) the accelerator can process at least 100 Frames per Second on every window size. This allows the user to

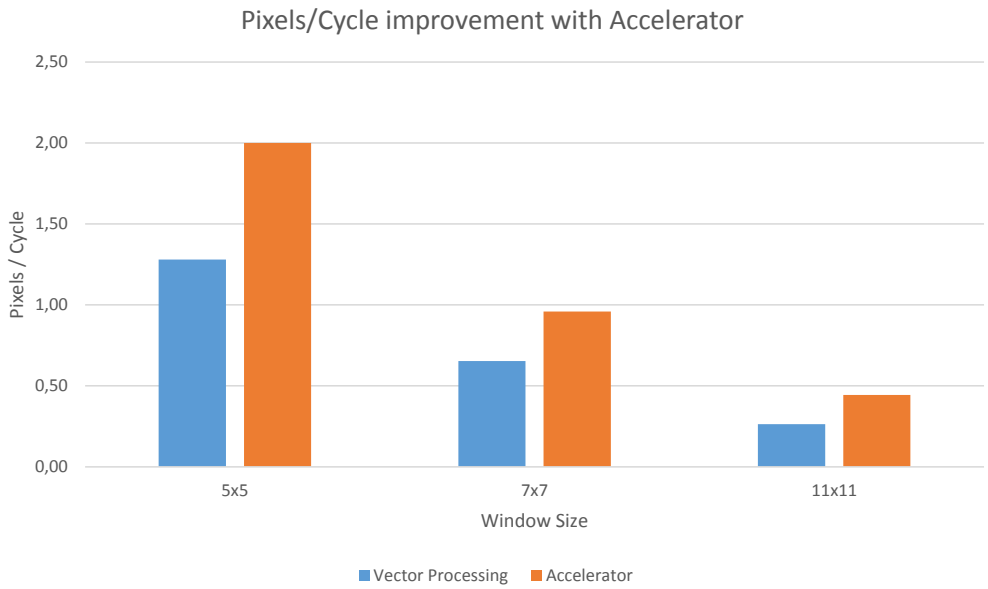


Figure 6.8: Pixels per cycle with and without accelerator.

create a slow motion video (used for action footage). A resolution of 4K (4096x2160 pixels) is also possible with all window sizes, although the 11x11 is on its limit with just 25 Frames per Second. Shooting a video at an 8K resolution (7680x4320 pixels) is only possible with a 5x5 window since the 7x7 results in 14 FPS and the 11x11 in 7 FPS. Currently, 8K video is hardly used, but it is interesting to include for future generations.

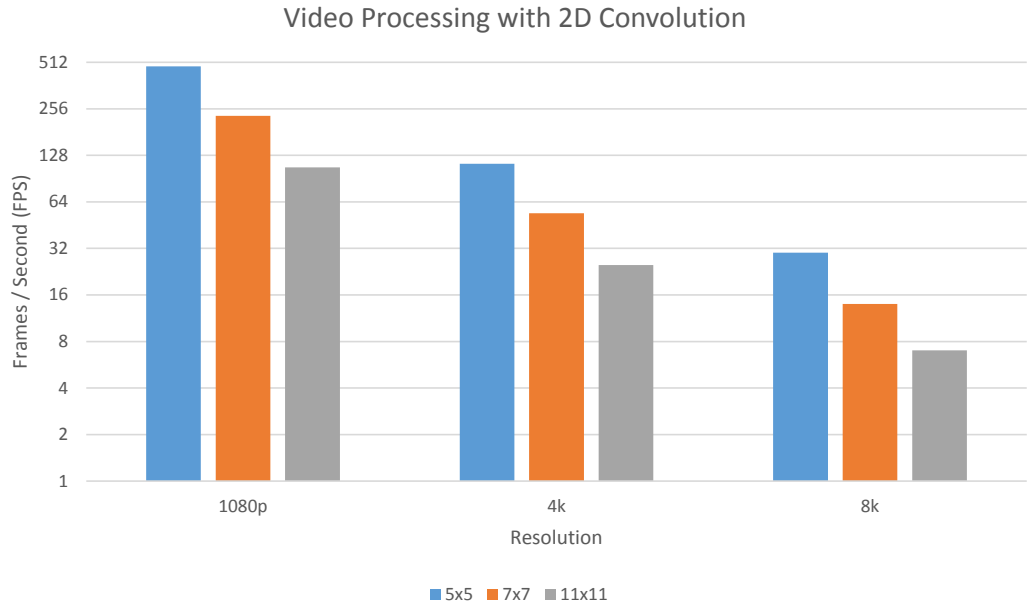


Figure 6.9: Video Processing with 2D Convolution accelerator.

6.3.4 Results on application level

In Chapter 3 we obtained the theoretical speedup for the different applications using Amdahl's law. The applications in the Intel Image Processing Unit that use the 2D Convolution are Convolutional Neural Networks (CNN) and Gaussian Blur. For the CNN, the 2D Convolution is used for 86,25% of the total number of operations. This resulted in a maximum theoretical speedup of 7,27x. The actual speedup for the CNN is obtained with:

$$\text{Speedup} = \frac{1}{(1 - 0,8625) + \frac{0,8625}{1,57}} = 1,46 \tag{6.4}$$

In this equation, the 0,8625 is the parallel fraction (2D Convolution), and the 1,57 is the obtained speedup for the 2D Convolution. Although significantly lower than the theoretical speedup, it still manages to accelerate the total application with 46%. The second application is Gaussian Blur. This application depends entirely on the 2D Convolution. In Chapter 3, we obtained a theoretical maximum speedup of 30,3x on an 11x11 convolution. The 7x7 convolution has a maximum speedup of 14,0x, and the 5x5 convolution has a maximum speedup of 8,3x. The actual speedup is 1,68x for the 11x11 convolution. The 7x7 convolution obtains a speedup of 1,47x, and the 5x5 Convolution achieves a speedup of 1,56x.

6.4 Census Transform

This section describes the results for the 5x5 Census Transform with the obtained speedup, processor utilization, and throughput. Finally, we compare the obtained speedup with the theoretical speedup for the different applications.

6.4.1 Speedup

The results for the 5x5 Census Transform are given in Table 6.4. Again, the cycles per pixel and the processor utilization have been calculated. The original version processes 32 values after 12 cycles. This is because the Sparse Census Transform requires 12 values to be compared, and each cycle one comparison is done. The accelerator outputs the same number of outputs as with the 5x5 2D Convolution; which is 48 outputs (8x6). The accelerator also takes 12 cycles before it is finished, but has more results available. In total, this results in a 1,50x speedup.

Table 6.4: Improvements for the Census Transform.

	Output Pixels	Cycles Required	Cycle / Pix	Speedup	Processor Utilization	Proc. Util. Improvement
5x5 Original	32	12	0,38	-	33,33%	-
5x5 Accelerator	48	12	0,25	1,50	8,33%	4,00

6.4.2 Processor Utilization

The original processor would require two Issue Slots to compute the Census Transform. The first Issue Slot is used for reading and writing data, the second one to do the actual comparisons. Using the accelerator also requires two Issue Slots, similarly to the situation with 2D Convolution described before and shown in Figure 6.6. However, these are not used continuously; the processor can do other tasks while waiting for the results. The first Issue Slot is used for four cycles to load the four vectors (and store the results from the previous calculation at the same time). The second Issue Slot is used for two cycles to load these four vectors from the local memory. In total, this results in a processor utilization of 8,33%, which is an improvement of 4,00 compared to the original situation.

$$\text{Processor Utilization Census Transform}_{5x5} = \frac{4+2}{6*12} * 100\% = 8,33\% \tag{6.5}$$

6.4.3 Throughput

Similarly to the calculations performed in Section 6.3.3, the real-time processing capability for the Census Transform is explored. The results are shown in Figure 6.10. Clearly both the original and new accelerator manage to provide real-time processing capability. At an 8K video resolution (7680x4320 pixels), the original system manages to process 40 frames per second. The accelerator manages to increase this to 60 frames per second. Unfortunately, the larger window sizes (7x7 and 11x11) are currently not implemented. With larger windows, a lower frames per second is expected because more comparisons are required.

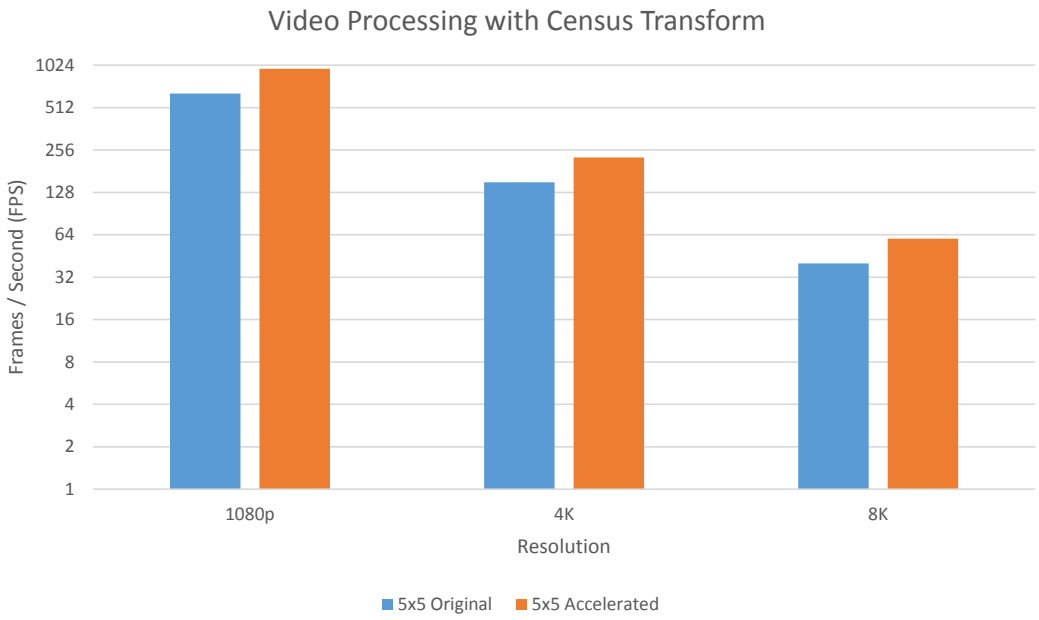


Figure 6.10: Video Processing with the Census Transform accelerator.

6.4.4 Results on application level

Again, we compare the obtained speedup against the theoretical speedup obtained in Chapter 3. The application in the Intel Image Processing Unit that uses the Census Transform is Stereo Vision. In this application, both the Census Transform and 2D Convolution are used. The Census Transform is used 5% of the total operations, and the 2D Convolution is used for 51% of the total operations. This results in a theoretical speedup of 1,05x when only accelerating the Census Transform, and 2,10x when accelerating only the 2D Convolution. Accelerating both, we obtained a theoretical speedup of 2,33x. In this chapter, we obtained the speedup for the Census Transform itself: 1,50x. The speedup for the 2D Convolution is 1,57x. This results in an actual speedup on application level of 1,02x when accelerating only the Census Transform and 1,23x when accelerating only the 2D Convolution. When both are accelerated, the actual speedup is 1,25x.

Finally, it is worth mentioning that the modified Local Binary Patterns (mLBP) is also explored in this thesis since it is very similar to the Census Transform. In Chapter 3, we obtained a theoretical speedup of 6,04x when accelerating this filter for the Face Detection application. The design of the mLBP accelerator is proposed in Chapter 5, featuring a combination of the 2D Convolution and Census Transform. Since the actual speedup for the Census Transform is 1,50x and the actual speedup for the 2D Convolution is 1,57x, we can estimate the speedup for the mLBP to be in the same order of magnitude (1,535x). This results in a total application speedup of 1,41x.

6.5 Area

As explained in Chapter 5, the design of the new accelerator is an extension on the original Block Matching and Bilateral Filter accelerator. This way, the area overhead is limited by reusing several components like the multipliers. The area for the new accelerator is compared to the original accelerator. This is done by synthesizing both designs using the same technology (14 nm). In total the area is increased from 403.511 gates to 573.938 gates (these gates are NAND2 equivalents); this means an increase of 42%. Figure 6.11 shows an analysis of the different sub-components (entities) used in both accelerators. The area in this figure is also presented as the number of gates (NAND2 equivalents). The second adder tree is clearly increased in the area. This increase was expected because the original adder tree works on 16-bit elements and is extended to support 24-bit numbers. The Rolling Plane and LUT are not modified, but still show a small difference in area. This can be explained by some variations in the synthesis process. The Shape Selection increased the most: a 94% increase compared to the original accelerator. This is caused because an 11x11 window is selected (121 elements) instead of selecting 61 elements. Finally, the area for the output registers is increased because the storage for more intermediate values is required. Unfortunately, the area for the extra multiplexers is not available since they do not belong to a separate entity, but the multiplexers are included in the overall increase of 42%.

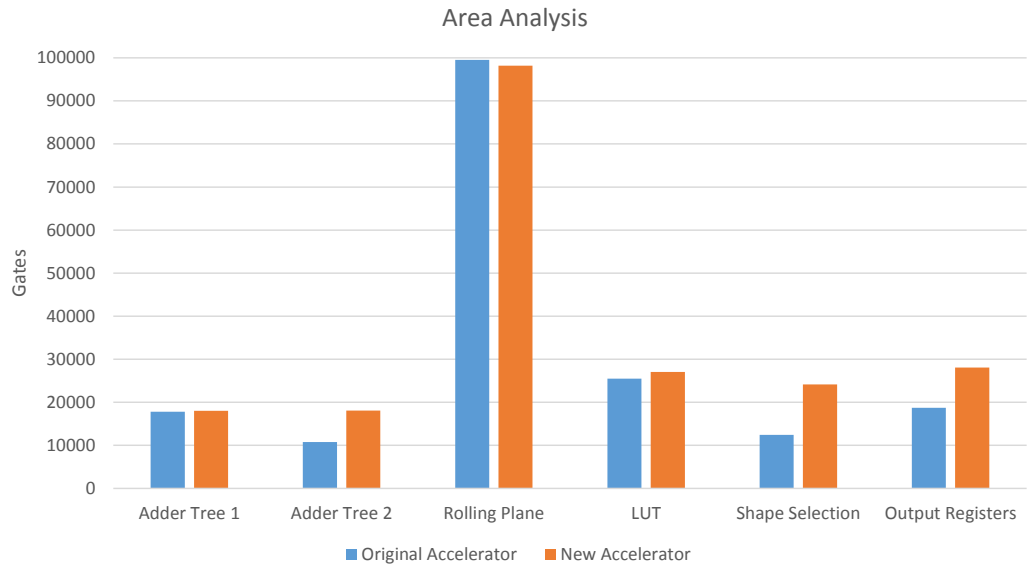


Figure 6.11: Area of the original and new accelerator.

Besides comparing the accelerators individually, the area is also compared at the system level. This is achieved by integrating the new accelerator in the Image Processing Unit where other Issue Slots and Functional Units are present. The area increases from 2.656.432 gates to 2.826.859 gates, that means an increase of only 6,42%.

6.6 Multiple instances of the accelerator

Although the earlier presented speedup is not 'massive', the accelerator manages to reduce the processor load quite significantly. As mentioned before, this reduction has a relation with the dynamic power consumption or it allows the compiler to use the processor for other tasks. A way to increase the speedup is to add the accelerator to multiple Issue Slots. This will influence speedup, the processor load, and area usage. This trade-off, for the 2D Convolution, is shown in Table 6.5 and Figure 6.12. This data is based on a comparison with the original processor as described before (shown in Figure 6.5), and it is based on the average of the 5x5, 7x7, and 11x11 results. It can be observed that the maximum speedup is 7,86x when using all the currently available Issue Slots (5 Issue Slots with the accelerator and 1 Issue Slot for loading data). An interesting result is that the processor utilization is still better than the original processor solution; around 2,5 times better. This can be explained by the fact that after loading data to the accelerator, the Issue Slot is available for other tasks until the accelerator is done. In the original processor, data has to be loaded every cycle, fully occupying the Issue Slots. Looking at the results, it becomes clear that the speedup increases faster than the additional area increase. So it is definitely possible to add more accelerators, especially in future generations when the transistor density increases.

Table 6.5: Using multiple 2D Convolution accelerators.

Issue Slots	Speedup	Processor Utilization Improvement	Area Increase
0	1,00	1,00	1,00
1	1,57	12,61	1,06
2	3,14	6,30	1,28
3	4,71	4,20	1,50
4	6,28	3,15	1,71
5	7,86	2,52	1,93
6	8,59	2,28	2,14
7	8,71	2,25	2,36
8	8,71	2,25	2,58
9	8,71	2,25	2,79

Increasing the number of Issue Slots above 5 shows an interesting property: the speedup and processor utilization are no longer influenced. The reason for this is that data transfers are becoming the dominant factor. Providing all Issue Slots with new data requires more cycles than the actual computation. The only way to deal with this problem is to increase the amount of data that can be transferred between the external DDR memory and the processor.

Using the data from Table 6.5, the optimal number of Issue Slots can be determined. The largest increase in speedup over the area increase is obtained when using two Issue Slots with the new accelerator. At this point, the speedup is doubled (1,57x to 3,14x) while the area only increases with 20% (1,06x to 1,28x). Another interesting metric is the combination of all three metrics: $(Speedup * Processor\ Utilization)/Area$. Using one Issue Slot provides the best results in this combined metric, this can be explained because the Processor Utilization is improved significantly at this point.

The same trade-off is performed for the Census Transform, and the results are shown in Figure 6.13. Clearly the speedup and processor utilization are saturated earlier; this already happens after filling 3 Issue Slots with the accelerator. This can be explained because computing the Census Transform requires 12 cycles (the 2D Convolution requires up to 27 cycles). The required amount of cycles to load data from the external memory when filling more than 3 Issue Slots is higher than 12 cycles. From the results, it becomes clear that the maximum speedup is 4,5x for the Census Transform.

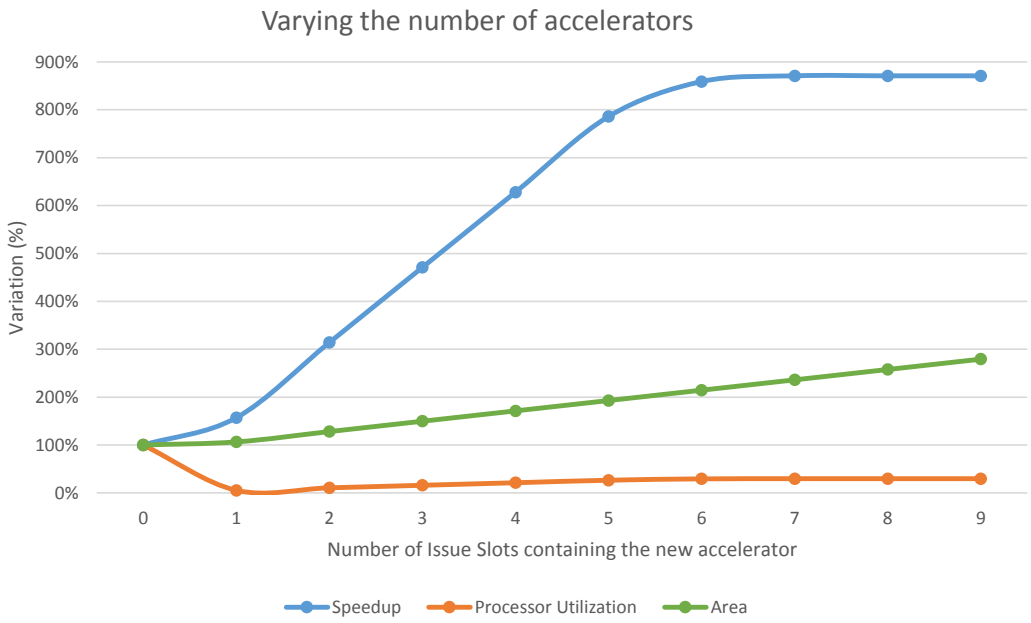


Figure 6.12: Trade-off between speedup, processor utilization, and area. Data is based on the 2D Convolution results.

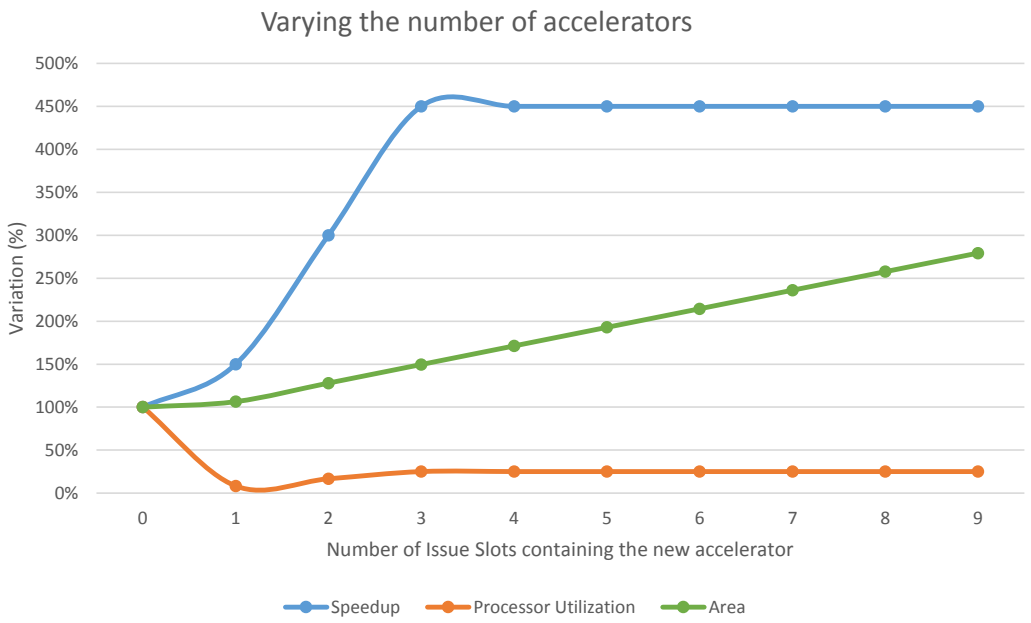


Figure 6.13: Trade-off between speedup, processor utilization, and area. Data is based on the Census Transform results.

6.6.1 Influence on the applications

In Section 6.3.4 and Section 6.4.4, we already obtained results on the application level. However, these are based on the assumption of a single accelerator. In this section, multiple accelerators are introduced. This increases the speedup to 7,86x for the 2D Convolution and 4,50x for the 5x5 Census Transform, when using all (currently) available Issue Slots. The first application is Convolutional Neural Networks; this application had a speedup of 1,46x on a single accelerator. When using multiple accelerators, a speedup of 4,04x is possible for the entire application. The second application is Gaussian Blur. Since this application depends entirely on the 2D Convolution, we obtain a speedup of 1,57x with a single accelerator and 7,86x with multiple accelerators. The next application is Stereo Vision. When accelerating both the Census Transform and 2D Convolution, a speedup of 1,25x is obtained with a single accelerator. Using multiple accelerators, the speedup increases to 1,94x. Finally, the Face Detection application is considered. The Face Detection application uses the modified Local Binary Patterns (mLBP), the mLBP can be implemented by combining the 2D Convolution and Census Transform. To obtain the speedup, we can estimate the speedup of the mLBP to be around the average of the speedup results obtained in this chapter. This results in an application speedup of 1,41x with a single accelerator and 3,33x with multiple accelerators. The previously mentioned results are summarized in Table 6.6. For most applications, the theoretical speedup is still around 2x higher than the achieved speedup with multiple accelerators.

Table 6.6: Speedup for the different applications.

Application	Theoretical	1 Accelerator	5 Accelerators
Convolutional Neural Network	7,27	1,46	4,04
Gaussian Blur	17,53	1,57	7,86
Stereo Vision	2,33	1,25	1,94
Face Detection	6,04	1,41	3,33

6.7 Conclusions

In this chapter, the results of this thesis are presented. The chapter starts by explaining the different metrics and how these can be obtained. These metrics are speedup, processor utilization, throughput, and area. Next, the simulation results and testing of the design is presented. It is shown that the rolling plane is correctly moving the data so each cycle a new window can be selected. Furthermore, the new signed multipliers implemented by combining two unsigned multipliers also provide the expected output.

The obtained speedup, processor utilization, and throughput are presented separately for the 2D Convolution and Census Transform. The speedup is based on the number of cycles that are required to obtain a single output pixel. For the 2D Convolution, the original processor is capable of producing 32 outputs after a certain amount of cycles: 25, 49, and 121 cycles for the 5x5, 7x7, and 11x11 window respectively. This assumes a fully-pipelined approach where new data is continuously loaded. The new accelerator

provides 48 pixels in 24 cycles for the 5x5 convolution. The 7x7 convolution provides 24 outputs after 25 cycles, and the 11x11 convolution provides 12 outputs after 27 cycles. Compared to the original situation, this results in an average speedup of 1,57x. The Processor Utilization determines how much of the processor is used. In the original processor, four Issue Slots are required to perform the 2D Convolution: one for loading the data, one for the multiplication, and two for the addition. With the new accelerator, only two Issue Slots are occupied. However, these are not continuously occupied; the processor can do other tasks or save power while waiting for the accelerator to finish. On average the accelerator reduces the processor load with a factor 12,61. The accelerator is also tested at real-time processing to obtain the throughput and to determine if it can handle large videos. The results show that the 2D Convolution accelerator can handle 4K video on all window sizes, but 8K video is only possible with the 5x5 convolution. The obtained speedup is also examined on the application level. The 2D Convolution has two applications: Convolutional Neural Networks (CNN) and Gaussian Blur. The theoretical speedup for the CNN is 7,27x (determined in Chapter 3), and the actual speedup is 1,46x. For the Gaussian Blur, the theoretical speedup is 17,5x on average, and the actual speedup is 1,57x since it depends entirely on the 2D Convolution.

The same metrics are determined for the 5x5 Census Transform. The original processor requires 12 cycles to give 32 results, and the accelerator provides 48 results in 12 cycles. This results in a speedup of 1,5x. The Processor Utilization is improved by 4,0x since the processor can do other tasks while waiting for the accelerator to finish. Again, real-time processing is examined. It becomes clear that the 5x5 Census Transform is capable of processing video in both 4K and 8K video, but it can be expected that the larger window sizes will show similar behaviour on real-time processing as the 2D Convolution. Again, the obtained speedup is compared on the application level. Only one application uses the Census Transform: Stereo Vision. This application can also use the 2D Convolution for a different part of the algorithm. The theoretical speedup when accelerating both the Census Transform and 2D Convolution is 2,33x, but with the actual accelerator the speedup is 1,25x.

Next, the area overhead is determined by synthesizing the design twice: once for the original accelerator and once for the extended accelerator. The results show an increase of 42% in the area for the accelerators. At the system level, the total area increase is only 6%. When looking at the sub-components, the Shape Selection has increased the most. This is because the original accelerator selects 61 elements while the new accelerator selects 121 elements. The second adder tree has increased as well, but this was expected because it was extended from 16-bit to 24-bit. The area for the output registers is increased because the new results have to be stored temporarily.

Finally, it is also possible to use multiple accelerators in the processor. This will improve the speedup for the 2D Convolution up to 7,86x when using all available Issue Slots. This increase in speedup comes at the cost of a higher processor utilization and an increase in area. The 5x5 Census Transform has a maximum speedup of 4,50x. On the application level, this results in a speedup of 4,04 for the Convolutional Neural Network. The Gaussian Blur obtains a speedup of 7,86 and the Stereo Vision application achieves a speedup of 1,94. Finally, the speedup for the Face Detection application can be estimated, this results in a speedup of 3,33. An interesting result is that the speedup

and processor utilization saturate after using the accelerator in a certain amount of Issue Slots. The reason is because of the data transfers from the external memory; more cycles are required for this than the actual computation. Thus, it can be concluded that adding more accelerators provides a trade-off between speedup, processor utilization (influencing dynamic power consumption) and area.

7 Conclusions and Future Work

This chapter describes the conclusions and provides suggestions for possible future work.

7.1 Conclusions

In this thesis, the acceleration of filters used for Image Processing has been explored. Specifically the filters known as 2D Convolution, Census Transform, and Local Binary Patterns are examined. The state of the art analysis in Chapter 2 examines these filters and their corresponding applications. The analysis shows that there are many different variants of each filter, and there is not a single variant that can be considered the best. The analysis also includes various acceleration techniques; comparing implementations on GPUs, FPGAs, and ASICs. For most situations, custom logic (ASIC) can obtain the best performance. Another observation is that most accelerators are aimed to accelerate a single application. In this work, a flexible accelerator that can be used for different applications is presented. The specific applications that can be accelerated by the mentioned filters and are used in the Intel Image Processing Unit are explored in Chapter 3. These applications are Convolutional Neural Networks (CNN), Gaussian Blur, Stereo Vision, and Face Detection. It becomes clear that a good speedup can be achieved when the different filters are accelerated. For the Convolutional Neural Network, the theoretical speedup is 7,27x. The Gaussian Blur depends entirely on the 2D Convolution and has a theoretical speedup of 17,53x. The Stereo Vision application has a theoretical speedup of 2,33x when accelerating both the Census Transform and 2D Convolution. Finally, the Face Detection application could potentially achieve a speedup of 6,04x.

We propose two options for designing the new accelerator: creating an entirely new accelerator or extending the functionality of an existing accelerator. We chose the second option since it reduces the extra area that is required. The structure of the current accelerator and the design of the new (extended) accelerator is described in Chapter 4 and 5 respectively. For the 2D Convolution, support for multiple kernel sizes (5x5, 7x7 and 11x11) is implemented to obtain the highest possible speedup. This is achieved by performing multiple 5x5 convolutions in parallel and combining the results together to create larger windows. The rolling plane and shape selection from the original accelerator are reused but required some modifications to support the different sizes. Additionally, the multipliers are reused. This required extra logic to combine the small unsigned 8-bit multipliers to a signed 16-bit multiplier. Adding the results after multiplication together is done by reusing the adder tree, which required an extension and extra outputs for the intermediate values. Support for the 5x5 Census Transform is also implemented, and the design for the other sizes is presented. The Census Transform requires a comparison of pixels with the center pixel. This has been achieved by reusing the subtraction units and storing the sign bit. Finally, a design for the modified Local Binary Patterns (mLBP) is

presented. The mLBP requires the comparison of pixels with the average value instead of the center pixel. The calculation of the average value can be performed with a 2D Convolution. Effectively, the mLBP combines the acceleration of 2D Convolution and Census Transform together.

Finally in Chapter 6, we evaluate the possible improvements introduced by the implemented accelerator. Once the implemented components required for the new accelerator are tested, the new and the existing accelerators are compared in terms of speedup, processor utilization, throughput, and area. The simulation results show that the implemented components correctly perform the expected operations. Next, the new accelerator is compared to the original Intel Image Processing Unit without acceleration. The resulting speedup for the 2D Convolution is 1,57x on average for the different window sizes. A higher speedup can be obtained by using multiple instances of the new accelerator, which in the current system is limited to five of these instances. Using five instances results in a speedup of 7,86x. Regarding the processor utilization, we demonstrate that the processor is less occupied when using the accelerator; this allows the processor to perform other tasks in the background or reduce dynamic power consumption by clock-gating the unused resources. The processor utilization is improved with 12,61x when using a single accelerator. When using five instances the processor utilization is still improved by 2,52x because the processor is not continuously required. The throughput is obtained by analyzing the obtained performance with video processing. The results show that the 2D Convolution accelerator can handle a 4K resolution on all window sizes, but an 8K resolution is only possible with the 5x5 Convolution. For the 5x5 Census Transform, the speedup is 1,50x, and it shows a processor utilization improvement of 4,00x. The maximum speedup is 4,50x when using three accelerators; adding more accelerators does not influence the speedup and the processor utilization. The corresponding processor utilization is improved by 1,33x, and the 5x5 Census Transform is capable of processing both 4K and 8K video. Next, the obtained speedup is compared on application level. As described before, the theoretical speedup for the CNN is 7,27x. The actual speedup is 1,46x when using a single accelerator, increasing the number of accelerators results in a speedup of 4,04x. The Gaussian Blur application has a theoretical speedup of 17,53x. The actual speedup is 1,57x with a single accelerator and 7,86x with multiple accelerators. The Stereo Vision application has a theoretical speedup of 2,33x. The actual speedup is 1,25x when using a single accelerator and 1,94x using multiple accelerators. Finally, the Face Detection application has a theoretical speedup of 6,04x. The Face Detection application requires the modified Local Binary Patterns (mLBP), but this is not implemented. However, the mLBP can be constructed as a combination of 2D Convolution and Census Transform. Using the results for these filters, we can estimate that the actual speedup to be 1,41x using a single accelerator and 3,33x using multiple accelerators. The reason for the differences between the theoretical speedup and the actual speedup is that the theoretical speedup assumes an infinite acceleration of the parallel fraction. In practice, the speedup is lower because of memory limitations. The area for the new accelerator is obtained and compared to the area of the original accelerator. The area of the accelerator increases with 42%. At the system level, the total area increase is only 6%.

We can conclude that the new accelerator is very flexible; it can be used for 2D Convolution, Census Transform, and Local Binary Patterns while retaining the original functionality of the Bilateral Filter and Block Matching accelerator. The use of a flexible window size results in even more options for the user and potentially a greater speedup when multiple sizes are required for the application. Varying the number of accelerators (see Figure 6.12 and Figure 6.13) provides a trade-off between speedup, processor utilization (also influencing the dynamic power consumption), and area usage.

7.2 Future Work

The design described in this thesis can be expanded by further research and implementations. Proposed topics are:

- Research can be done on other filters that share similarities with the filters described in this thesis. Especially filters that require the sliding of a window across the image have a good potential to be accelerated. This will create an even more flexible solution, capable of performing many different tasks with a limited increase in area.
- Implementation of more acceleration options in CHDL, specifically the 7x7 and 11x11 Census Transform can be added to extend functionality. Additionally support for the modified Local Binary Patterns can be added by combining the 2D Convolution and Census Transform, as shown in Figure 5.8.
- Implementation of a smaller accelerator that only computes the 5x5 and 7x7 Convolution. Removing the 11x11 reduces the number of elements to select from the data plane from 121 to 49. The original accelerator also provides a smaller version, and parts of this can be reused. Additionally an even smaller version can be made by implementing the 2D Convolution as a separate functional unit, but this is only a viable solution if the original accelerator is no longer required. Otherwise, the extension of the original accelerator will always result in lower area overhead compared to adding a new functional unit, this is because adding extra multipliers requires more area than adding multiplexers to reuse existing multipliers.
- Obtain results for the change in dynamic power consumption. Power measurements are currently not possible due to organizational reasons. However, the processor utilization is decreased significantly; this allows the processor to reduce power by clock-gating most of its components.
- Testing the design on an FPGA. Similarly to the dynamic power measurements, this is currently not possible due to organizational reasons.

Bibliography

[1] Apple Inc. Kernel convolution. [Online]. Available: https://developer.apple.com/library/mac/documentation/Performance/Conceptual/vImage/ConvolutionOperations/ConvolutionOperations.html

[2] R. C. Gonzalez and R. E. Woods, *Digital image processing*. Prentice hall Upper Saddle River, NJ:, 2002.

[3] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE Transactions on Neural Networks*, vol. 8, no. 1, pp. 98–113, 1997.

[4] F. Mamalet, S. Roux, and C. Garcia, "Embedded facial image processing with convolutional neural networks," in *Proceedings of 2010 IEEE International Symposium on, Circuits and Systems (ISCAS)*, 2010, pp. 261–264.

[5] Theano Development Team. Convolutional neural networks (LeNet). [Online]. Available: http://deeplearning.net/tutorial/lenet.html

[6] E. S. Gedraite and M. Hadad, "Investigation on the effect of a Gaussian blur in image filtering and segmentation," in *2011 Proceedings ELMAR*, 2011, pp. 393–396.

[7] S. Khorbotly and F. Hassan, "A modified approximation of 2D Gaussian smoothing filters for fixed-point platforms," in *2011 IEEE 43rd Southeastern Symposium on System Theory (SSST)*, 2011, pp. 151–159.

[8] P.-Y. Hsiao, S.-S. Chou, and F.-C. Huang, "Generic 2-D Gaussian smoothing filter for noisy image processing," in *TENCON 2007-2007 IEEE Region 10 Conference*, 2007, pp. 1–4.

[9] R. Zabih and J. Woodfill, "Non-parametric local transforms for computing visual correspondence," in *Computer VisionECCV'94*. Springer, 1994, pp. 151–158.

[10] Y. K. Baik, J. H. Jo, and K. M. Lee, "Fast census transform-based stereo algorithm using SSE2," in *The 12th Korea Japan Joint Workshop on Frontiers of Computer Vision, Seoul National University Computer Vision Lab., Seoul, Feb.–2006*, 2006, pp. 305–309.

[11] B. Froba and A. Ernst, "Face detection with the modified census transform," in *2004. Proceedings. Sixth IEEE International Conference on Automatic Face and Gesture Recognition*, 2004, pp. 91–96.

[12] E. Irijanti, M. Nayan, and M. Yusoff, "Local stereo matching algorithm: Using small-color census and sparse adaptive support weight," in *National Postgraduate Conference (NPC), 2011*, 2011, pp. 1–5.

[13] L. Ma, J. Li, J. Ma, and H. Zhang, "A modified census transform based on the neighborhood information for stereo matching algorithm," in *2013 Seventh International Conference on Image and Graphics (ICIG)*, 2013, pp. 533–538.

[14] X. Luan, H. Zhou, F. Yu, X. Li, B. Xue, and D. Song, "A robust local census-based stereo matching insensitive to illumination changes," in *2012 International Conference on Information and Automation (ICIA)*, 2012, pp. 801–805.

[15] D. Scharstein, H. Hirschmüller, Y. Kitajima, G. Krathwohl, N. Nešić, X. Wang, and P. Westling, "High-resolution stereo datasets with subpixel-accurate ground truth," in *Pattern Recognition*. Springer, 2014, pp. 31–42.

[16] F. Pelissier and F. Berry, "PhD forum: BiSeeMos: A fast embedded stereo smart camera," in *2011 Fifth ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC)*, 2011, pp. 1–3.

[17] J. Zheng, G. A. Ramírez, and O. Fuentes, "Face detection in low-resolution color images," in *Image Analysis and Recognition*. Springer, 2010, pp. 454–463.

[18] M. Humenberger, T. Engelke, and W. Kubinger, "A census-based stereo vision algorithm using modified semi-global matching and plane fitting to improve matching quality," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2010, pp. 77–84.

[19] C. Pardo Beainy, F. Jimenez Lopez, E. Gutierrez Caceres, and L. Fredy Sosa Quintero, "Disparity map generation, from the use of rectified images," in *2013 XVIII Symposium of Image, Signal Processing, and Artificial Vision (STSIVA)*, Sept 2013, pp. 1–6.

[20] D. Huang, C. Shan, M. Ardabilian, Y. Wang, and L. Chen, "Local binary patterns and its application to facial image analysis: a survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 41, no. 6, pp. 765–781, 2011.

[21] H. Jin, Q. Liu, H. Lu, and X. Tong, "Face detection using improved LBP under bayesian framework," in *2004 IEEE First Symposium on Multi-Agent Security and Survivability*, 2004, pp. 306–309.

[22] K. Meena and A. Suruliandi, "Local binary patterns and its variants for face recognition," in *2011 International Conference on Recent Trends in Information Technology (ICRTIT)*, 2011, pp. 782–786.

[23] A. Sindhuja, S. D. Mahalakshmi, and K. Vijayalakshmi, "Age invariant face recognition with occlusion," in *2012 IEEE International Conference on Advanced Communication Control and Computing Technologies (ICACCCT)*, 2012, pp. 83–87.

[24] J. Wang, D. Lee, and K. N. Plataniotis, "Face detection in mobile phones using co-occurrence of adjacent local binary patterns," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013, pp. 2410–2414.

[25] S. Che, J. Li, J. W. Sheaffer, K. Skadron, and J. Lach, "Accelerating compute-intensive applications with GPUs and FPGAs," in *2008. SASP 2008. Symposium on Application Specific Processors*, 2008, pp. 101–107.

[26] E. S. Chung, P. A. Milder, J. C. Hoe, and K. Mai, "Single-chip heterogeneous computing: Does the future include custom logic, FPGAs, and GPGPUs?" in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, 2010, pp. 225–236.

[27] S. Asano, T. Maruyama, and Y. Yamaguchi, "Performance comparison of FPGA, GPU and CPU in image processing," in *2009. FPL 2009. International Conference on Field Programmable Logic and Applications*, 2009, pp. 126–131.

[28] L. M. Russo, E. C. Pedrino, E. Kato, and V. O. Roda, "Image convolution processing: A GPU versus FPGA comparison," in *2012 VIII Southern Conference on Programmable Logic (SPL)*, 2012, pp. 1–6.

[29] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun, "CNP: An FPGA-based processor for convolutional networks," in *2009. FPL 2009. International Conference on Field Programmable Logic and Applications*, 2009, pp. 32–37.

[30] S. Potluri, A. Fasih, L. K. Vutukuru, F. Al Machot, and K. Kyamakya, "CNN based high performance computing for real time image processing on GPU," in *2011 Joint 3rd Int'l Workshop on Nonlinear Dynamics and Synchronization (INDS) & 16th Int'l Symposium on Theoretical Electrical Engineering (ISTET)*, 2011, pp. 1–7.

[31] J. Sabarad, S. Kestur, M. S. Park, D. Dantara, V. Narayanan, Y. Chen, and D. Khosla, "A reconfigurable accelerator for neuromorphic object recognition," in *2012 17th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2012, pp. 813–818.

[32] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto, and H. P. Graf, "A massively parallel coprocessor for convolutional neural networks," in *Application-specific Systems, Architectures and Processors, 2009. ASAP 2009. 20th IEEE International Conference on*, 2009, pp. 53–60.

[33] N. Farrugia, F. Mamalet, S. Roux, F. Yang, and M. Paindavoine, "A parallel face detection system implemented on FPGA," in *2007. ISCAS 2007. IEEE International Symposium on Circuits and Systems*, 2007, pp. 3704–3707.

[34] C. D. Moreno, F. J. Quiles, M. Ortiz, M. Brox, J. Hormigo, J. Villalba, and E. L. Zapata, "Efficient mapping on FPGA of convolution computation based on combined CSA-CPA accumulator," in *2009. ICECS 2009. 16th IEEE International Conference on Electronics, Circuits, and Systems*, 2009, pp. 419–422.

[35] M. Peemen, A. A. Setio, B. Mesman, and H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," in *2013 IEEE 31st International Conference on Computer Design (ICCD)*, 2013, pp. 13–19.

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

[37] T. Jain, P. Bansod, C. S. Kushwah, and M. Mewara, "Reconfigurable hardware for median filtering for image processing applications," in *2010 3rd International Conference on Emerging Trends in Engineering and Technology (ICETET)*, 2010, pp. 172–175.

[38] C. Murphy, D. Lindquist, A. M. Rynning, T. Cecil, S. Leavitt, and M. L. Chang, "Low-cost stereo vision on an FPGA," in *2007. FCCM 2007. 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2007, pp. 333–334.

[39] D. Han, J. Choi, J.-I. Cho, and D. Kwak, "Design and VLSI implementation of high-performance face-detection engine for mobile applications," in *2011 IEEE International Conference on Consumer Electronics (ICCE)*, 2011, pp. 705–706.

[40] W. Wang, J. Yan, N. Xu, Y. Wang, and F.-H. Hsu, "Real-time high-quality stereo vision system in FPGA," in *2013 International Conference On Field-Programmable Technology (FPT)*, 2013, pp. 358–361.

[41] J. Boutellier, I. Lundbom, J. Janhunen, J. Ylimäinen, and J. Hannuksela, "Application-specific instruction processor for extracting local binary patterns," in *2012 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, 2012, pp. 1–8.

[42] T. Kryjak, M. Komorkiewicz, and M. Gorgon, "FPGA implementation of real-time head-shoulder detection using local binary patterns, SVM and foreground object detection," in *2012 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, 2012, pp. 1–8.

[43] S. K. Rethinagiri, O. Palomar, J. A. Moreno, O. Unsal, and A. Cristal, "An energy efficient hybrid FPGA-GPU based embedded platform to accelerate face recognition application," in *2015 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS XVIII)*, 2015, pp. 1–3.

[44] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *Artificial Neural Networks–ICANN 2010*, 2010, pp. 92–101.

[45] T. Kim and R. Wooju, "Convolutional Neural Network (CNN)," in *Intel Internal document*, 2014.

[46] T. Ojala, M. Pietikäinen, and T. Mäenpää, "Gray scale and rotation invariant texture classification with local binary patterns," in *Computer Vision-ECCV 2000*. Springer, 2000, pp. 404–420.

[47] D. Maturana, D. Mery, and A. Soto, "Face recognition with local binary patterns, spatial pyramid histograms and naive Bayes nearest neighbor classification," in *2009 International Conference of the Chilean Computer Science Society (SCCC)*, 2009, pp. 125–132.

[48] G. Voronov, "IPU4 imaging pipeline overview," in *Intel Internal document*, 2014.

[49] Wikipedia. Bilateral filter. [Online]. Available: https://en.wikipedia.org/wiki/Bilateral_filter

[50] F. Bouwens, "Block matching and bilateral filter accelerators: Micro-architecture-specification," in *Intel Internal document*, 2014.

[51] B. Parhami, *Computer arithmetic: algorithms and hardware designs.* Oxford University Press, Inc., 2009.

[52] M. Ashwath and B. Premananda, "Signed fixed-point multiplier for DSP using vertically and crosswise algorithm," in *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*, 2013, pp. 1–6.