Department of Physics and Astronomy

University of Heidelberg

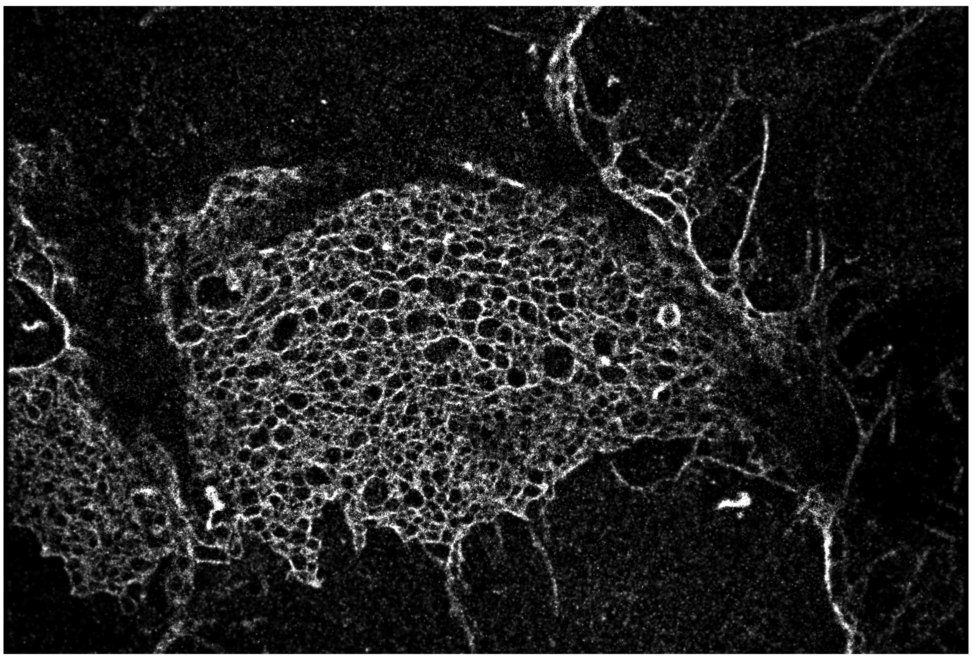
Diploma Thesis
in Physics

submitted by
Manfred Kirchgessner
born in
Bad Soden, Germany

Oct 2011

FPGA-Based Hardware Acceleration of Localization Microscopy

Development of a hardware compatible algorithm for the processing of localization microscopy data and implementation of this algorithm on an FPGA card.



This diploma thesis has been carried out by **Manfred Kirchgessner**

at the

Kirchhoff Institute of Physics

under the supervision of

Prof. Dr. Udo Kebschull

Abstract

FPGA-Based Hardware Acceleration of Localization Microscopy - Development of a hardware-compatible algorithm for processing localization-microscopy-data and implementation of this on an FPGA card

This diploma thesis describes the development and the implementation of an algorithm for acceleration of data processing from localization microscopy. The aim is to realize data processing in real time, which means to calculate the localization data within the same time that is required for acquisition of the images. In order to achieve high computation speed FPGAs are applied.

The idea of localization microscopy exists since the 1990s. This new technique provides the possibility to analyze images from light microscopy of biological samples with improved optical resolution. Although this technique promises great new findings in biology or medicine, it is rarely used. One of the reasons for this certainly is the high processing duration until the final high resolution image can be generated. An acceleration of this process would support the spread of the localization microscopy and strongly improve its practicability.

The algorithm is initially developed in the programming platform Matlab. Its performance is evaluated using a Monte-Carlo simulation and the results are compared to already existing methods. Afterwards, the new algorithm is translated to hardware using a special graphical development environment of Silicon Software. Finally, the execution speed of the developed algorithm is determined.

FPGA-basierte Hardware-Beschleunigung von Lokalisations-Mikroskopie - Entwicklung eines Hardware-kompatiblen Algorithmus zur Datenauswertung von Lokalisations-Mikroskopie und Implementierung von diesem auf einer FPGA-Karte

Diese Diplomarbeit beschreibt die Entwicklung und die Implementierung eines Algorithmus zur Beschleunigung der Auswertung von Daten aus der Lokalisations-Mikroskopie. Das Ziel ist es diese Auswertung in Echtzeit zu realisieren. Dies bedeutet die Lokalisations-Daten in derselben Zeit zu berechnen, welche für die Aufnahme der Bilder benötigt wird. Für die Beschleunigung der Berechnungen auf den Lokalisations-Aufnahmen sollen FPGAs eingesetzt werden.

Die Idee der Lokalisations-Mikroskopie existiert bereits seit Mitte der 1990er Jahre. Diese neuartige Technik bietet eine Möglichkeit um licht-mikroskopische Aufnahmen von biologischen Proben mit gesteigerter optischer Auflösung zu analysieren. Obwohl die Lokalisations-Mikroskopie großartige neue Erkenntnisse in der Biologie und Medizin verspricht, findet sie nur sehr vereinzelt Anwendung. Eine mögliche Ursache dafür ist sicherlich die lange Auswertungsdauer, bis schließlich das hochauflösende Bild erstellt werden kann. Eine Beschleunigung dieses Prozesses würde eine weitere Verbreitung der Lokalisations-Mikroskopie begünstigen und die Anwendbarkeit deutlich verbessern.

Der Algorithmus wird zunächst in Matlab entwickelt. Mit einer Monte-Carlo Simulation soll dann die Leistungsfähigkeit bestimmt werden und die Ergebnisse anschließend mit den bereits verfügbaren Methoden verglichen werden. Anschließend wird der neue Algorithmus in Hardware übersetzt. Dafür wird eine besondere grafische Entwicklungs-Umgebung der Firma Silicon Software aus Mannheim verwendet. Im Anschluss wird die Geschwindigkeit des entwickelten Algorithmus bestimmt.

Acknowledgements

I would like to thank Prof. Dr. Udo Kebschull and Prof. Dr. Michael Hausmann for giving me the opportunity to work on the exciting field of localization microscopy.

Special thanks go to Frederik Grüll, who was supporting me the entire time, and helped me much at any questions about writing code. I would like to thank him for his efforts and for being such a good teacher and i wish him all the best for his PhD thesis.

Also, I would like to thank my step-sister Silvia, and two close friends of mine, Katharina and Diana, who agreed immediately to proof-read and edit the language part of this thesis. Thank you very much.

Finally, I want to thank my family for always supporting me during my studies. Without the mental and financial support of my parents, my studies would not have been possible in this form.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Motivation

Microscopy has developed over a long period of time. The old Romans already knew how to use water-filled glass bowls to enlarge the views of small objects. When in the 16th century the first lenses came up, the development of optical instruments began and thereby the enlargement was highly improved for the first time. With the availability of lenses it did not take very long, until the first microscope, consisting of more than one lens, came up in the late 16th century. However, the first microscopes were altogether very simple. The arrangement of the lenses was always a product of trial and error and the aberrations became too grave with bigger enlargement factors, so that the best microscopes did not surpass enlargement factors of 50.

It was not until 1846 that Ernst Abbe[1] and Carl Zeiss described a method to calculate the optical characteristics of a microscopy. This made it possible to systematically change the configuration of a microscope and, as a result, resolutions up to the optical resolution limit became reality.
The resolution defines the minimal distance of two points at which they still can be distinguished. Ernst Abbe described the optical resolution limit by

$$\Delta r = \frac{\lambda}{2NA} \tag{1.1}$$

containing the wavelength λ and the numerical aperture of the optical system $NA = n \sin\alpha$, where n defines the refractive index and α the opening angle.

Since then the resolution was connected to the wavelength of the light and it was clear that ultraviolet (UV) light should be preferred to light with a longer wave length. In order to increase the optical resolution, the numerical aperture can be optimized e.g. by using oil with a higher refractive index than air. Thus, the limit of the resolution of an optical microscope is at about 200nm.

Since Louis de Broglie has discovered the wave nature of electrons[2], it has been possible to find far shorter wavelengths, and after the development of the first electron microscope by Ernst Ruska and Max Knoss in 1932 [3] the resolution limit of 200nm was finally broken. However, beside the high resolution of electron microscopy it has some disadvantages compared to optical light microscopy. The electrons have a very high energy and hence they strongly interact with the specimen. Electron microscopy can only be used in vacuum and it requires complicated sample preparations. Furthermore, the method provides no color information at all. For these reasons optical microscopy still remains the most preferred microscopy technique in life sciences.

Nevertheless, the quality of the images of optical microscopes has been further improved by achromatic lenses and high-precision optics during the 20th century. With the development of fluorescence microscopy it was possible to bring very specified areas in cells to illumination, which highly increases the contrast of the biological structures. Therefore, fluorescence microscopy became essential in research and the development of medicine, pharmacy and biology.

The first optical microscopy technique that truly overcame the Abbe-limit was stimulated emission depletion (STED) microscopy, invented in 1994 [4]. This method is a scanning microscopy technique that uses special shaped laser-beams to increase the optical resolution. Therefore, one initial laser pulse stimulates fluorescence within the specimen. Immediately after the first laser pulse a second pulse is applied. This second laser pulse is shaped like a torus having a central intensity minimum to decrease the lateral area size, where fluorescence light is emitted.

Since then, other ideas were developed to achieve nanometer resolution. One method should be mentioned explicitly. A method that achieves sub-diffraction limited resolution without scanning the specimen line by line:

If a single molecule glows separately from the surrounding molecules, it is possible to determine its location with sub diffraction-limited accuracy. To separate diffraction spots, a stochastic on-off toggling of the fluorescing markers between a dark (non-detectable) state and a fluorescent (detectable) state of the fluorescing molecules can be used. In order to create one final image out of the calculated positions, thousands of spots on a few thousand images of the same section of the specimen have to be acquired and analyzed.

None of the new techniques can break with the physically given limit. But they provide that the taken image of the molecules can be attributed to a spot that is smaller than the common airy disk.

Compared to the well known light microscopy these new methods are highly complicated and most of them require a big effort in terms of data processing after image acquisition, until the final image can be analyzed. This data processing can last several hours and this is probably the main problem why high resolution fluorescence microscopy is still hardly used in medicine and biology.

1.2 Overview

This thesis describes an approach on how to perform these complex calculations on a highly specialized hardware. Therefore, a new algorithm is developed and finally implemented by the author. The algorithm is tested and evaluated using simulated and real image data and is also compared to existing methods. It turns out that several features have to be added to improve the quality of the final results and to guarantee the usability with various image data.

The algorithm is implemented on an FPGA-card from Silicon Software, a company from Mannheim, Germany, that provides a graphical programming tool to develop the algorithm for their frame grabber cards.

The Idea

The given problem is to speed up the processing of localization microscopy images. The problem of the existing algorithms is that they all use iterative fits to determine the center of the spots, which takes a lot of calculation time. Additionally, iterative fits can hardly be realized in an FPGA driven implementation. This is why a method is needed, that allows to calculate the precise position of the center of a spot in a fixed number of calculation steps: for example by running once or twice over all pixel values of a detected spot. The idea for the FPGA algorithm is to build the algorithm as one long pipeline that takes one pixel value in every clock cycle. Therefore, a fixed calculation effort in every step of the algorithm is necessary. The approach that is analyzed in this thesis for determining the parameters is a center of gravity algorithm that is already used in particle physics[5] on FPGAs. As it turns out, this algorithm provides nearly equivalent results as the existing methods.

The Aim of the Thesis

In this thesis the development and implementation of the new high-speed algorithm for localization microscopy is described. The localization microscopy shall be supported by a tool that accelerates the data processing and therewith, the whole work flow of this technique to realize data processing in real time. To achieve this goal the application of special hardware resp. FPGAs is analyzed. Therefore, a new hardware compatible algorithm is developed and finally implemented on an FPGA-card. The aim of this thesis is to introduce this new algorithm and to give an evaluation for the confidence of the calculated parameters. In order to support the localization microscopy with a fast and reliable tool for high resolution image analysis in real time, the whole hardware environment is finally integrated into a usable interface.

Requirements of the Algorithm

At the beginning of this thesis an algorithm has already existed. This algorithm has a high localization accuracy but a very low calculation speed. So, the algorithm that is developed in this thesis on the one side has to accelerate the data processing, including all features that the old algorithm provides and on the other hand the new method has to return parameter values with comparable accuracy.

The single steps of the algorithm can be described as follows:

1. Background elimination
2. Noise reduction
3. Signal detection and thresholding
4. Signal extraction
5. Parameter calculation
6. High-resolution-image generation

The final aim for the acceleration is the calculation of the parameters in real time. This means the resulting high resolution image should be available at the moment the last image of the sample is acquired. Today, data acquisition for images with a resolution of 0.1MPixel takes about 20-80 microseconds per image. Consequently, the total processing duration of a 350MB data stack containing 2000 images should not exceed 3 minutes in order to reach image processing in real time.

1.3 Outline

The following chapters give a fundamental understanding of the challenges of visible light microscopy. Several new high resolution microscopy techniques are introduced, especially localization microscopy is described in detail. Since most of these techniques are based on fluorescence microscopy the principle of microscopy with fluorescing molecules is presented.

Furthermore, the basic functionality of the applied hardware, FPGAs, is described. At the end of the first part of this thesis a short introduction to the used programming tools and the environment for programming the hardware components is given.

In the second chapter, some actual approaches for image processing in localization microscopy are introduced. Apart from some different ideas about hardware usage, common used mathematical approaches for data processing are described.

The theoretical part is followed by the conceptual draft for the implementation with various ideas for the single tasks of the algorithm. The development of the algorithm is split into two parts: In the first part, the final algorithm and its performance in finding the position of spots are described. This was done without hardware usage, so the first part of the conceptual draft describes the ideas about the matlab implementation and the setup of the simulation in matlab. In the second part of the conceptual draft the chosen hardware platform is described and the specialties are worked out. Furthermore, data-management for transmitting the data to and from the FPGA-card is introduced. In the following chapter the measured localization accuracy and the performance of the Matlab and the hardware implementation of the algorithm is presented, as well as the results of the comprehensive timing measurements. Finally, the graphical user interface of the Matlab implementation and the utilization of the algorithm on the FPGA-card are described.

Chapter 2

Basic Principles

This chapter gives a short introduction to the physics of optical imaging. The processes that influence the final shape of the image are explained as well as how this changes the resulting picture of the specimen. In addition, several modern microscopy techniques are introduced, in particular, there are several methods that are able to break through the optical resolution limit. A short overview about certain techniques is given and their basic principles are presented. Moreover, the functionality of the *field programmable gate arrays* (FPGAs) that are used to implement the developed algorithm, is described. The last part of this chapter contains a short presentation of the applications and the development environment that is used for important parts of this thesis.

2.1 Optical High Resolution Microscopy

In high resolution microscopy the detector only takes a picture of the reality. The optical characteristics of the microscope define how close this picture comes to the real object.
In order to describe the performance of a microscope, the ability to display a point shaped light source can be used as a criterion for the optical qualities of an imaging system. The answer of an optical system on a point like disturbance is called the *point spread function* (PSF). The PSF defines how each single point of the object is represented in the final image. Hence, the final image $g(\vec{r})$ is defined by integration over each single intensity distribution created by each single point of the specimen. Mathematically the imaging process can be described as a convolution of the object function $f(\vec{r}')$ with a certain function $h(\vec{r}-\vec{r}')$ that defines the PSF and is determined by the given instrument.

$$g(\vec{r}) = f * h = \int f(\vec{r}')\, h(\vec{r}-\vec{r}')\, d\vec{r}' \tag{2.1}$$

The convolution function $h(\vec{r}-\vec{r}')$ of a common optical microscope has a cylindrical shape with a centered maximum surrounded by wavelike structures. It can be approximated in lateral direction by the following function:

$$h(\vec{r}-\vec{r}') = \frac{\sin(|\vec{r}-\vec{r}'|\,\pi)}{|\vec{r}-\vec{r}'|\,\pi} \tag{2.2}$$

A typical PSF is shown in figure 2.1. Since the shape of the PSF determines the properties of the microscope, it can also be used to describe its resolving capability. In general, the resolution of a microscope is defined by the minimum distance of two spots in which they still can be

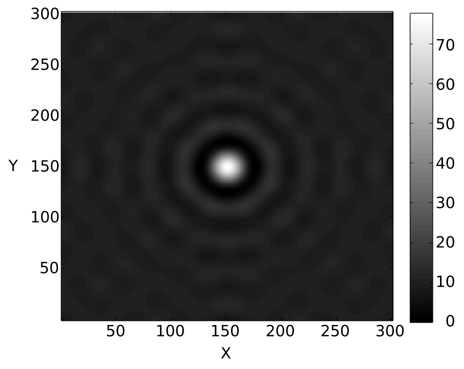


Figure 2.1: Point spread function (PSF) of a dot-like light source

distinguished. The Rayleigh criterion[6] defines this distance by half of the distance of the first minimum of the intensity distribution:

$$\Delta x = 0,61 \frac{\lambda}{NA} \tag{2.3}$$

containing the numerical aperture $NA = n \sin(\alpha)$, the refractive index n and the maximum opening angle α. However, this criterion describes only the physically limiting factors. In order to reach this limit, the microscope has to provide very good optical properties.

Due to aberration, chromatic aberration and irregularities of the lenses, the imaging properties of a microscope are further decreased. All these facts also have an influence on the PSF of the imaging system. So in general, the lateral dimensions of the PSF are larger than it would have been defined by the physical limit.

Due to the fact that the convolution of the object function with the PSF is a very complex operation in local space, it is convenient to switch to the frequency space. The convolution theorem says that a convolution in local space changes in frequency space to a simple multiplication.

$$\tilde{F}(r) = F(r) \cdot G(r - r') \tag{2.4}$$

As a result, the final image can be determined by a multiplication of the Fourier transform of the object function with the Fourier transform of the PSF. The Fourier transform of the PSF is called *optical transfer function* (OTF).

This process of optical Fourier transformation can be described by the theory of Fourier optics. It shows how a lens can serve as a Fourier transformer:

If parallel light of the object plane hits a lens, the Fourier transform of the light source is created in the focal plane of the lens (Fig. 2.2). To detect all information of the object plane all frequencies from $-\infty$ to $+\infty$ have to be transmitted. This requires an opening angle of 180 degrees. Since an opening angle of 180 degrees is technically not feasible, an optical system can only detect frequencies up to a maximum value k_{max}. Thus, a microscope always functions as a low-pass filter on the original image. This means that small structures can only be resolved up to a minimum size.

The following example shows how this affects the image of a small glowing spot:
A very small dot can mathematically be described by the delta-function. The Fourier transform of the delta-function is constant in the entire frequency space. Since an optical system applies a low-pass filter on the image, the information of higher frequencies are lost. Consequently,

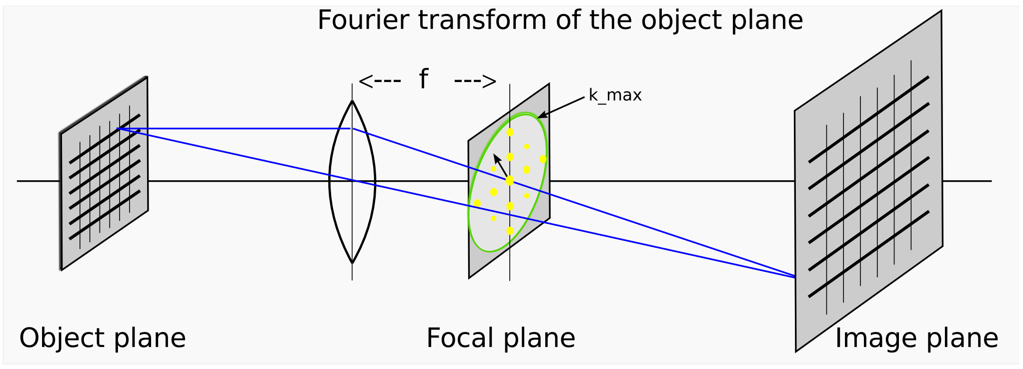


Figure 2.2: Optical systems as a low-pass filter: a lens creates the Fourier transform of the image in its focal plane. The further optics can only transmit frequencies up to the maximal frequency k_{max}. Information of higher frequencies are lost.

the image of a small point results in the so called *airy disc*. Within its first roots, the airy disc can be assumed as a two-dimensional Gaussian distribution.

Background and Noise

In general, the principle of visible light microscopy is always the same. The specimen is illuminated from one side, this can be from top or bottom, and the scattered light is focused on the detector. The detected light has its origin in the ideal case only in the focal plane. However, due to the axial expansion of the specimen, it is impossible to avoid the detection of light from outside the focused regions. This light from the background decreases the contrast and can even cover the structures of interest.

Additionally, there is noise that decreases the quality of the image. The value of each pixel of an image represents the amount of detected photons. As in each measurement of physical quantities, the detected result has fluctuations. These fluctuations are described by noise. In microscopy, the photon noise is assumed to be Poisson distributed with mean value N and standard deviation $\sqrt{N}$. Therefore, an intensity of N photons will fluctuate about $\pm\sqrt{N}$. By a higher intensity N, the relative fluctuations will decrease and the *signal to noise ratio* (SNR) will increase with the number of photons.

$$SNR = \frac{N}{\sigma} = \frac{N}{\sqrt{N}} = \sqrt{N} \tag{2.5}$$

Beside the photon noise, the detector itself has uncertainties. In order to analyze the detected signal, the analog value in the detector has to be transformed into a digital value. This readout noise is a camera specific quantity. In normal photography the read out noise plays only a minor role, but in microscopy, where the acquisition times are short and the detected intensities are very low the noise of the detector can become a limiting factor.

2.2 Fluorescence Microscopy

Fluorescence was first described in 1852 by Sir George G. Stokes. It is the attribute of a material to absorb high energetic light and emit light with a longer wavelength. In nature there are several different occurrences of fluorescence. A lot of minerals e.g. calcite emit visible light, when being exposed to an ultraviolet light source.

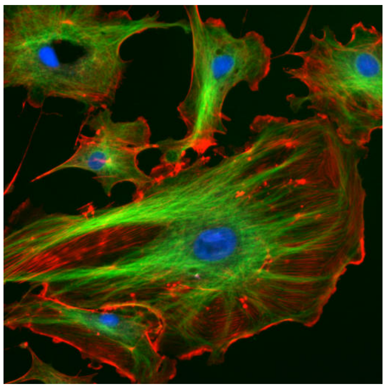


Figure 2.3: Endothelial cells under the microscope. Containing blue, green and red marked structures[1].

The physical process of fluorescence is shown in figure 2.4. A molecule at room temperature is generally in a low energy state, the ground state. When a photon having sufficiently high energy hits the molecule, the molecule can absorb the entire energy of the photon and get into an excited state. The excited molecule instantly dissipates a part of the absorbed energy to vibrations or to collisions with surrounding molecules until a state with a larger energy gap is reached. To finally reach the initial ground state, the molecule has to emit a photon carrying the energy of the width of the gap. In practical applications the loss of energy to vibrations and collisions is very important, since this is the reason for the shift of the maximum of the emission spectrum against the maximum of the absorption spectrum. This energy difference makes it possible to separate the fluorescence light of the specimen from the light of the source using chromatic filters.

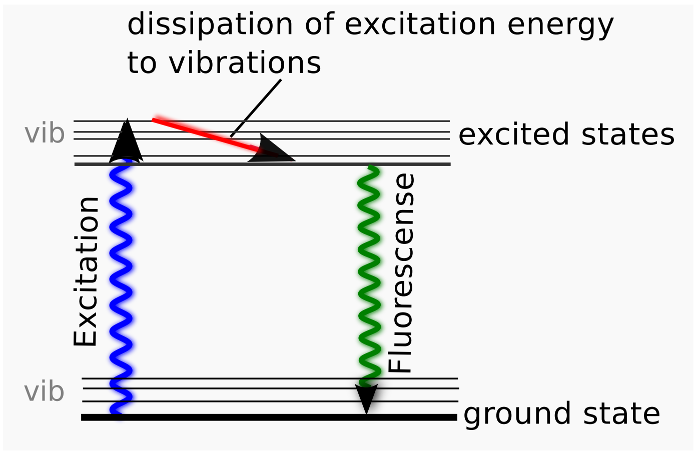


Figure 2.4: Energy scheme of a molecule excited by a high energy photon and emitting a photon having less energy

Not only minerals show this behavior. There are also several fluorescing substances in biological material: e.g. chlorophyll in plant cells show fluorescence (autofluorescence) as well as several proteins.

The first fluorescence microscopes were introduced between 1911 and 1913 by Otto Heimstädt and Heinrich Lehmann. They were built to observe autofluorescing structures in ani-

[1]Source: ImageJ, “Fluorescent cells.” [Online]. Available at: http://rsb.info.nih.gov/ij/images/FluorescentCells.jpg

mal and plant cells. The breakthrough of fluorescence microscopy came in 1940, when Albert Coons developed a method that made it possible to label certain structures with a fluorescing protein. This was realized by using antibodies which can be prepared in a way that they only bind with the desired cell structures, e.g. certain parts of the DNA, membranes or other cell structures.

Fluorescing Markers - GFP

Fluorescence microscopy started in a new era, when Douglas Prasher isolated the DNA of a green fluorescing protein in 1992, the GFP[7]. It is a natural protein of the jellyfish Aequorea Victoria and, in contrast to most other fluorescing proteins, it has no photo-toxic impact on most cell types. Since Prasher figured out how to connect the GFP with other proteins, it was possible to bring any protein in a cell to autofluorescence. Thus, the temporal development and the spatial distribution of the marked proteins or certain cell types can be analyzed in vivo.

To mark a protein with GFP, the genome for GFP expression has to be inserted into the DNA sequence near the desired protein. Consequently, wherever the tagged protein is produced, the GFP will be expressed too.

The original GPF of A. Victoria has two excitation peaks in the visible light spectrum. The major peak is at 395nm (purple) and another minor one at a wavelength of 475nm (blue). The emission peak is at a longer wavelength of 509nm (green). Once GFP was discovered, several mutants of the original GFP that provide enhanced glowing stability and an improved photo-efficiency[8] were found. Also, GFP derivatives with emission peaks at different wavelengths were introduced.

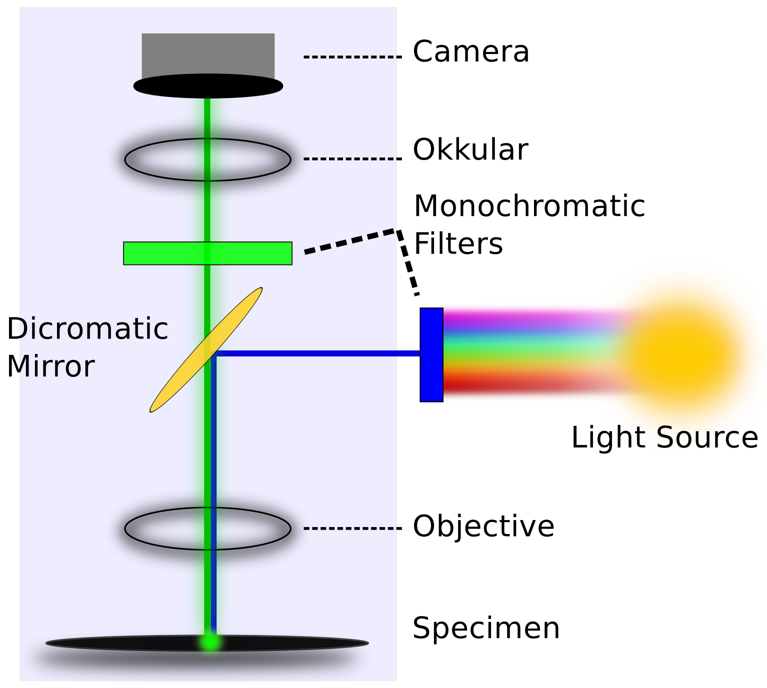


Figure 2.5: Setup of a fluorescence microscope. The dichromatic mirror reflects only the high energy light of the light source. The fluorescence light passes the mirror and reaches the detector.

The most important derivatives show fluorescent behavior in the yellow band of the visible light spectrum, e.g. the YFP, which has an excitation peak at 514nm and an emission peak at 527nm. Another marker is a cyan fluorescing protein called CFP that has an emission peak at 480nm [9]. The variety of marker colors allows the marking of various structures within the same sample with different colors, so that multicolor fluorescence images with an highly increased contrast at the different marked structures can be created.

The setup of a typical fluorescence microscope is shown in figure 2.5. The specimen is illuminated by light of a certain wavelength. The light source can be a multichromatic lamp having a chromatic filter in front, or a monochromatic laser that emits the desired wavelength. The light from the source is reflected by the dichromatic mirror. The only light that passes the mirror and reaches the detector is the fluorescence light from the sample. Thus, only the marked structures can be seen in the acquired image. This process strongly increases the contrast of the observed structures. Additionally, it is possible to mark different structures with various markers within one sample. By using several exciting wavelengths, a multi chromatic picture that shows incredible details can be created (Fig. 2.3).

2.2.1 Confocal Microscopy

The quality of images of fluorescence microscopy suffers strongly from out-of-focus light. In order to suppress this unwanted effect, the technique of confocal microscopy has arisen. It was developed and patented by Minsky in 1961 [10] and first experimentally realized in 1979 [11]. At the same time, benefiting from recent improvements in computer and laser techniques, laser scanning confocal microscope was developed by C. Cremer[12][13] in Freiburg. Furthermore, since the 1990s the principle of confocal microscopy was further improved and today there are commercial confocal microscopy systems available.

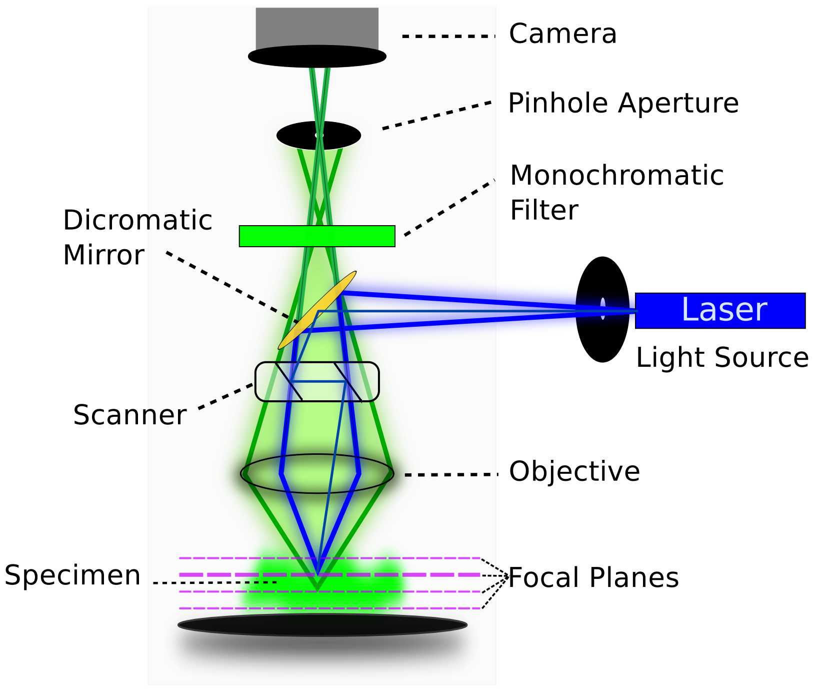


Figure 2.6: Schematic setup of a confocal microscope with two pinhole apertures.

Principle

The setup of a confocal microscope is shown in figure 2.6. A point-like light source is focused on the specimen. The reflected light is collected by the focusing objective and is separated afterwards from the light from the source by using a dichromatic mirror like in fluorescence microscopy. The difference is that the fluorescence light is additionally focused on the center of a pinhole aperture. Thereby, the fluorescence light of points below or above the focal plane is hardly confocal with this aperture and forms highly magnified airy discs in the pinhole plane. This is why the intensity of light from outside the focus plane is strongly decreased on the detector, which results in a far better contrast of in-focus structures (Fig. 2.7).

To reconstruct an entire image, the specimen has to be completely rasterized. The imaging speed is defined by the speed of the scanning instruments. In order to provide high frame rates for in vivo imaging, systems of computer controlled rotatable mirrors are needed.

In addition to the strong contrast of the images, confocal microscopy provides three dimensional imaging of fluorescing samples. Since only light from the focus plane is detected, it is possible to localize the detected structures in the axial direction. After several images of different focal planes have been acquired, a three dimensional computer model of the sample can be generated.

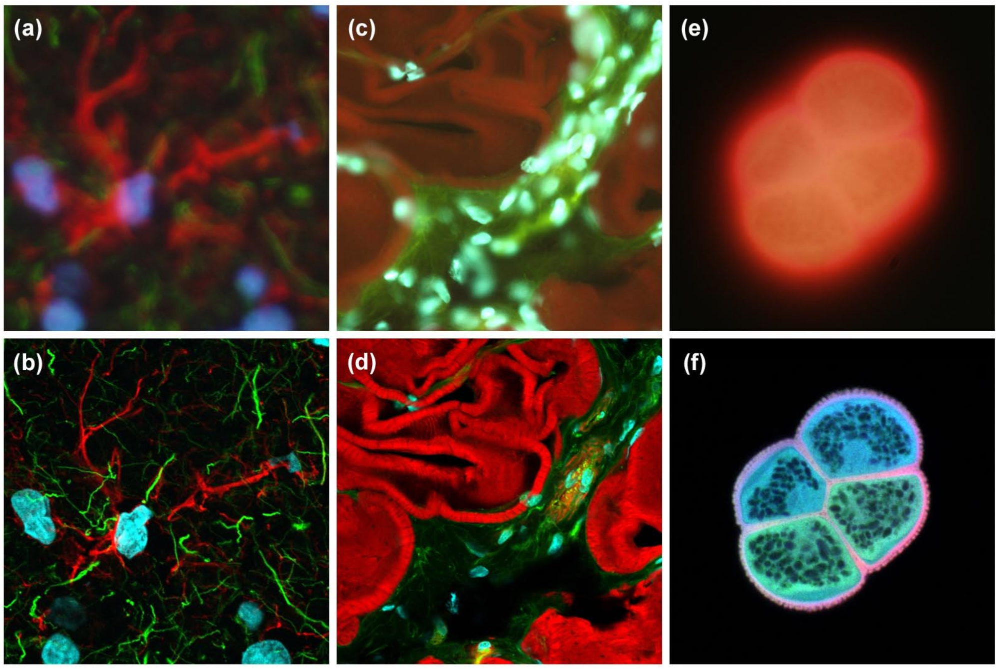


Figure 2.7: Comparison of wide-field (upper row) and laser scanning confocal fluorescence microscopy images (lower row). Note the significant amount of signal in the wide-field images from fluorescent structures located outside of the focal plane. (a) and (b) Mouse brain hippocampus. (c) and (d) Thick section of rat smooth muscle. (e) and (f) Sunflower pollen grain[13].

Details

The principle of confocal microscopy can be described as follows:
The excitation light is focused on the specimen in a very small spot with the highest intensity in an area of about 0.28 to 0.8 micrometers in lateral direction and 0.5 to 1.5 micrometers in axial direction. This small spot is additionally focused on the pinhole aperture in front of the

detector. In this way, the pinhole aperture of the light source and the other pinhole aperture in front of the detector are *confocal*.

The small spot is moved line by line over a selected area of the sample. At the method of laser scanning confocal microscopy, the illumination point is moved by a system of two mirrors that oscillate with very high speed driven by computer controlled galvanometer motors. Thereby, one mirror moves the spot along the x direction and the other mirror proceeds it in the y direction. Usually,only one spot is used for scanning. However, there are also systems that use more than one spot for imaging. By using a system of rotating mirrors and one laser spot, a frame rate of about one frame per second can be achieved[13].

In order to create three dimensional images the sample is cut optically into thin slices that range from 0.5 to 1.5 micrometers, which requires a very precise step motor. Using such thin slices up to 50 micrometers in axial direction can be analyzed and transformed into a three dimensional computer model.

The advanced contrast improvement of the biological structures and the ability to create three dimensional images are primary advantages of confocal microscopy over the common wide-field methods.

2.2.2 STED

The first microscopy technique that finally broke the limit of Abbe using visible light was stimulated emission depletion (STED) fluorescence microscopy[4, 14]. It was theoretically described for the first time by Hell and Wichmann in 1994 and technically realized in 1999 in Hell's group [15]. Since 2007, commercial STED microscopes are available. STED microscopy is a scanning laser fluorescence microscopy technique similar to confocal microscopy.

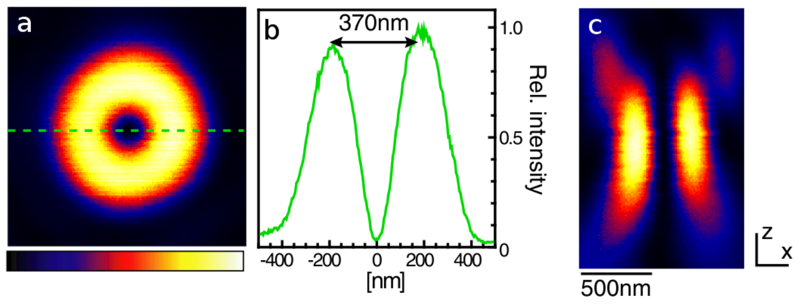
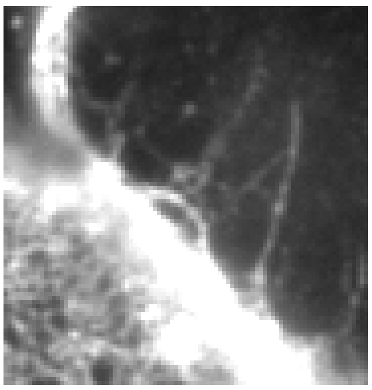


Figure 2.8: a) Torus shaped intensity distribution within the focal plane (lateral) and b) its central profile.c) Focal intensity distribution along the optical axis (axial)[16].

The detectable fluorescence light can only originate in the region, which is hit by the exciting laser spot. Due to diffraction, the minimal size of a spot in the focal plane is about half of the wavelength of the used light. To gain a smaller extension of the spot, the first laser pulse is immediately followed by a second red shifted laser pulse of the same wavelength as the fluorescence light that is emitted by the molecules in the sample. The so called *STED-pulse* has a torus shaped intensity distribution with an intensity minimum in the middle. It is aligned around the center of the first excitation laser pulse. Using stimulated emission, the STED-pulse induces at the stimulated regions immediate transition from the excited state to a lower state before fluorescence can occur. Due to the special shape of the STED-pulse, the edges remain dark and only the innermost region of the illuminated area can contribute to the fluorescence signal. The size of the emitting region decreases with higher intensity of the STED-pulse. By scanning the specimen with this narrowed spot, structures with a size below the diffraction limit can be revealed.

2.2.3 Localization Microscopy - STORM and SPDM and PALM

Due to diffraction, the image of a glowing molecule will always expand to an airy disk with a size of about half of the wavelength of the detected light. However, if the spot of a single molecule is optically separated from surrounding signals, the exact position can be determined with an uncertainty far below the diffraction limit of 200nm. This is the principle of localization microscopy methods such as STORM, PALM, or SPDM[17, 18, 19]. In conventional fluorescence microscopy a certain area of the specimen is illuminated and all of the fluorophores in this area are glowing at the same time. With the discovery of additional dark states or metastable triplet states of the fluorescing molecules, it became possible to alternate the fluorescing behavior of the markers. Several techniques are introduced in [20, 21, 22, 23, 24]. The original idea in *stochastic optical reconstruction microscopy* (STORM)[18] and *photo activated localization microscopy* (PALM) [19] was to use molecules that can be switched between a fluorescing state and a non fluorescing stable dark state. At the beginning of the imaging process, all the fluorophores are in the ground state. After illumination with light of the excitation wavelength the fluorescence of the molecules is activated and the molecules start to alternate between the fluorescent state and the ground state (Fig. 2.4). Additionally, there exists a low probability that molecules might get into the metastable dark state and remain there instead of returning into the ground state. The more molecules remain in the dark state, the more decreases the brightness of the fluorescence. The few remaining molecules in the fluorescing state can be detected optically separated.


(a) Widefield image of a cell membrane

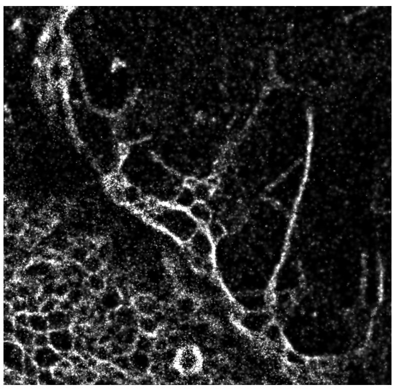

(b) Localization image. The single marked locations of the detected signals can still be recognized in the image.

Figure 2.9: Resolution enhancement of localization microscopy by a factor of 10

If most of the molecules remain dark, the fluorescence can be reactivated using a laser pulse of another wavelength. The photons of the reactivation laser can pump the molecules from the dark state back to the fluorescing state. Since this is a stochastic process, the intensity of the laser pulse and its duration determine the amount of reactivated fluorophores. If only a fraction of the molecules is switched back, single fluorophores can be imaged and their precise location can be determined. To take an image of all of the fluorophores in the field of view, the switching process has to be repeated several times. Typically image stacks from 1000 to several 10000 images are taken to get one final high resolution image[18, 19, 17]. The number of images containing blinking molecules, that can be acquired depends on the applied fluorophores in the specimen, because steady exposition to laser light can destroy the fluorescing molecules. This effect is called *photobleaching*. However, this effect can also be used intentionally to destroy unwanted fluorescence signals such as the autofluorescence of a cell in order to increase the contrast.

Latest methods do not require a stable dark state to achieve blinking anymore [22]. The important fact for the blinking is, that the fluorophore remains much longer in a dark state ($\sim 10^4$ times) than it does in a fluorescing state. Before a molecule returns to a dark state, a certain amount of photons depending on the type of fluorophore is emitted. The final accuracy of the position of the molecule, rises with the number of emitted resp. detected photons. With σ, the standard deviation of the distribution, and N, the number of photons, the localization error is described by $\Delta x = \frac{\sigma}{\sqrt{N}}$. With a photon count of up to 10^4 photons detected in one glowing phase, it is possible to achieve a localization accuracy of single molecules down to a few nanometers and structural resolutions up to 20nm [18]. Compared to the wavelength of visible light and the optical resolution limit of about 200nm, this defines an increase of the resolution by a factor of 10. However, due to the high processing duration until the final high resolution image can be generated, this promising method is still hardly used.

2.3 Field Programmable Gate Arrays (FPGAs)

The preferred machine to perform complex calculations is usually the common workstation PC. This machine consists of fixed hardware components that runs variable resp. programmable code that is serially executed by the *central processing unit* (CPU). The computation speed is defined by high clock frequencies of up to 4 GHz that are applied in actual processors. High flexibility and fast processing speed is the secret of success of actual workstation computers.

Another approach to implement an algorithm is to use special hardware that mostly provides better performance in parallelizable problems. This hardware can either be really specific like an *application specific integrated circuit* (ASIC) or commercially purchasable and freely programmable like an FPGA. The advantage of an FPGA in contrast to an ASIC is, in addition to its far lower costs, that it can be reused if the requirements of the implemented algorithm change.

a_{in}	b_{in}	c_{in}	d_{out}	$carry_{out}$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 2.1: Truth-table of a two bit adder.

$$d_{out} = \overline{ab}c + \overline{a}b\overline{c} + a\overline{bc} + abc$$
$$carry_{out} = \overline{a}bc + a\overline{b}c + ab\overline{c} + abc$$

$[\overline{n} \equiv not\, n]$

$[+ \equiv OR]$

$[ab \equiv a AND b]$

Computing with Hardware

In the following a short introduction in calculating with hardware is given.
Information technology only knows two digital states: zero and one. These are analog measurable sizes, represented by voltage levels. The voltage levels can be set by switches. On a chip, these switches are transistors that operate also with the high and low voltage of the signals

[25]. Using transistors, it is possible to create logical functions such as *AND*, *OR* and *NOT*, e.g. the latest CMOS technology uses *NAND* and *NOR* functions. These are very simple functions consisting of two (*NOT*) resp. four (*AND, OR, NAND, NOR*) transistors. The first programmable hardware devices described exactly these functions consisting of *AND*- and *OR*-arrays with flexible connections (Fig. 2.10).

However, using these functions as building blocks, almost each function of any number of inputs can be created. For better understanding, the truth table of a simple adder with carry is shown in table 2.1.

The first programmable devices have been *Programmable logic arrays* (PLAs) or later *complex programmable logic devices* (CPLDs) that have either fixed *AND* or fixed *OR* connections, the other logical functionality remains programmable. In order to achieve higher complexity, more connections and thereby longer wires are required. However, having longer wires means having slower devices. An approach to solve this problem and to provide fast devices with increasing complexity was presented by Xilinx in 1985, when Xilinx introduced the first *field programmable gate array* (FPGA). This was a very complex chip that was able to realize more extensive functionality than the existing programmable devices. Through specific configuration, it is possible to realize small functionality in an FPGA, such as simple counters or adders, as well as high complex circuits like entire microcontrollers. Since FPGAs are re-programmable hardware devices, they are mainly used for the development and fast prototyping of ASICs or the support and acceleration of calculation-intensive algorithms.

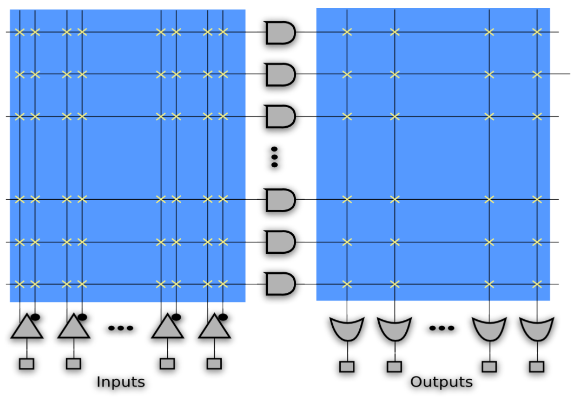


Figure 2.10: Simple programmable device with AND and OR gates

Using FPGAs, it became possible to introduce compact and product specific circuits also in small lot sizes. Due to the lower costs and increasing performance of FPGAs, they become more and more an attractive alternative for the production of ASICs, which is always very expensive.

Although FPGAs of different vendors differ in their specific functionality, the basic setup is always the same: The logic functionality is provided by a number of small logic units, the so called *complex logic blocks* (CLB). The CLBs consist of up to four slices that contain the programmable functionality. A typical slice of an Virtex 4 FPGA by Xilinx is shown in figure 2.11. The main part of the slice are the *lookup tables* (LUTs),which are programmable SRAM cells with 4 to 6 inputs and one output. For each pattern at the inputs of the LUTs, the output can be set to one or zero. In this way, any function of the input signals can be realized. In order to provide more complex functions of e.g. more signals, the neighboring slices are directly connected. The additional logic provides further functionality like registered or direct signals, clocks or fast carry logic. Carry logic is needed e.g. for arithmetic operations such as large

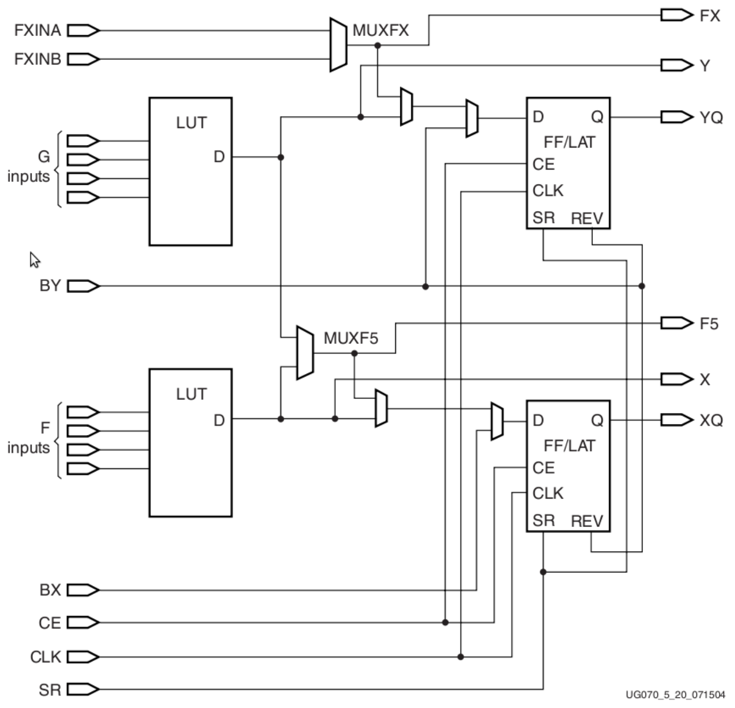


Figure 2.11: One slice of a Xilinx Virtex 4[1], containing two LUTS for the logic functionality and two Flip-Flops to synchronize the signals.

adders. Due to performance reasons, it is very important to place the functionally close-by CLBs also locally nearby.

The development of the final functionality of the FPGA is strongly supported by tools that are provided by the FPGA manufacturer. In general, the designing process is limited to the description of the basic behavior of the logic circuits. The later synthesis steps and the translation into the available hardware resources are taken over by special tools.

Connection Matrix

Apart from the interconnections between neighboring slices, a large bus system lays between the CLBs, which is also configurable. It provides fast signal transfer between neighboring CLBs as well as signal distribution over the whole chip using special long lines. The interconnection between the horizontal and the vertical wires is realized by programmable connection matrices in a way that each signal can be routed in any direction.

Memory and Other Components

Additionally, an FPGA has several types of internal memory. For bigger memory requirements, several areas within the chip are filled with *block ram cells* (B-RAM) that can keep up to 32 KB of memory. For even more memory requirement, external memory chips such as flash-memory or DDR-RAM can be connected with an FPGA. Very small storage can even be provided by the LUTs that can also be used as small programmable memory. Most of the FPGAs have additional fix hardware components having certain functionality. For instance there are blocks for

[1]Source: Xilinx, Virtex 4 FPGA User Guide UG070 (v2.6) December 1, 2008.

input and output. For computationally intensive designs, most of the FPGAs contain built-in multipliers that allow a multiplication in only one clock cycle. In order to create special clock-signals, FPGAs contain integrated *phase locked loops* (PLLs) and *delay locked loops* (DLLs) that provide clock operations such as phase shifts or clock multiplication for faster clocks. Typical applications for phase shifts are controlling DDR-RAMs or synchronization problems of internal clocks with external signals.

Additionally, some FPGAs contain even whole CPUs, so called *hard-cores* like the Xilinx Virtex 5 that contains an integrated 32 Bit RISC Power PC by IBM having a clock speed of 400 MHz [26]. Besides, it is also possible to implement a CPU or even a whole computer system into an FPGA. However, generated CPUs are slower and need a lot of logic cells. But in some cases a serial working CPU is the more convenient solution. But there is no need to reinvent the wheel. There are CPUs and other precompiled operations and net-lists, called *soft-cores* or *IP-cores*, provided by the FPGA vendor or purchasable in libraries.

Building Hardware

The work flow for making an FPGA design work is always similar. The developer describes the behavior and the algorithms of the FPGA with a hardware description language (HDL) in a text editor. The most common hardware description languages are VHDL and Verilog[27, 28]. For the following steps the FPGA vendor provides special tools that take over the rest of the translation process. First of all, the code is checked for syntax errors, as in any other compiler. The next step is the translation of the written HDL into logical elements. This process is called synthesis. It creates a netlist of standard primitives defined in the SIMPRIM library. At this moment, a simulation of the behavior is possible. In a behavioral simulation, each signal can be checked whether it arrives at the right time (clock period) at the expected place. If everything is alright, the synthesized hardware then is mapped on LUTs, Flip-Flops, and other resources of the chip. The last step is to place and route each single part of the design.

The duration of each step scales with the size and complexity of the design and can take up to several hours on a normal computer. The most time-consuming step is the place and route procedure. The more the chip is filled the longer it takes to find a fitting location for each part of the circuit. Since the placement of thousands of connected logic functions can become a really complex problem, it can occur that connections between two blocks become too long for the signal to propagate within one clock cycle. Therefore, the entire functionality of the FPGA has to be checked, and only if all timing constraints are met, the implementation can work correctly.

2.4 Important Programs Used for the Development

This chapter will shortly introduce the programs that are used to develop the main parts of this thesis. For the development of the algorithm the programming environment Matlab is used. In order to implement the algorithm on the FPGA, the graphical hardware development tool VisualApplets is applied. This section is not supposed to serve as a user manual but only as a short presentation of the abilities and advantages of the named tools.

2.4.1 VisualApplets

Silicon Software is a company based in Mannheim, Germany, which develops frame grabber cards for industrial applications. A frame grabber card is a plug-in PCI-Express card to extend a workstation computer system. It contains basically a programmable FPGA, external memory, and a camera link. The card can grab and process images at very high frame rates or use the images as a trigger for automation processes. The on-board FPGA is able to perform individually programmable tasks at high speed. The development of the algorithms for the FPGA on the frame grabber is done with a special environment, provided by Silicon Software, called VisualApplets. In contrast to the common work flow, the algorithm is not described in text form. VisualApplets provides graphical hardware development using a library of operators optimized for image processing.

An operator can be a simple bit-shift as well as a highly complex image filter operation. The significant advantage of describing hardware using a high level tool is that the timing analysis in the gate level is completely done by the tools. The user does not need to analyze waveforms and to concern about transitions of signals in the right clock cycle. This saves a lot of developing time and the user can mainly focus on the algorithms.

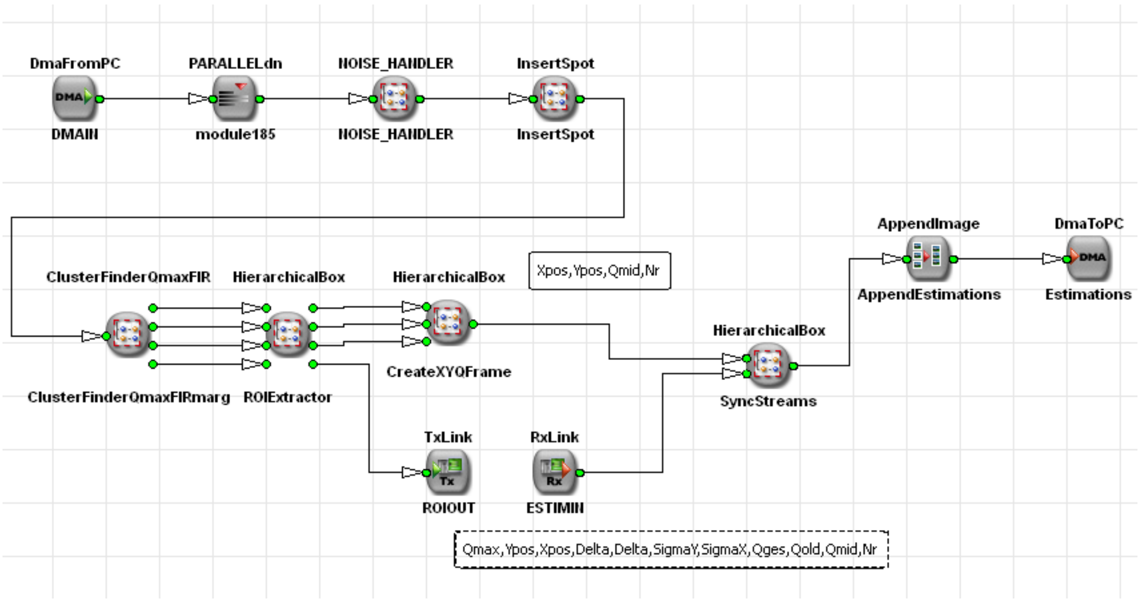


Figure 2.12: Data-path in VisualApplets

In order to get an idea of how processes are developed in VisualApplets, a short introduction to the characteristics of VisualApplets is given:
VisualApplets is basically made for image processing. Thus, each connection between two operators defines the path of a whole image. E.g. the '+'-operator describes the pixel-wise sum of two images of equal dimensions.

However, the image is not processed completely in parallel. With each clock cycle a new pixel arrives at the beginning of the data path. The data path defined by the operators is built as a long pipeline through which pixel values propagate. The length of this pipeline defines the latency of the calculations, meaning the number of clock cycles until the first valid result reaches the output. The calculation speed is defined by the clock speed.

In VisualApplets there are two types of operators called **O**-operators and **M**-operators. The first type mainly includes arithmetical-, logical- and shift-operators. **O**-operators can be combined in any way. **M**-operators define all kind of memory operators. When using **M**-operators special synchronization rules have to be taken into account.

Apart from the synchronization rules, there is another important restriction in VisualApplets. The pipelined data path is a strictly serial structure. This means that no loops are allowed in the data path. This is a very strict restriction since in this way it is really hard to reuse former results for later calculations without implementing the functionality twice.

The reason why not everyone uses high level tools for hardware development is that such tools have always limitations. These tools produce always overhead in hardware structures and do not reach the optimization level that could be achieved manually. In addition, the provided libraries determine the way an algorithm has to be described. Mostly, high level tools simplify the work for the developer, however, certain structures can be blown up and their implementation can become much more complicated than in the common text implementation. Another limitation of these tools is that most of the hardware that is produced cannot reach the highest performance level that can be realized on the chip.

2.4.2 Matlab

Matlab is a commercial program, developed by The MathWorks[29] that provides a large collection of math libraries in an easy usable environment. Especially for numerical matrix calculations image processing and numerical simulations it shows a high practicability with acceptable performance. Using Matlab, visualization of image data as well as performance tests can easily be realized. Additionally, Matlab provides a tool to generate basic *graphical user interfaces* (GUIs).

The latest version of Matlab 2011a also includes support of parallel computing[30]. The Matlab parallel computing toolbox contains functions that support computing on multiple CPU cores, computer clusters or even distributed computing in a computer cloud as well as support for calculations on NVIDIA graphics cards.

Chapter 3

State of the Art

This chapter focuses on the actual approaches of data processing in localization microscopy. There are several methods to calculate the precise mid-point of a Gaussian-like distribution. The most popular one is a least squares fit method which is proposed in the first section of this chapter. Another approach is the method of maximum-likelihood which is also presented. Further, a totally different computation method is mentioned that is using hardware to speed up the calculations. This method is based on an implementation on graphics cards.

3.1 Mathematical Methods of Data Processing

In science it occurs very often that a variable can not be measured directly or only with insufficient accuracy. Therefore, a set of data is acquired to increase the precision of the measuring process by further data processing. If the collected values contain errors like noise or statistical fluctuations the error of the achieved parameter X decreases with the amount N of data points like $\Delta X = 1/\sqrt{N}$. Due to fluctuations, a straight interpolation will often fail to determine the desired parameter.

The general problem can be described in the following way:
A function $f(x_i, a_n)$ that approximates a number N of data points $(x_1, \ldots, x_i)_{1\ldots N}$ and that compensates most of the measurement errors. The function f has a given shape that suits to the corresponding problem. Additionally, the function contains a set of parameters a_n which have to be defined during the fitting process.

In the case of localization microscopy the dataset is a measured intensity distribution of a glowing spot, detected by the camera. Depending on the microscope, the camera and the light source, the dataset consists of a count N of intensity values that represent the airy disk. The airy disk can mathematically be approximated by a two-dimensional Gaussian distribution. The Gaussian function that is fitted to the data set is represented by

$$f(x,y) = \frac{A}{2\pi\sigma_x\sigma_y} exp\left(-\frac{(x-\mu_x)^2}{2\sigma_x^2}\right) exp\left(-\frac{(y-\mu_y)^2}{2\sigma_y^2}\right) + B + g(x,y) \tag{3.1}$$

with

- the coordinates of the data points x, y
- the width of the distribution σ_x and σ_y
- the entire intensity value A of the distribution

- the normalization factor $\frac{1}{2\pi\sigma_x\sigma_y}$
- the coordinates of the real center of the distribution μ_x andμ_y
- a parameter for the constant background level B
- and an optional function that describes a ramp in the background $g(x,y)$.

This is a function of six parameters at minimum that have to be defined. If one of the parameters is given by the setup or previously known by another measurement , it can be set to a constant value. Especially the parameters that describe the shape of the background can be eliminated in most cases. The few unknown parameters in the Gaussian distribution describe a big advantage of this distribution function for fitting it to measured data. Generally In science, it is very important to know the uncertainty of the determined parameters. For this reason, the applied fitting method has to provide the confidence interval resp. the errors of the calculated values.

3.1.1 The Method of Least Squares

The method of least squares[31] was developed in the simplest form by Legendre and Gauss. Because it is one of the most widely used statistical methods and due to its practical significance, especially in localization microscopy, a short overview of the algorithm will be given. In chapter 6 the performance of one least squares method from the Matlab library will be presented and compared to the results of other algorithms.

The problem is to find a mathematical function that describes the values of a measurement. Normally, a model function containing several unknown parameters $\mathbf{a} = a_j(j = 1...m)$ is given. It can be assumed that the model function fits the problem. In order to solve the problem, the method of least squares minimizes the sum of the squared differences of the measured values and the function values by varying the parameters a_j:

$$\chi^2 = \sum_{i=1}^{n} g_i [y_i - f(x_i;\mathbf{a})]^2 \quad \rightarrow \quad \min_{\mathbf{a}} \left(\chi^2(\mathbf{a})\right) \tag{3.2}$$

With the weighting factors g_i. The factors g_i provide that values with a higher certainty have a higher influence on the result:

$$g_i = \frac{1}{\sigma_i^2} \tag{3.3}$$

In statistical processes e.g. Poisson noise the error of every measured value is:

$$x_i = N_i$$
$$\sigma_i = \Delta x_i = \sqrt{N_i} \tag{3.4}$$

If $f(x_i;\mathbf{a})$ is a model function with linear parameters,

$$f(x_i;\mathbf{a}) = \sum_{j=1}^{m} a_j \cdot \varphi_j(x_i) \tag{3.5}$$

the minimization of χ^2 can be done straightforward by setting the first derivative to zero:

$$\frac{\partial \chi^2}{\partial a_k} = 2\sum_{i=1}^{n} g_i \varphi_k(x_i) \left[y_i - \sum_{j=1}^{m} a_j \varphi_j(x_i) \right] = 0 \qquad k = 1,...,m \tag{3.6}$$

The resulting equations define an inhomogeneous system of m linear equations with m different fitting parameters.

$$\mathbf{A}\cdot\mathbf{a} = \beta$$

$$\sum_{i=1}^{n}\sum_{j=1}^{m} g_i\Big(\varphi_k(x_i)\varphi_j(x_i)\Big)a_j = \sum_{i=1}^{n} g_i y_i \varphi_k(x_i) \qquad k = 1,...,m \tag{3.7}$$

In order to calculate the error of the fitting parameters a_j the covariance matrix $\mathbf{C}$ has to be determined. This is done by inverting the normal Matrix $\mathbf{N}$:

$$N_{ij} = \frac{1}{2}\frac{\partial^2\chi^2}{\partial a_i \partial a_j} = \sum_{k=1}^{n} g_k \varphi_i(x_k)\varphi_j(x_k) \tag{3.8}$$

$$C = N^{-1} \tag{3.9}$$

The diagonal elements of the covariance matrix define the squared error of the parameters:

$$\sigma_{a_j} = \sqrt{c_{jj}} \tag{3.10}$$

Additionally, the covariance matrix contains the correlation coefficients of the fitting parameters:

$$r_{ij} = \frac{c_{ij}}{\sqrt{c_{ii}c_{jj}}} \tag{3.11}$$

The values r_{ij} define how the parameters a_i, a_j correlate. Examples can be found in literature.

Non Linear Model Functions

If the model function is not linear, the first derivatives (Equ. 3.6) result in a real complex system of equations that often cannot be solved analytically. However, it is possible to linearize the equations by replacing the function with the Taylor series up to the first derivatives.

$$f(x_i;\mathbf{a}) \approx T(x_i;\mathbf{a}) = f(x_i;\mathbf{a}^0) + \sum_{l=1}^{m}\left(\frac{\partial f(x_i,\mathbf{a})}{\partial a_l}\right)_{\mathbf{a}=\mathbf{a}^0}\cdot(a_l - a_l^0) + \ldots \tag{3.12}$$

This is called the *Gauss-Newton-Method* (GN-Method). The GN-Method is an iterative approximation algorithm that approaches iteration by iteration the optimal parameters of the model function. Before the first iteration can be calculated, the values of the parameters $\mathbf{a}^0$ have to be guessed in order to define an acceptable starting point for the calculations. The result of a single iteration is a set of new parameter values $\mathbf{a}^{n+1}$ that lay closer to the optimal values than the previous values $\mathbf{a}^n$. These improved values define a new starting set for the next iteration and so on.

If the approximation $T(x_i;\mathbf{a})$ is inserted in equation 3.2, the first derivatives (Equ. 3.6) will also result in a system of linear equations:

$$\mathbf{A}\cdot(\mathbf{a}-\mathbf{a}^0) = \beta \tag{3.13}$$

with

$$\alpha_{ij} = \sum_{k=1}^{n} g_k \frac{\partial f(x_k,\mathbf{a})}{\partial a_i}\frac{\partial f(x_k,\mathbf{a})}{\partial a_j}\Bigg|_{\mathbf{a}=\mathbf{a}^0} \tag{3.14}$$

and

$$\beta_i = \sum_{k=1}^{n} g_k \left(y_k - f(x_k; \mathbf{a}^0) \right) \left. \frac{\partial f(x_k, \mathbf{a})}{\partial a_i} \right|_{\mathbf{a}=\mathbf{a}^0} \tag{3.15}$$

If the difference $\Delta\mathbf{a} = \mathbf{a}^{n+1} - \mathbf{a}^n$ is below a certain limit, the algorithm can be stopped. After reaching this limit, the normal matrix and the covariance matrix can be determined to calculate the confidence interval resp. the error of the finally calculated parameters.
In the two-dimensional case, the normal matrix can be calculated in the same way as in the linear case (Equ. 3.8), so the equation to determine the normal matrix **N** is identical to equation 3.14.

Convergence

The success of this numerical method depends strongly on the quality of the parameter guess in the first step. If the parameters are chosen badly, the algorithm may take many iterations till it converges or even may diverge and the calculations will be useless. Generally, it is desired to gain a certain stability in the algorithm and some independence from the initial values.

Levenberg-Marquardt

The Levenberg-Marquardt approach is an optimization of the stability and the convergence of the original least squares algorithm. The original problem of solving the matrix equation

$$\mathbf{A} \cdot (\mathbf{a} - \mathbf{a}^0) = \beta \tag{3.16}$$

is modified to the equation:

$$(\mathbf{A} + \lambda \mathbf{D}) \cdot (\mathbf{a} - \mathbf{a}^0) = \beta \tag{3.17}$$

with the diagonal matrix D:

$$D_{ii} = A_{ii} \tag{3.18}$$

The parameter λ can be assumed to be a small value which is increased or decreased after each iteration depending on

$$\chi^2_{t+1} > \chi^2_t \quad \text{or} \quad \chi^2_{t+1} < \chi^2_t$$

to guarantee the convergence of the method. The Levenberg-Marquardt algorithm is part of the Matlab standard library and probably the most frequently used variant of the least squares methods. Therefore, it is also used in this thesis in order to compare and evaluate the results of the developed algorithm. Its performance is evaluated and compared in the results chapter (Chap. 6).

3.1.2 The Method of Maximum-Likelihood [32]

The method of maximum likelihood is a method to estimate unknown parameters of statistical models like the Poisson, the Binomial or the normal distribution. It is often used if it is not possible or unlikely to gain a big amount of data to achieve satisfying statistics.

For a set of measured data x_i the distribution function $f(x_i; \mathbf{a})$ gives the probability for every data point of the series. The so called likelihood function is defined by the product of all probabilities of the measured series.

$$L(\mathbf{a}) = \prod_{i=1}^{N} f(x_i; \mathbf{a}) \tag{3.19}$$

statistical model	function	parameters
Binomial Distribution	$f(n,k) = \binom{n}{k} a^k (1-a)^{n-k}$	a
Poisson Distribution	$f(k) = \frac{a^k}{k!} e^{-k}$	a
Normal Distribution	$f(x) = \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$	σ, μ

Table 3.1: Common distribution functions and their parameters

The likelihood function only depends on the parameters a_j and its value defines the confidence of the choice of the parameters. Consequently, the maximum of the likelihood function defines the parameter values having the highest confidence. The maximum can be determined by setting the first derivative of the likelihood function, with respect to the parameters a_j, to zero. However, the derivative of a product containing many factors is not very pleasant to deal with. To transform the problem in a more convenient form, the logarithm of the likelihood function can be taken. This does not change the position of the maximum but turns the product into a sum:

$$\lambda(\mathbf{a}) = \ln L = \sum_{i=1}^{N} \ln f(x_i; \mathbf{a}) \tag{3.20}$$

In this way, the problem of the parameter estimation of a distribution function containing only one parameter is reduced to solving the likelihood equation $l' = \partial\lambda/\partial a = 0$. In the general case of p parameters the likelihood equation becomes a system of p simultaneous equations:

$$\frac{\partial\lambda}{\partial a_j} = 0 \quad , \quad j = 1, 2, ...p \quad . \tag{3.21}$$

The solution of the system of equations returns a set of parameters values $\tilde{a}_1 ... \tilde{a}_p$. The error of the resulting values is given by the diagonal elements of the covariance matrix. To get the covariance matrix **C** the normal matrix **N** has to be determined and inverted:

$$N_{ij} = \left. \frac{\partial^2\lambda}{\partial a_i \partial a_j} \right|_{\mathbf{a}=\tilde{\mathbf{a}}} \tag{3.22}$$

$$\mathbf{C} = \mathbf{N}^{-1} \quad \Rightarrow \quad \Delta a_j = \sqrt{c_{jj}}$$

In contrary to the least squares method, the maximum likelihood algorithm is a non iterative algorithm to determine parameters of a distribution function. The method provides also the confidence of the estimation after a fixed number of calculation steps.

Estimator for a Two-Dimensional Normal Distribution

The method of maximum likelihood is illustrated by an example that shows how to determine the parameters of a two-dimensional normal distribution. The distribution function is given in equation 3.1 without the background terms. The system of the four simultaneous likelihood equations for the four parameters $\mu_x, \mu_y, \sigma_x, \sigma_y$ results in the four estimators:

$$\mu_x = \frac{1}{N} \sum_{i=1}^{N} x_i \qquad \mu_y = \frac{1}{N} \sum_{i=1}^{N} y_i \tag{3.23}$$

$$\sigma_x = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i-\mu_x)^2} \qquad \sigma_y = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i-\mu_y)^2} \tag{3.24}$$

The error of each value is given by the diagonal elements of the covariance matrix (Equ. 3.9)[1]. They are defined by:

$$\Delta\mu_x = \frac{\sigma_x}{\sqrt{N}} \qquad \Delta\mu_y = \frac{\sigma_y}{\sqrt{N}} \tag{3.25}$$

$$\Delta\sigma_x = \frac{\sigma_x}{\sqrt{2N}} \qquad \Delta\sigma_y = \frac{\sigma_y}{\sqrt{2N}} \tag{3.26}$$

3.2 Hardware Usage in Localization Microscopy

The general approach to execute the calculations to determine the precise location of molecules with sub-pixel accuracy is to use common workstation computers that calculate using fast CPUs. The algorithms that are applied to analyze localization images use mostly iterative least squares fits in order to determine the parameters. This method is used because it provides high precision of the results but also slow calculation speed. There are few approaches to optimize the processing speed of this method. For instance, certain steps in the algorithm can be contributed to several computing cores. In actual workstation computers four to eight computing units can be applied. Especially the parameter calculation of the detected signals can be executed in parallel on multiple cores.

Nevertheless, an implementation with multiple CPUs cannot accelerate the calculations to real-time. Consequently, special hardware usage or alternative algorithms are required for an additional speed up. An FPGA-based implementation that accelerates the data processing of single molecule fluorescence microscopy has not yet been presented. However, a few different solutions have been published.

3.2.1 Non Iterative Localization Microscopy on a CPU

An analytical solution to determine the exact positions of single fluorescing molecules that first was presented in 2007 is the fluoroBancroft algorithm [33]. It is inspired by the Bancroft algorithm for localization in the *global positioning system* (GPS). The idea of this method is to calculate the precise position directly from the measured intensity distribution. Therefore, the original Bancroft algorithm requires i n addition to the measured intensity, the exact position of the measurement. In microscopy these positions are defined by the detector pixels. Finally, the calculations of this analytical algorithm result in one system of linear equations for each signal that can be solved by matrices calculations.

The speed up compared to an iterative least squares fit is pronounced by a factor of 200 by maintaining the localization accuracy [34]. In order to give an idea of the performance of the fluoroBancroft algorithm, a Matlab implementation of the fluoroBancroft is created and the results of an evaluation using simulated data are compared to other methods (Chap. 6).

3.2.2 Localization Microscopy on Graphics Cards

The fastest method in localization microscopy presented so far in 2010 is called MaLiang [35]. The algorithm is based on the method of maximum likelihood and is implemented on a graphics card. The advantage of using a graphics card is the amount of processing units that can

[1]Calculations for the covariance matrix are listed in the Appendix E.

execute many computations in parallel. If the algorithm supports high parallelization, a translation on graphics cards can provide very high processing speed.

An evaluation of this implementation with simulated frames produced a very high calculation speed. The calculation time for 30 simulated frames with a resolution of 512×512 pixels lasts only 0.11 seconds. In every image, 100 molecules have been inserted. In other words, the processing speed is at about 20000 fluorescing molecules per second with a localization accuracy that is comparable to the precision of the calculations of other methods. Furthermore, the authors of this method state that this method reaches already image processing in real-time.

Chapter 4

Development of the Algorithm

This chapter describes the conceptional draft of the implementation. It is explained how it is possible to gain a high resolution localization image out of a stack of noisy images containing single spots overlaid with background. Therefore, the reader has to understand the imaging process and the difficulties in extracting and finding the glowing spots within the images.

The currently available algorithm is based on a least squares fit and runs on a common workstation computer. Its localization accuracy is quite high but it takes very long to analyze the image data. The new algorithm has to be fast enough to realize data processing in a few minutes or less. In order to guarantee high speed data processing highly specialized hardware is applied. The final algorithm shall be implemented on an FPGA-card. So the development of the single modules of the algorithm has always to be considered to be translatable to hardware. However, iterative least squares fits can hardly be translated and realized on FPGAs. Therefore, an approach is presented that is based on the method of maximum likelihood. Further, the single steps of the algorithm are described and the basic ideas for their realization are introduced.

4.1 The Imaging Process

In localization microscopy it is possible to detect single molecules that are optically separated from each other. The basic principle is described in section 2.2.3. The fluorescence of the molecules can be activated manually and the emitted photons can be detected by a camera. According to the glowing intensity of each molecule a number of photons is detected by the camera during the fluorescing phase. In addition to the photons of the desired signals, the camera detects also information from noise and the background that is mainly created by fluorescence from out-of-focus regions or autofluorescence. So the signals have to be separated from the background, before they can be further processed.

Because of photobleaching (Sec. 2.2.3), the background changes during the acquisition of the images. Generally, the intensity of the background follows an exponentially decrease. That means the decrease is very fast within the first few frames and becomes very slow to the end of the acquisition like it is shown in figure 4.1.

Since the activation of the fluorescence is a totally statistical process, it happens that the spots of two molecules overlap. This has also to be considered to obtain a stable algorithm.

In order to create a high resolution image of a biological structure, a large number of signals has to be detected to gain a satisfying structural resolution. Therefore, thousands of images are taken from one structure to gain the locations of hundred thousands of blinking molecules. To finish these calculations within a short time, the algorithm has consequently to be very fast. Furthermore, the generation and visualization of the final high resolution image is another

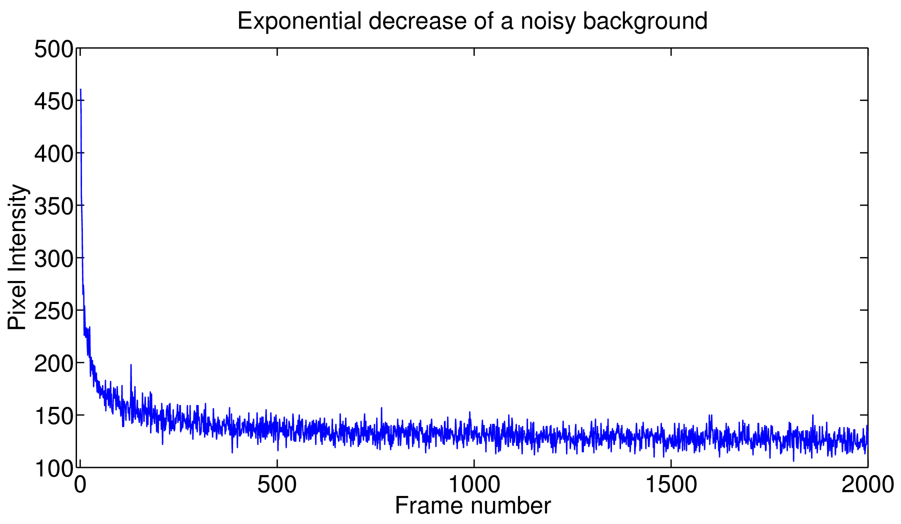


Figure 4.1: Temporal intensity development of the background

exercise that may not take longer than the actual data processing.

4.2 Steps of the Algorithm

After this introduction the tasks of the algorithm can be described as

1. Elimination of the background
2. Detection of the spots and extraction into separate frame sections
3. Separation of close-by spots
4. Calculation of the desired parameters
5. Creation of a super-resolution image from the collected locations

4.2.1 Background and Noise Handling

As mentioned before, the background changes throughout the entire image recording. The method for the background handling has to provide the subtraction of the variable background in each image. Since the background varies only slowly from one image to the next, the information about the background from one image can be reused for the calculations on the background of the following images.

Difference Image

In theory, a glowing molecule is only visible on one frame I_j and remains dark in the next frame I_{j+1}. Since the difference of the background from one image to the next is rather small, the previous frame can be used as a background map for the next:

$$D_j = I_j - I_{j-1} \tag{4.1}$$

This is a simple approach and works quite well for images which contain only a few bright spots. Because the background map I_{j-1} itself contains signals, problems can occur if two signals intersect in two following frames. The close-by signals will loose too much information due to the subtraction process to achieve a satisfying localization accuracy.

In addition to overlapping signals, the noise of the images also increases the uncertainties. The noise can be considered as Poisson noise which means that the error of a measured intensity N is defined by the square root of the measured value:

$$\sigma_N = \sqrt{N} \tag{4.2}$$

Consequently, the error resp. the noise of the intensity value rises by this subtraction to

$$\widetilde{\sigma}_N = \sqrt{N_j + N_{j+1}} \approx \sqrt{2N}. \tag{4.3}$$

At images containing very low noise, this will not have any big effect on the later calculations. Considering higher background levels however, this can have a bad influence on the results of the final parameter estimations.

Mean Value

The first idea to realize the subtraction of the background without increasing the noise is to use an average value for each image:

$$D_j = I_j - \langle I_j \rangle \tag{4.4}$$

If the background is homogeneous enough, this is a good approach to separate the signals from the background without increasing the noise. Unfortunately, in the general case the background is not homogeneous at all, so this method will work only on very special images.

Moving Average Background Map

An approach that adopts most of the advantages of the methods mentioned above is a background map which only contains background information with no signals and as less noise as possible. Additionally, the background map has to change with every image in order to adopt variations in the background. This is all provided by a moving average background map:

$$BM(x,y)_j = BM(x,y)_{j-1} + \frac{1}{BGF}\left[I(x,y)_j - BM(x,y)_{j-1}\right]. \tag{4.5}$$

The map stores the information of the background for each pixel, averaged over the last few frames. The value of the averaging factor BGF determines the speed of the adaption of the background. An increase of the factor means a decrease of adaption speed. In contrary, the bigger the BGF the more reduction of the noise can be achieved:

$$\sigma_N = \sqrt{N/BGF} \tag{4.6}$$

Because it is known that the background changes slowly and decreases primarily over the image recording, it is possible to distinguish signal peaks from background information. Consequently, it can be assumed that the background does not rise within two images more than the fluctuations of the noise. If it does, it must be a signal of a molecule and its contribution to the background can be limited:

$$BM(x,y)_j = BM(x,y)_{j-1} + \frac{1}{BGF}\left[\min\left(I(x,y)_j - BM(x,y)_{j-1}\,,\ \sqrt{I(x,y)_j}\right)\right] \tag{4.7}$$

The first background map can either be simply the first image or a mean image of the first n frames

$$BM(x,y)_1 = I_1 \qquad or \qquad BM(x,y)_1 = \langle I_1, ..., I_n \rangle \quad e.g.\ n = BGF. \tag{4.8}$$

After the background map *BM* is subtracted from the image *I* a difference image *D* is generated that contains, in the best case only signals of the glowing molecules. In order to avoid negative values and for further noise reduction, all negative values of the difference image are set to zero.

$$D_j = \max(I_j - BM_{j-1}\,,\,0) \tag{4.9}$$

Noise Reduction

Every measurement contains errors. Especially in statistical processes fluctuations in the measured values can never be avoided. For that reason, there are several methods to lower the influence of noise to the result. Normally, it is possible to lower the relative error using a bigger sample size. However, this cannot always be realized.

In image processing there is another approach. As described in section 2.1 an optical system can be described as a frequency filter. Consequently, it can be assumed that the image contains only frequency information of the sample up to a certain maximal frequency. Therefore, frequencies beyond that limit belong to noise and can be erased. That means, the noise can be decreased by applying a low pass filter on the image. However, a frequency filter is a very computation intensive process, because the whole image has to be analyzed by a Fourier transform, the required frequencies have to be separated and finally the whole thing has to be transformed back into the local space.

Besides the low pass filter, a simple subtraction of a constant value to every pixel in the image can also reduce noise.

Because the background map cannot eliminate the whole noise, the difference image still contains fluctuations. These small fluctuations around the each signals can be erased by the so called `cutoff` which is a mean value of about of the size of two times the noise:

$$cutoff_j = \sqrt{\langle I_j \rangle} \times 2 \tag{4.10}$$

4.2.2 Detection of the Signals

After elimination of the background, the spots have to be found and cut into smaller sections, the so called *regions of interest* (ROIs). A ROI is a small frame section that contains a spot with the local spot maximum centered in its middle.

Due to the fluctuations of the noise in the signal, it occurs that the center of the spot is not defined by its highest pixel value. If the spot is not centered correctly in the ROI, later calculations will be falsified. For this reason it is very important to find the real mid-point of the signal. Therefore, a special filter f can be applied on the difference image D. This filter is a two dimensional *finite-impulse-response-filter* (FIR-filter) with an edge size of 3 pixels. The FIR-filter calculates for each pixel the mean value of the pixel and its surrounding pixels (Equ. 4.11).

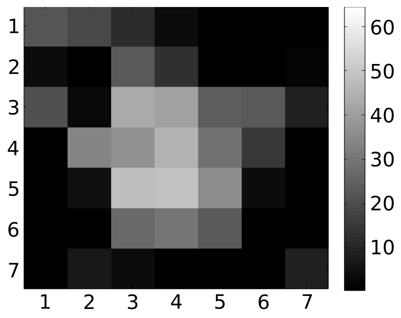


Figure 4.2: An extracted signal centered in a ROI

$$\widetilde{D}(x_0,y_0) = f\left(D(x_{-1},y_{-1}),\ldots,D(x_1,y_1)\right) = \frac{1}{9}\sum_{i,j} D(x_i,y_j) \qquad i,j = -1,\,0,\,+1 \tag{4.11}$$

After this operation, the local maxima in the filtered image define the locations of the real center of the spots.

However, not every local maximum can be assumed to be really a signal. Values below a certain limit can be disregarded and treated as noise.

The strength of the fluctuations is known and can be assumed to be of the size of the square root of the image value $I(x,y)$. So a factor ϑ can be introduced that defines the threshold for detecting a spot to be a signal of a glowing molecule.

$$Threshold = \vartheta \sqrt{I(x,y)} \tag{4.12}$$

Since the `cutoff` subtracts 2 times $\sqrt{\langle I_j \rangle}$ the threshold factor should be significantly larger than two. A value for the threshold factor ϑ that works very well to suppress most of the fake signals is

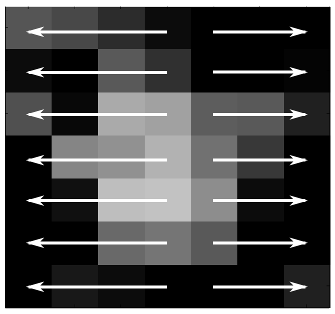
$$\vartheta = 4. \tag{4.13}$$

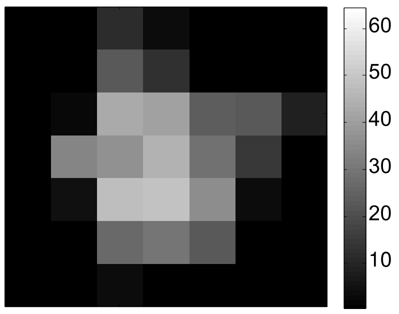
After a spot is detected, the whole signal has to be extracted into a ROI. The dimension of the ROI has to be of a size which makes the whole signal fit into the ROI area. However, if the size is considered too large, the edges of the ROI contain too much noise, what should be avoided. A measurement of a typical spot size is presented in section 6.3.1. Due to this measurement the size of a ROI is specified to 7 pixels in each direction.

4.2.3 Separation of Overlapping Signals

The concentration of the signals in an image often varies within a frame. In every image there are areas that remain almost dark and others that show high fluorescence activity. For instance, molecules can form clusters with thousands of fluorophores within a small area. Therefore, it can happen that spots are not completely optically separated anymore. Nevertheless, up to a certain distance spots can manually be separated.

Normally, the maximum value of a ROI is centered in the middle of the ROI and the pixel values decrease from the middle to the edge. If pixel values rise again towards the edge it can be assumed that an intersection with another spot occurred. To avoid an influence of the second spot on the later calculations, the rising values after the local minimum are set to zero towards the edges.


(a) A second signal overlaps the ROI from top left



(b) After separation

Figure 4.3: Separation of overlapping spots

The operation shown in figure 4.3 is to separate signals in the x-direction. It is repeated in y-direction to provide better results. As it can be seen in figure 4.3(b) the cleared pixels do probably not correspond to the original signal. In addition to decreased noise, the final shape of the spot is rounder than it has been before.

4.2.4 Estimation of the Parameters

Now the desired parameters can be extracted from the adjusted ROI. But what parameters are needed?

The detected signals can be approximated by the two-dimensional normal distribution (Equ. 3.1). This function contains several parameters that have to be determined in order to fit this function to the pixel values. The fitting method has to provide these parameters and additionally the error of the resulting values. As mentioned before an iterative least squares fit can hardly be translated to a hardware pipeline. So the method of maximum likelihood becomes the applied method (Equ. 3.23 and 3.24) that is also able to determine the parameters of a distribution function including the errors.

Beside the parameters of the distribution function μ_x, μ_y and σ_x, σ_y further values can be extracted from a ROI:

The maximum intensity value of the ROI: *Imax*

The cumulative intensity of the ROI: Q

The number of the image containing the ROI: *imgNR*

The error of the mean values: $\Delta\mu_x$, $\Delta\mu_y$

The error given by equation 3.26 defines the uncertainty of the parameter values estimated from the values of the ROI. However, the preprocessing of the ROI that is described above creates further deviations. Further, the acquisition of every single photon in the camera contains errors.

This means that the error formula has to be extended by a term that includes the error of the background and an additional term which describes the pixelation error that defines the uncertainty of the position of each photon within the pixel.

The pixelation error is defined by $\Delta x^2 = 1/12$ pixel. The error added by the background can be described by Poisson noise:

$$\Delta N_B = \sigma_B = \sqrt{N_B}. \tag{4.14}$$

The final error formula can be determined by common error propagation.

Since the error of the position defines the localization accuracy of the spot, it is a very important size for evaluating the algorithm and for the later analysis of the image.

In summary the formulas for all parameters are listed below.

$$\mu_x = \frac{1}{Q}\sum_{j=1}^{Q} x_j \qquad \mu_y = \frac{1}{Q}\sum_{j=1}^{Q} y_j \tag{4.15}$$

$$\sigma_x = \frac{1}{Q}\sum_{j=1}^{Q} (x_j - \mu_x)^2 \qquad \sigma_y = \frac{1}{Q}\sum_{j=1}^{Q} (y_j - \mu_y)^2 \tag{4.16}$$

$$\Delta\mu_x = \sqrt{\sum_{j=1}^{Q}\left(\frac{\partial\mu}{\partial x_j}\Delta x_j\right)^2 + \sum_{j=1}^{Q}\left(\frac{\partial\mu}{\partial q_j}\Delta q_j\right)^2} \tag{4.17}$$

$$= \sqrt{\frac{1}{12Q} + \sum_{i=1}^{N}\left(\frac{x_i-\mu_x}{Q}\right)^2 (q_i + N_B)} \tag{4.18}$$

The index j runs over each photon of the ROI. In formula 4.18, the index i runs over each pixel of the ROI.

A closer look to the formulas reveals the big advantage of the algorithm compared to a least squares fit: the final value of each parameter is received by one single run over each data point of the ROI, which makes the algorithm very fast. The algorithm is also suitable for hardware usage because it consists of a fixed number of calculation steps.

The whole algorithm described so far will further be named as *Estimator algorithm* or simply as *Estimator*.

To gain an idea of the performance of the algorithm, a comprehensive simulation of the whole algorithm is designed. The whole simulation is described in the implementation chapter (Sec. 5.1.1). The results of the simulation, including the comparison with other methods, are presented in the results chapter (Chap. 6).

4.2.5 Display the Positions in a High Resolution Image

The algorithm returns a data set that contains amongst other values the precise positions of the molecules. In order to create an image from these positions, they can simply be highlighted in a two-dimensional map. However, the examination of biological structures can be difficult if only single dots are visible in an image. For the human eye, it is more natural to see a structure as a plain and connected area. This is why several methods are introduced to generate such a plane structure from single dots. What these methods all have to provide is the highlighting of areas with a high dot density. Other areas containing few spots should remain dark.

Furthermore, the original signal intensity or the localization accuracy can be used to describe the brightness of a spot in the image.

Gaussian Blurring

A signal is described by its position the error of the position its width and the detected intensity. These values are not completely independent. So a spot having a very high intensity also has a high localization accuracy. If a spot is described by a normal distribution that has a width of the localization error, a more laminar illustration of the spots can be achieved. Also the brightness of a normal distribution in the image depends on its width. If it is thinner, it becomes brighter.

This approach illustrates basically the localization accuracy of the fitting algorithm. The Gaussian blurring method provides fast generation of high resolution images. However, if only few spots having high localization accuracy are detected, the blurred dots do not intersect and the image still consists of dots.

Nearest Neighbor

This is another method that works quite similar to the Gaussian blurring method. Instead of using the localization error, the distance to the nearest spot is used as width of the normal distribution that is inserted to display a signal. This idea ensures that the spots in the image always intersect. By using the distance to the next spot instead of the position error the image gets brighter in areas where a lot of spots have been detected, and remains dark where only few spots have been found. In other words, the idea of the Nearest Neighbor method is to realize a more realistic illustration of the structural resolution. However, the calculation afford rises dramatically with the amount of detected molecules [$O(n^2)$], because the nearest neighbor of each detected signal has to be found within all detected locations.

Triangulation

Another method that has been developed by David Baddeley[36], is based on the triangulation algorithm of Boris Delauney presented in 1934[37]. The final image of this proposal only consists of plain triangles. The edges of each triangle lay on the position of a spot and its two nearest neighbors. Each triangle has a brightness value that is proportional to its area. In order to hide the edges of the triangles to create a smoother illustration, further iterations can be applied to improve the quality of the final image. In each iteration the spots have to be shifted within their localization accuracy to displace the triangles. The triangle structure vanishes after a few iterations.

4.3 Development of the Hardware Algorithm

In this section the conceptional draft of the hardware version of the Estimator algorithm for the FPGA card is presented. To reach the goal of real-time data processing this specialized hardware is to be applied. The general translation of the basic features of the algorithm is very straightforward. However, in hardware there are different restrictions that have to be considered.

In the following pages these restrictions and additionally difficulties will be discussed and the final choice of using an FPGA will be further motivated. Also, some considerations for the commercial environment VisualApplets which is used for the implementation are introduced.

4.3.1 Choosing the Hardware Platform

The way to speed up an algorithm by application of special hardware is either to increase the calculation speed or to do more calculations simultaneously. The additional calculation speed of one single high-end CPU will not have a very big effect on the execution time of the available least squares algorithms. On a workstation computer with multiple cores, certain tasks can be distributed on the single processing units (CPUs). However, it cannot be expected that the speedup of a multi-threaded solution will reach the requirements. The most promising approaches are the implementation on a graphics card or an FPGA.

The heart of a graphics card is the *graphics processing unit* (GPU). It provides a very high number of simple but fast processing units supported by very fast memory. It is able to do very much calculations in parallel. The limiting factor of the usage of graphics cards is mostly the data transfer between the CPU of the hosting workstation and the graphics card. The big advantage of a GPU-implementation is that commercial hardware is available and the implementation is limited only on programming the code for the algorithm.

An FPGA provides flexible hardware resources that can be interconnected to describe any algorithm (Sec. 2.3). The clock frequency of an FPGA is slower than the frequency of a high-performance CPU, but the advantage of an FPGA-implementation is that the logical units of an FPGA allow perfect customization of the circuits to the required algorithm.

This can be used by translating the single steps of the algorithm into a very long pipeline. This pipeline can be described in a way that with every clock cycle one pixel of the image enters the FPGA and propagates cycle by cycle one calculation step further. Of course this means a certain latency until the first result can be stored, but then the results arrive at every clock because every step of the pipeline computes simultaneously. In this way the pipeline defines a high degree of parallelization.

The problem of an FPGA-based implementation is that commercial FPGA-environments are less available than graphics cards. Because it is not intended to create an own FPGA-board, a commercial system has to be found that fits the requirements.

Additionally, an implementation of an algorithm is much more complicated than an implementation in software because there are much more constrains that have to be considered.

Nevertheless, the final choice came down to an FPGA-based solution. Not last, because a cooperation between the University of Heidelberg and Silicon Software, a company from Mannheim, was created to investigate new fields of application for their FPGA-driven frame grabber cards, and to test if these cards are able to do such calculations. Another big advantage that is provided by the cooperation with Silicon Software is the ability to use their graphical tool VisualApplets for implementing the algorithm in hardware (Sec. 2.4.1).

Since the products of Silicon Software are mainly designed for image processing, VisualApplets is generally used for developing frame based algorithms in a long pipeline. Therefore, the tool is naturally capable to translate the Estimator algorithm onto the FPGA. Additionally, the frame grabber card provides a fast solution for the data transmission between the card and the hosting workstation PC.

In this thesis a comprehensive investigation of an FPGA-based acceleration of the processing of localization microscopy data is described. However an additional implementation of e.g another algorithm on a graphics card would also be worth to be taken into consideration.

4.3.2 The Algorithm in Hardware

The implementation of a mathematical algorithm on an FPGA underlies several restrictions that differ from an implementation in software. Especially in the graphical development tool from Silicon Software there are additional constraints. Since VisualApplets is designed for image processing, it is optimized for calculations with two-dimensional frames of integers having a fixed bit-width. Silicon Software's VisualApplets does not support floating point calculations. Additionally, in VisualApplets it is basically not allowed to create loops within the data flow graph. These design rules have to be regarded when implementing an algorithm in VisualApplets.

In order to get the best results from the given tools some considerations have to be made before implementing the algorithm. These considerations are presented in the following.

The images are sent to the frame grabber pixel by pixel. Since the dimensions of the images are passed to the frame grabber, the hardware is able to recognize whenever a new image arrives and marks the start and the end of every image. So it is possible to describe the image processing frame by frame and not pixel based. Beside the computations on the whole image, the algorithm calculates several parameters on only a small section of an image that contains only one single spot. Therefore, the small section has to be extracted from the rest of the image. So the original data stream in the FPGA is split up into a second data stream that contains only the extracted ROIs. In order to calculate the final position of each signal, and to minimize the data traffic between the FPGA-card and the hosting system, these two streams have to be synchronized before the results are transmitted.

The calculations on the first data stream contain the background handling, the spot detection and the separation of the spots into ROIs. After the separation of the ROIs, the further needed data of this part are only the global coordinates and additional resynchronization information. This data is needed to calculate the global position of the spot and to check if the resynchronization worked fine. In order to extract the ROIs, special memory operators can be used.

The second section of the hardware algorithm is to implement all calculations on the ROIs, including the execution of the cutoff and the separation of overlapping spots (as described in section 4.2.1 and 4.2.3). At the output of the operator, which extracts the ROIs from the image stream, a new data stream of the dimensions of the ROIs appears. The combination of values of both streams can only be done after a special synchronization.

Frame-Based Data-Flow

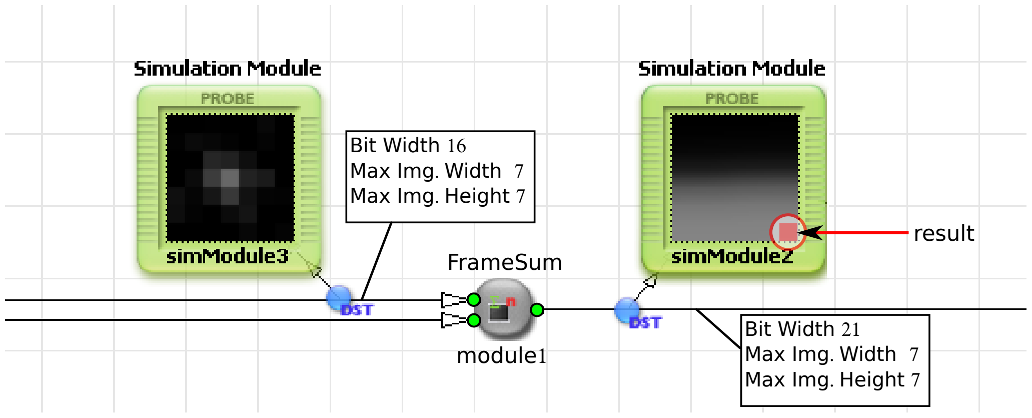


Figure 4.4: Frame-based calculation: operator `FrameSum`. This operator is used to calculate the total intensity of one signal.

The data flow in Silicon Softwares FPGA-card is frame-based. That means that each operator inserted in the data flow graph in VisualApplets does its computations on the whole image frame. For instance, the `FrameSum`-operator calculates the sum of all pixel values of one frame, pixel by pixel (Fig. 4.4). The output of this operator is also a frame having the same dimensions as the input frame. The pixel values of the output frame contain the accumulated pixel values of all previous pixels. So only the very last pixel of the output frame contains the total sum over all pixels of one frame. All other pixels contain no useful information.

This means for the determination of the parameters $(\mu_x, \mu_y \ldots)$ every calculation results in such a frame. Since only the last pixel value is required, these values have to be extracted from each frame and have to be summarized in one final result-frame which contains all parameter values that are extracted from one ROI.

Fix Point Accuracy in Hardware

In order to achieve sub-pixel accuracy, the integer values have to be changed to fix-point values with several decimal digits. Since the images contain exclusively 16-bit integers it is preferred to provide also 16-bit pixel values at the output.

Further, the image data from the camera contain only 12-bit gray values. So the upper 4 bits remain unused. In order to increase the precision of the background map (Sec. 4.2.1) three of these additional bits can be used as fix-point digits. This can simply be realized by a 3-bit right-shift (the fourth bit is required as sign bit). The same idea can be applied to the computations of the parameters. The final implementation is described in section 5.2.3.

Chapter 5

Implementation

This chapter summarizes most of the work of the thesis. It describes how the single tasks of the Estimator algorithm are finally realized.

The first step was to develop and test the algorithm in software. Therefore, the software platform Matlab was used. Using Matlab the single modules can easily be programmed and tested. Furthermore, the program provides a lot of built-in functions that make it easy to try out new ideas. Another big advantage of Matlab is that results can be displayed graphically. This can strongly simplify the analysis of results.

After the algorithm was determined, the implementation of the algorithm in a hardware pipeline was started. The development of the hardware implementation was made using the program VisualApplets, a graphical hardware development tool by Silicon Software that allows the development of algorithms for their frame grabber cards.

To control the final FPGA-environment and the in and output of the data, a special software has to be developed. This software is described in the last part of this chapter.

5.0.3 General Aspects of Matlab

Matlab is a program that is used basically for numerical calculations. Especially matrices can be handled very easy and matrix calculations can be done very fast. Therefore, Matlab provides several specific operators. For instance, the addition of two $n \times m$ matrices **A** and **B** can be written like

```
RES = A + B ;
```

Since these two matrices are two-dimensional arrays, the addition can also be done using two nested for-loops:

```
for x = 1 : n
  for y = 1 : m
    RES(y,x) = A(y,x) + B(y,x);
  end
end
```

An important thing of the Matlab environment is that the internal operators will do the computations much more efficiently than any nested-loop construction. This is why it is highly recommended to use Matlab operators wherever possible.

5.1 Matlab Implementation

The main goal of the new algorithm is to speed up the image processing. So in each step of the implementation the performance has to be considered.

The code has an hierarchical design of three levels. An overview of the whole Estimator algorithm is displayed in figure 5.1.

The highest level contains the calculations that are applied on the data stack. The main function is called fastSPDM. It includes the generation of the first background map as well as the computations for the background handling and the computation of the difference image.

The difference image is passed to the next lower level that is described by the function (clusterfind). The second level searches each image for molecule signals and determines the coordinates to extract the ROIs. If a ROI is selected it is passed to the lowest level where all computations on the ROI are done.

These computations contain the separation of overlapping signals and the estimation of the precise position and other parameter values. This is done by the functions cl_separator and estimator. Also the cutoff is applied. After all signals are extracted from the image stack, the raw positions of the molecules are further processed and improved by the multiframepoints algorithm. The last step is the generation of the localization image from all the locations of the detected molecules.

The following chapter describes the features of each single function and their implementations.

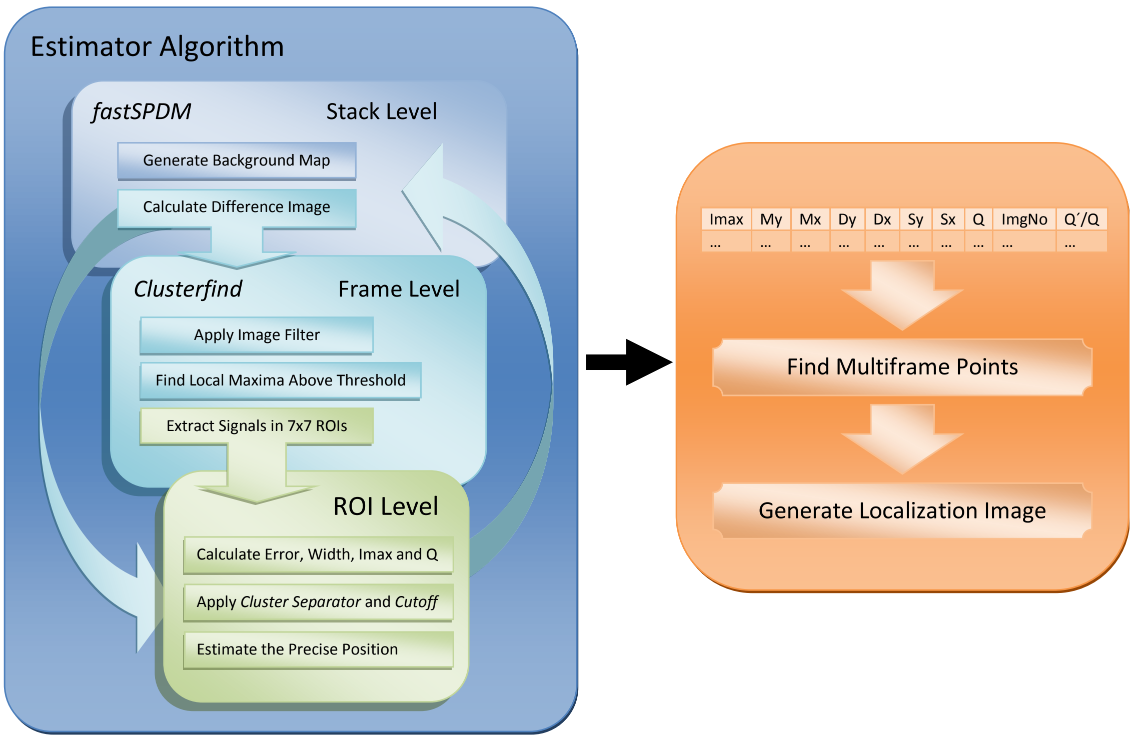

Estimator Algorithm
fastSPDM
Stack Level
Generate Background Map
Calculate Difference Image
Clusterfind
Frame Level
Apply Image Filter
Find Local Maxima Above Threshold
Extract Signals in 7x7 ROIs
ROI Level
Calculate Error, Width, Imax and Q
Apply Cluster Separator and Cutoff
Estimate the Precise Position
Imax My Mx Dy Dx Sy Sx Q ImgNo Q´/Q
Find Multiframe Points
Generate Localization Image

Figure 5.1: Scheme of the algorithm with three levels of processing.

fastSPDM

This is the top level of the algorithm where all the work on the data stack, especially the background handling, is done.

Background Subtraction

The first background map BM_1 is defined by the mean value of the first BGF images. The factor BGF describes the strength of signal and noise suppression of the background map. Additionally, it defines how fast the background map can adopt changes in the real background. The smaller the value is, the faster changes the background map. However, if the factor BGF is set to low, the noise of the background map increases.

Since the algorithm has to be translated to hardware the BGF was defined to 8. Because a division of a number by 8 can be realized in hardware by a simple right-shift by 3.

$$BGF = 8$$

After the first background map is created the function `addimagetobackground` calculates the difference image and the moving average background map for every remaining image of the stack. This is done as it is described in section 4.2.1.

As it can be seen in listing 5.1, the limit for the increase of the background map is defined as the square root of the mean value of the image, not as the square root of each pixel of the image. This constraint was made because of performance reasons since applying a square root is a calculation-intensive operation. Additionally, all negative values of the difference image are set to zero.

This function shows the style of calculations using arrays in Matlab very well: the variables `oldBM`, `img`, `diffimg` and `newBM` are each two-dimensional $n \times m$ arrays. Although these are four operations in which each pixel of the image has to be touched, this function can be described by only five lines without using any for-loop:

```
function
  [diffimg,meanbg,newBM]=addimagetobackground(oldBM,img,BGF)

  meanbg  = round(mean(mean(img)));
  cutoff  = sqrt(meanbg);

  diffimg = img - oldBM;

  newBM   = oldBM + min(diffimg, cutoff)./BGF;

  diffimg = max(diffimg,0);
end
```

Listing 5.1: Matlab code for the background handling function:

The same result can be achieved by running over the variables in a nested for-loop [1] and

[1]A nested for-loop implementation of this is shown in appendix A.1.

doing the calculations pixel by pixel. However, this implementation will never be as fast as the code listed above. An analysis of this is presented shortly in section 6.2.5.

The function `addimagetobackground` returns the difference image and the background map for the following image. Also, the value `meanbg` is returned. This describes the mean value of all pixels of one image and is used for the later thresholding. The difference image contains the spots and a small rest of noise. It is passed to the function `clusterfind` that defines the next lower level of the algorithm.

Clusterfind

The function `clusterfind` defines the second level of the algorithm. It searches for signals in an image and extracts the detected signals into the ROIs.

In the **first** implementation the spots were searched within a for-loop that runs over each pixel and tests if its value is above a certain limit. The threshold is defined as a multiple ϑ of the square root of the variable `meanbg` $= \langle I \rangle$ that is introduced in listing 5.1:

$$\text{Threshold} = \vartheta \sqrt{\text{meanbg}} \tag{5.1}$$

The factor ϑ is called *thresholdfactor*.

A pixel value that is higher than the threshold may belong to a signal of a molecule. In order to center the signal within a ROI, the coordinates of the maximum intensity of the spot have to be found. Therefore, each pixel having a value above the threshold has to be tested whether it describes a local maximum. But due to noise it often occurs that the brightest pixel is located next to the real center of the spot. Thus, the signal cannot be centered correctly in the ROI what reduces the accuracy of the subsequent calculations. Furthermore, the finding of the spots by running in a for-loop over each pixel is very inefficient in Matlab.

So a **second** approach was developed:
Since the signal center is defined by the area that contains the highest intensity values, the problem of finding the real center of the signal is resolved by applying a two-dimensional FIR-filter on the difference image as described in equation 4.11. Therefore, the built-in Matlab function `imfilter()` can be used to gain maximal calculation speed. The code of the filtering process is listed in listing 5.2. Now the molecule signals can be searched on the filtered image.

For performance reasons, the for-loop that runs over the entire image is replaced by another built-in Matlab function `find()` It is used to find the pixel values in the filtered image having an intensity value above the threshold. But since the image filter lowers the maximum values of the spots the threshold has to be adapted:

$$\widetilde{\vartheta} = \vartheta - 1. \tag{5.2}$$

Now each position having an intensity above the threshold is stored in `ypos,xpos` and is tested if it is a local maximum. The coordinates of the local maxima define the real position of the center of the spots in the difference image. Using these coordinates the spots can be cut in the ROIs. The further calculations are done only on the small arrays.

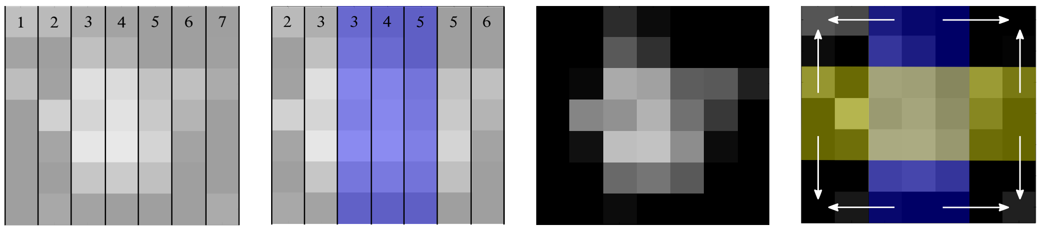
The size of the ROIs was specified to seven pixels in each direction. It turned out that real molecule signals fit completely into that size.

```
Threshold = (thresholdfactor -1)*sqrt(meanbg);

stencil = ones(3,3)./9;

firimg = imfilter(diff,stencil);

[ypos,xpos] = find(firimg>Threshold);
```

Listing 5.2: Matlab code for the image filter and finding of the positions of possible signals:

Cluster Separator and Estimator

Before the final parameters can be estimated, the ROI has to be searched for overlapping signals. Therefore, it is passed to the `cluster separator`.



(a) Original ROI with rows 1 to 7 in x-direction (b) Comparision frame for x-direction. Blue=fix (c) Resulting frame without intersecting signals. (d) Two directions of the separator. Fix areas marked.

Figure 5.2: Cluster Separator: The original frame 5.2(a) is tested for overlapping spots at the edges. Pixel values greater than in the comparision frame 5.2(b) are set to zero. The shape of the comparision frame defines the changed and the fixed rows (3,4,5). This is also applied in y-direction.

The `cluster separator` tests the values of the ROI if they decrease from the center to the edge. The values are tested in x- and in y-direction row by row. In order to conserve the center of the spot, only the outermost rows are checked and changed. So the colored regions in figure 5.2(d) stay unchanged during the separation in x-, resp. in y-direction. Using Matlab, this operation can be written in a very short way that is shown in listing 5.3 and 5.4:

Listing 5.3: Separation in x-direction:

```
xframe = [roi(:,2:3),roi(:,3:5),roi(:,5:6)];
roi = roi.*(xframe+sqrt(roi) > roi);
```

Listing 5.4: Separation in y-direction:

```
yframe = [roi(2:3,:);roi(3:5,:);roi(5:6,:)];
roi = roi.*(yframe+sqrt(roi) > roi);
```

To achieve a more conservatively cutting of the spots, small fluctuations of $+\sqrt{q_i}$ within neighbored rows are allowed. If the `cluster separator` cuts too much signal the two spots are intersecting too strong and cannot be separated anymore. If this occurs the spot cannot taken into consideration and has to be neglected. The practice has shown that the limit for neglecting a spot should be at about 30% loss of the total intensity after separation:

$$\frac{Q}{Q_{old}} > 0.7 \tag{5.3}$$

Cutoff

Signals in the corners of the ROI depend mostly to noise but have a very strong contribution to the result of the later calculations. Also, the original shape of the signal is round and the quadratic shape of the ROI results in a shift of the final result. So a further step that also reduces the noise is required before the parameters can be determined.

The additional noise reduction is achieved by subtraction of a constant value from each pixel in the ROI, like it is introduced in section 4.2.1. Negative values are not allowed and set to zero. The value of the `cutoff` has to be of the strength of the noise in the ROI which depends on the noise of the background:

$$Cutoff = \sqrt{meanbg} \times 2 \tag{5.4}$$

Estimator

The maximum-likelihood Estimator algorithm implements the equations described in section 4.2.4. The big advantage of this method is that each parameter of the distribution function can be calculated by an accumulator that runs only once over the entire ROI. A closer look to the equations reveals that the accumulators for $\sigma_{x,y}$ require the result of the accumulators for $\mu_{x,y}$. However, this dependency can be resolved by a simple transformation of equation 4.15 and 4.16:

$$\mu_x = \frac{1}{Q}\sum_{j=1}^{Q} x_j = \sum_{i=1}^{N} \frac{q_i x_i}{Q} \tag{5.5}$$

$$\begin{aligned}\sigma_x &= \frac{1}{Q}\sum_{j=1}^{Q}(x_j-\mu_x)^2 = \frac{1}{Q}\sum_{i=1}^{N} q_i(x_j-\mu_x)^2 \\ &= \frac{1}{Q}\sum_{i=1}^{N} q_i x_i^2 - 2\mu_x \sum_{i=1}^{N}\frac{q_i x_i}{Q} + \frac{\mu_x^2}{Q}\sum_{i=1}^{N} q_i \\ &= \frac{1}{Q}\sum_{i=1}^{N} q_i x_i^2 - \mu_x \end{aligned} \tag{5.6}$$

Since each intensity count of a pixel contributes the same value to the calculation, all counts of one pixel i can be summarized to q_i. So the index j is changed to i which runs over entire pixels in the ROI to save calculation steps. Afterwards, the equations for $\mu_{x,y}$ and $\sigma_{x,y}$ consist only of independent accumulators and both parameters can be estimated simultaneously (Equ. 5.5 and 5.6).

A similar transformation can be applied on the error formula (Equ. 4.18). The transformed error formula then looks like:

$$\Delta\mu_x^2 = \frac{\sigma_x^2}{Q} + \frac{1}{12Q} + \frac{N_B}{Q}\left(\sum_{i=1}^{N} x_i^2 - \mu_x(2\sum_{i=1}^{N} x_i - N_{\oslash}\mu_x)\right) \tag{5.7}$$

containing $N_{\oslash}$ that describes the number of non-zero pixels in the ROI.
The resulting error formula is composed of the calculated width of the spot, divided by Q, the pixelation error and three independent accumulators that describe the uncertainty added by the background.

After these transformation each parameter can be determined by accumulators that require only one single run over each pixel value of the ROI. This fact will later be very important for the translation of the algorithm onto the hardware resources of the FPGA.

The `cluster separator` as well as the `cutoff` removes signal from the ROI what improves the shape of the included spot. However, this has also influence on the calculations of the localization error $\Delta\mu$ and the spot-width σ. For this reason, the resulting value is smaller than expected. In order to fix this, the calculations of these values have to be done before the two operations `cutoff` and `cluster separator` are applied.

All parameters are stored in a big matrix, further called Orte-matrix, that has following shape, displayed in table 5.1

Imax	μ_y	μ_x	$\Delta\mu_y$	$\Delta\mu_x$	σ_y	σ_x	Q	*ImgNR*	Q/Q_{old}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Table 5.1: Orte-matrix as returned by the fastSPDM algorithm

The results of a detailed performance measurement containing the localization accuracy and the computation time of the Matlab implementation is described in the result chapter (Chap. 6).

5.1.1 Setup of the Simulation

The algorithm is a new approach in localization microscopy. To gain an idea about its localization accuracy the algorithm has to be tested using simulated data. In order to be able to transfer the results of the simulation to real data, the simulated spots have to remain as close as possible at the real signals.

The simulation consists of three steps:
The first step is to generate an image that contains one spot. Then the algorithm has to find the spot within the image and to calculate its position, its width and the error. In the last step the result of the computation is compared to the real values.
To evaluate the accuracy, the squared mean value of the difference between the estimated value and the real value is regarded:

$$\Delta loc = \sqrt{\sum_{i}^{N} \frac{(\mu_i^{est} - \mu_i^{real})^2}{N}} \tag{5.8}$$

Generation of Signals for the Simulation

The simulation has to include noise and background like it is found in real images. Since the noise rises with the intensity of the background the simulation has to apply background to the images to create realistic results. So the image is filled with a constant value and the spots are added on top. Then, the Poisson noise is applied on each pixel described by a standard deviation of $\sqrt{N_j}$. The signals are shaped as a Gaussian distribution (3.1) having a certain width σ and intensity Q . The width of the generated signals has to correspond to the width of the real signals in real images. A measurement that determines the value of the width of real signals is presented in section 6.3.1 and returns:

$$\sigma = 1.4 \tag{5.9}$$

To get an idea of the performance of the algorithm, spots having different *signal to noise ratios* (SNRs) have to be evaluated. The SNR is defined by the intensity of the spot compared to the background level:

$$\mathbf{SNR} = \frac{Imax}{\sqrt{Imax + N_B}} \tag{5.10}$$

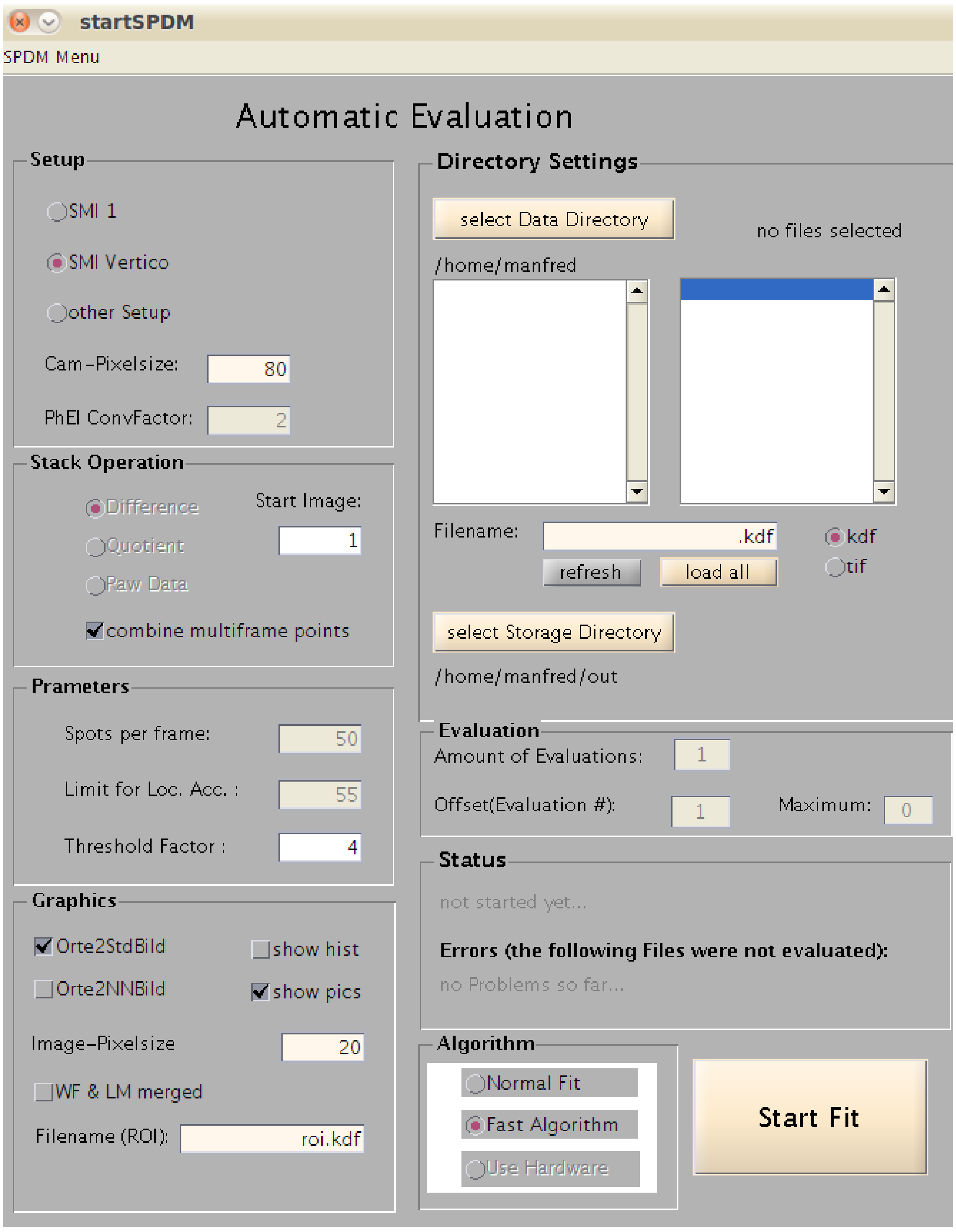
With the maximum intensity of the spot $Imax = Q \ / \ 2\pi\sigma^2$ and the background level N_B. The implemented simulation contains various SNRs between $SNR \approx 5$ and $SNR \approx 18$. To gain significant statistics, 4000 spots for each SNR are evaluated by the Estimator algorithm and the other compared methods. The results of the simulation are presented in the following result chapter.

5.1.2 The Graphical User Interface

If a program is to be used, it is not very convenient if variables must be changed in the code to adapt the program to suit the respective needs. A good program should be self-explanatory and easy to adapt with just a few steps. So the most comfortable way to make a program usable is to show each important information in a so called *graphical user interface* (GUI), where all adaptations can be done.
Matlab provides a built-in GUI generation tool called GUIDE. Using GUIDE one can comfortably create basic GUIs. The GUI for `fastSPDM` is called `startSPDM` (Fig. 5.3) and includes the adaptations for all parameters that are required for the Estimator algorithm. Also the old algorithm is still available.

Especially the selection of the data files is very easy: this can be done by a common file-browser. Even multiple files and different file types can be selected for evaluation. Allowed file types are **tif** and **kdf**. Further parameters of the GUI are:

- The **Pixelsize** in nanometers of the detector pixels in the recorded images
- **Start Image** defines the number of the first frame of the image stack that contains signals
- The user can chose if signals that were detected in **multiple** following **frames** should be **combined** to one signal having a higher accuracy.
- **Threshold-factor** defines the minimum intensity of a spot over the background. It is multiplied by $\sqrt{meanbg}$
- **Image-Pixelsize** in nanometers. It defines the size and also the resolution of the later high resolution image
- The user can chose if images are created and displayed on the screen using checkboxes.



startSPDM

SPDM Menu

Automatic Evaluation

Setup

SMI 1 [✓ SMI Vertico] other Setup

Cam-Pixelsize: 80

PhEl ConvFactor: 2

Stack Operation

[✓ Difference] Quotient Raw Data

Start Image: 1

[✓ combine multiframe points]

Prameters

Spots per frame: 50

Limit for Loc. Acc. : 55

Threshold Factor : 4

Graphics

[✓ Orte2StdBild] Orte2NNBild

show hist [✓ show pics]

Image-Pixelsize 20

WF & LM merged

Filename (ROI): roi.kdf

Directory Settings

select Data Directory

no files selected

/home/manfred

Filename: .kdf

refresh load all

[✓ kdf] tif

select Storage Directory

/home/manfred/out

Evaluation

Amount of Evaluations: 1

Offset(Evaluation #): 1 Maximum: 0

Status

not started yet...

Errors (the following Files were not evaluated):

no Problems so far...

Algorithm

Normal Fit [✓ Fast Algorithm] Use Hardware

Start Fit

Figure 5.3: The graphical user interface of fastSPDM

The interface is programmed in a way that all parameters are read out not until the start button is pressed. So all variables take over the values that are defined at that moment.

5.1.3 Tools to Evaluate the Data

In addition to the simulation, the algorithm needs also to be tested with real data. To give the user a feeling of the finding process and a certain confidence in the program, there are several tools available:

CheckOrte

Using `CheckOrte` it is possible to select a small area, containing 9 pixels, from the final high resolution image. The selected area is extracted from the data stack and printed against the corresponding image number. So the temporal development of the intensity of this area is displayed (Fig. 5.4). If a spot is found at this area, the corresponding frame is marked in the plot. So the user can test if there has really been a signal at the calculated position.

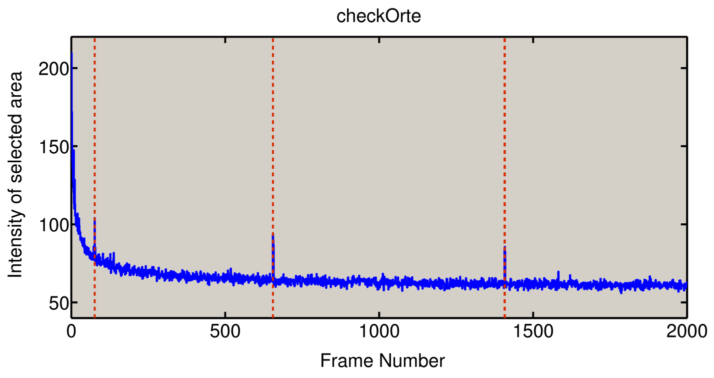


Figure 5.4: Check Orte: Detected signals are marked by vertical lines

TestOrte

In contrary to the script described above, `testOrte` analyses the distribution of the detected positions within one whole frame. Therefore, one frame is displayed and each location found on that frame is marked on the screen. Using this tool, the user can very quickly decide if the found positions fit the data. For a convenient usage it is possible to slide through the whole data stack by pressing *enter*. Figure 5.5 shows the display of one frame containing the marked locations.

Multiframepoints

A very important tool for the analysis of the images is the function `find_multiframpoints`. It is required to increase the localization accuracy of certain spots and provides for instance the possibility for a later analysis of molecule clusters.

The Estimator algorithm regards only spots on single frames independently from the spots on the later frames. However, it occurs that the same location is found in several following frames. It can be assumed that these locations all depend on the same molecule that glows over several frames and is detected on a sequence of images. Using the information of all locations that correspond to the same spot, the data can be summarized. After this operation, the final location has a much higher accuracy. All information needed for these calculations can

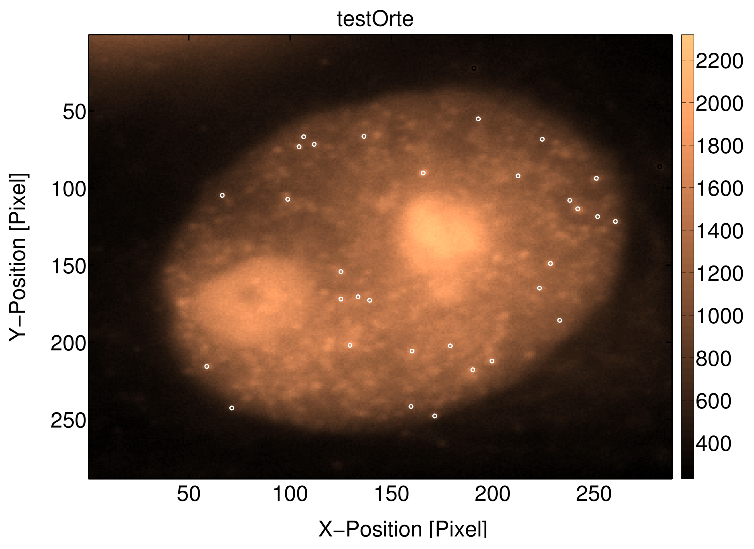


Figure 5.5: TestOrte: Markers define detected locations within the frame

be extracted from the `Orte`-matrix. The original image-stack is not required anymore.

To summarize multiframe points the data matrix has to be searched for close-by locations in successive frames. This requires comparing the position of each point of a frame with the positions of all points in the next frame. If two points lay within a certain distance that is defined by the mean localization error, a *multi-frame-point* is found and the next frame is analyzed whether the spot can still be detected. This is repeated until the first frame is found in which the position of the spot remains dark.

This method has a calculation complexity of the order $\approx O(n^2)$. For n detected locations in each frame. However, the spots containing to one frame have first to be extracted from the whole `Orte`-matrix what requires additionally processing time.

In order to speed up the `find_multiframpoints` algorithm the locations of the original `Orte`-matrix can be sorted by frame number and stored in an own container variable for each frame. So the whole matrix has to be searched only once for sorting. The later search for multi-frame-points can be done only on these container variables. This method is used in the final implementation of `fast_multiframepoints`.

If a set of N signals is found in the `Orte`-matrix, the corresponding entries have to be combined. Most of the parameters are quite easy combinable. The maximum intensity *Imax* as well as the whole Intensity Q are summarized by an addition:

$$\widetilde{Imax} = \sum_{i=1}^{N} Imax_i \qquad\qquad \widetilde{Q} = \sum_{i=1}^{N} Q_i$$

It can be assumed that all intensities of each ROI can be summarized to one signal containing the total intensity of the N signals, distributed over the pixels of one ROI. So the parameters can be calculated from each single location by following formulas:

$$\widetilde{\mu} = \frac{1}{\widetilde{Q}} \sum_{i=1}^{N} \mu_i Q_i \tag{5.11}$$

$$\widetilde{\Delta\mu} = \sqrt{\frac{1}{\widetilde{Q}^2}\sum_{i=1}^{N}\Delta\mu_i Q_i} \tag{5.12}$$

FastOrte2StdBild

A matrix containing locations is only the first step to evaluate localization microscopy data. The biological structures have to be visualized in a high resolution image. To generate this image there is already a Matlab script available. It describes the principle of visualization by Gaussian blurring mentioned in section 4.2.5. The idea of the existing script was to insert all spots having a certain localization accuracy in an image and apply a convolution of the whole image with a Gaussian function of the corresponding width. This convolution requires a lot of calculation time and having a large number of signals, the rendering process can take even longer than the actual calculation of the locations.

This is why there was a demand for a faster method to draw that image that is realized in the `FastOrte2StdBild` Matlab function. The idea is to avoid the convolution with the Gaussian distribution on the entire image. Instead of the whole image, only a small stencil of the size of one blurred spot is generated and inserted at each position having the corresponding localization accuracy. This strongly accelerates the rendering process.

Simulation Tools

The simulation described in section 5.1.1 is implemented in several functions. These are executed by one single Matlab script that also evaluates and prints the results. All parameters, especially the different simulated signal intensities, can be defined in the top level of the simulation: `EstimatorSimulation`. After the script has finished, the results of each simulation run can be found in the global variable **results**. The result is also stored in the simulation directory.

5.2 Hardware Implementation

The main work of this thesis was done in this chapter. After the algorithm was defined and its practicability was confirmed it is translated into hardware. Because the hardware usability of the algorithm was kept in mind during the development of the Matlab implementation, the most of the translation can be done straightforward. Therefore, only the complicated and important parts are described in this section.

5.2.1 Background Handler

The background handler is to implement the idea of the moving average background map as it is described in section 4.2.1. But the concept of the background map causes a severe problem because each new background map requires the information of the previous background map. This can only be realized by the feedback of a loop, and as mentioned in section 2.4.1, loops are strictly forbidden in VisualApplets.

Nevertheless, with the help of Silicon Software a solution was found. They offer an additional plug-on FPGA-card, called **Pixelplant PX** to extend their frame grabber cards and also to realize loops in special cases. Using that Pixelplant it is possible to reuse the former background map to calculate the map for the following image.

To connect a Pixelplant plug-on card in VisualApplets there exist special operators to receive and transmit the data between the two FPGA-cards. A loop then is created in the following way: the data stream is sent to the Pixelplant and without processing anything it is transmitted back to the frame grabber card. The data stream at the output of the receive operator can now be used to calculate the next background map.

The first background map is described in the current implementation simply by the first image. Consequently, the first spots can already be detected in the second frame. By choosing the *BGF*-parameter in the background formula (Equ. 4.7) to 8 or 16 the division can be avoided and replaced by a simple bit-shift of 3 resp. 4 bits. In the final implementation the *BGF* is set to 8.

In order to increase the calculation accuracy of the background handling, the 16 bit integer values of the image pixels are transformed to 13.3 bit fix point values. The three additional digits can simply be realized by shifting each pixel value of every incoming image by 3 bit. The three digits are kept only for the calculations to generate the background map and the difference image. The pixel values of the difference image are again represented by 16 bit integer values. As in the Matlab implementation, negative values in difference image are not allowed and set to zero.

5.2.2 Signal Finder

The search for signals in the images works as in the Matlab implementation. A threshold is defined and local maxima having an intensity above this threshold are identified as the signals of glowing molecules. As in the Matlab implementation an image filter is applied before the local maxima are searched. But in contrary to the software implementation the image filter is a very cheap operation:

To describe the sum of all surrounding pixels for each pixel a special structure of VisualApplets can be used that describes exactly what is needed for the image filter. It centers an $n \times m$ frame around each pixel. By applying a sum over this 3×3 kernel structure the filter is already realized.

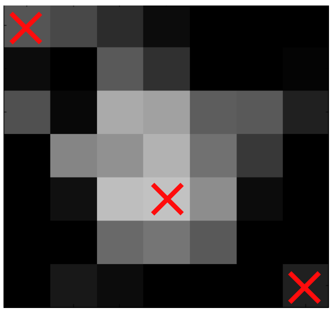
The determination of the positions of the local maxima can be realized in a similar way: instead of the `sum`-operator a `max`-operator can be applied on the 3×3 kernel structure of the filtered image. If the maximal value of the 3×3 kernel is located in the center of the frame the maximum is found.

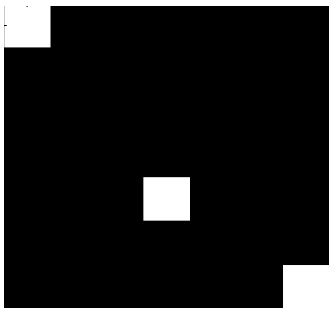
Since the signal finder is also pipelined, each pixel runs through the routine that detects the local maxima and the thresholding is applied afterwards. An illustration of how these operations effect the values of an image is shown in figure 5.6.

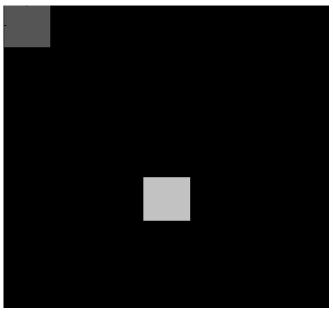
The final output of the signal finder are three streams containing the x- and y-position and the pixel value.

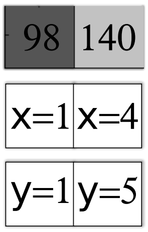
With the found coordinates the signals can then be extracted into ROIs. For the extraction, a special **M**-operator exist: `ImageBufferMultiRoiDyn`. This operator gets as input the stream that contains the background-freed difference images, and the coordinates of the corners of the ROI. For each set of coordinates, the operator stores one ROI into the external RAM. The size of a ROI is chosen in a way that a whole spot can fit in it. As in the Matlab implementation the width and length of a ROI is set to **7 pixels**. This is a fix value and can only be changed by a complete redesign including a new synthesis etc. .

If too much signals are found within one image an overflow can occur. This happens if signals are found faster than the ROIs can propagate through the following pipeline steps. To

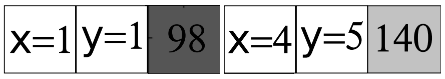

(a) Image containing 3 local maxima


(b) Binary image with positions of local maxima marked


(c) Local maxima in original image above threshold



(d) Pixelvalue and coordinates



(e) Final frames containing coordinates and value of central pixel

Figure 5.6: Example of the cluster finder: in a) a filtered image is displayed. The finder marks local maxima (b). Then all the pixel values, which lie below the threshold at the marked positions are set to zero (c). All zero pixels are deleted and the pixel value and the x and y coordinates from the remaining values are stored (d). In the last step the three data streams are combined into a single stream (e).

avoid this, the fill-level of the memory can be read out. This can be used to throttle the stream of new images that are sent to the FPGA-card if required.

Cluster Separator and Cutoff

Before the precise location of the spot center can be determined, the ROI has to pass these two operations. The `cluster separator` and the `cutoff` are implemented straightforward as in the Matlab version of the algorithm. However, the generation of the comparison frames for the `cluster separator` is worth mentioning. These frames are realized by special frame buffers. Using these buffers the order of rows or columns can be swapped. This is used to create new frames like it is done in the Matlab implementation, shown in figure 5.2(b). The comparative frames can very simple be compared to the original ROI by using an ordinary comparison operator. If the pixel value of the comparative frame is smaller than in the original ROI the pixel value is cleared.

5.2.3 Parameter Estimation in Hardware

After the ROI is cleaned from noise and intersecting spots the desired parameters can be determined. The implementation of the estimators for the parameters is done like described by equation 5.5, 5.6 and 5.7. Each sum in the formulas can be described by accumulators.

During the implementation it has to be considered that operations like divisions and square

roots require lots of hardware resources. So these operations have to be saved where possible. Since all calculations in hardware are fix-point calculations the computations of the parameters in hardware loose exactness. To avoid numerical artifacts in the results following restrictions are considered:
The expected localization accuracy of the Estimator will not reach below 0.01 pixels.
So *8-bit* $\equiv 1/256 \approx 0.004$pixels should be a sufficient fix-point accuracy for the position of a molecule. Since the accumulators calculate with integer values, this fix-point accuracy is achieved by a left-shift by 8-bit before a division is applied.

The same considerations are valid for the calculation of the width of the signal (σ_x, σ_y). Because square roots are very expensive operations in hardware, the values for the width are returned as squared values (σ_x^2, σ_y^2). If necessary the root can be calculated later in software.

Fix-point accuracy of μ_x, μ_y, σ_x^2, σ_y^2 is defined by: *8-bit* $\equiv 1/256 \approx 0.004px$

To save further resources, the spot width σ^2 is calculated as a mean value of the original width in x and y direction. This can be done because most detected spots are round.

The error of the location is implemented as equation 5.7. However, the square root is omitted to save resources. In addition to the location of the molecule, the localization error is also a very important size. In order to achieve the same fix-point accuracy of 8 bit as the other parameters a fix-point accuracy of 16-bit is required for the squared value of the error. Since the error is a quite extensive calculation that needs a lot of hardware resources, it is also implemented as a mean value of $\Delta\mu_x$ and $\Delta\mu_x$.

Synchronization of Global and Local Coordinates

The reader would have noticed that the estimation of the parameters within the ROI only returns the local position within the borders of the small ROI. But to localize the molecule in the entire image the estimated values have to be combined with the global coordinates. The global coordinates are available as they were used to extract the ROI from the image. But the calculation of the final location is slightly complicated. The problem is that after the **M**-operator `ImageBufferMultiRoiDyn` the streams of the local- and the global coordinates have to be resynchronized. This is made even more complicated by the `cluster separator` that contains additional **M**-operators. What has happened after synchronization has been an error in the result frame: the global coordinates did not correspond anymore to the parameters of the ROI.

So in the first approaches of the hardware realization this synchronization was not implemented. Without that synchronization the global coordinates and the estimated parameters of the ROIs had to be separately transmitted to the hosting PC in two different *direct memory access* (DMA) channels. The combination of the coordinates then finally was done in software. But the two DMA streams were not synchronized, so the global coordinates corresponding to one estimation had first to be found in the frames of the other DMA stream, what required additionally synchronization information that had to be extracted from the image resp. the ROIs.

But then it became clear that empty images containing no spots are the reason for the synchronization failure. Since a data stream has a different behavior if no spots where found in an image, than the `ImageBufferMultiRoiDyn`-operator, different data arrives at the synchronization operator. The problem was fixed by inserting one generated spot in each frame.

In order to not affect the evaluation of the images the spot is inserted in the upper left corner of the frame. After the image filter is applied on the image, the center pixel defines always

5000	5000	5000
5000	5000	5000
5000	5000	5000

Table 5.2: Shape of the spot that is inserted in each image at I(6-8,6-8).

the local maximum of the spot. It is also found by the `cluster finder` and can be identified by its maximum intensity of 5000. Furthermore, the inserted spot signals the begin of a new image. So its appearance in the results can be used to reassign the number of the original image to the estimated values of an extracted ROI.

5.2.4 Properties of the FPGA-Card

The final hardware on that the algorithm is implemented is the microEnable IV VD1-CL frame grabber card from Silicon Software (Fig. 6.12). To realize the background calculations and to provide additional resources, a Pixelplant PX200 is plugged onto the VD1.

The VD1 contains 256MB RAM to buffer incoming images and a Xilinx XC3S 1600E FPGA for the image processing. Its PC interface is a PCIe×2 that means a maximum transmission rate of 200 MB/s can be achieved. The pixel clock is specified by 85 MHz.
The main selection criteria for this platform is its ability to receive data by DMA input. Most of the frame grabbers from Silicon Software can only receive images by a direct connection to the camera called *camera link*. But since the images are already stored on the computer, the DMA input is required.

5.2.5 Controlling Software

On the one hand the translation of the algorithm into hardware using VisualApplets is the main part of the work. On the other hand the hardware has to be accessed and controlled by the user with a program running on the PC. The concept for this controller code is very important to provide fast data transfer to the card. For testing the design, Silicon Software provides a simple interface to visualize the output of the FPGA-card. But the final controller code has to be developed specifically for each design. The task of the controller code is to send the image data to the FPGA-card and to receive the results of the computations. Additionally, the code should offer an interface to the user to select the image stacks and to adapt the parameters of the Estimator algorithm like it is realized in the Matlab GUI (Sec. 5.1.2).

Silicon Software provides libraries and header-files for the software development kit (SDK) of Microsoft Windows `Microsoft Visual C++ 2010 express`. This developing platform can be downloaded from Microsoft Windows for free [1].

All functions for communication of the hosting PC with the FPGA-card are provided by driver functions in own libraries. Especially the data transfer using the DMA controller can be realized by provided functions.

Principle Setup of the Controller Code

To describe the frame grabber in C++, a frame grabber structure is created. This structure is initialized with the path of the hardware programming file (hap-file) by the `FG_init()`-function.

[1]A detailed description of the setup of a Visual C++ project is given in the appendix B.

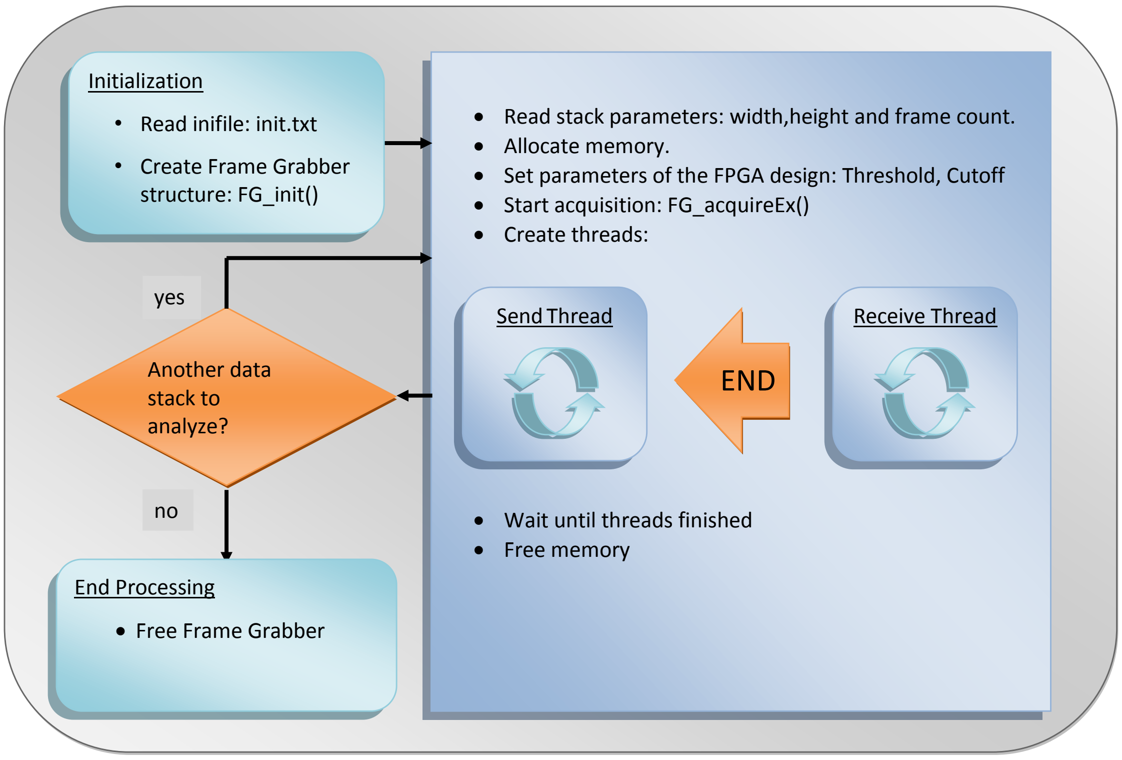

Initialization
Read inifile: init.txt
Create Frame Grabber structure: FG_init()
Read stack parameters: width,height and frame count.
Allocate memory.
Set parameters of the FPGA design: Threshold, Cutoff
Start acquisition: FG_acquireEx()
Create threads:
yes
Another data stack to analyze?
Send Thread
END
Receive Thread
no
Wait until threads finished
Free memory
End Processing
Free Frame Grabber

Figure 5.7: Scheme of the controlling program for the frame grabber

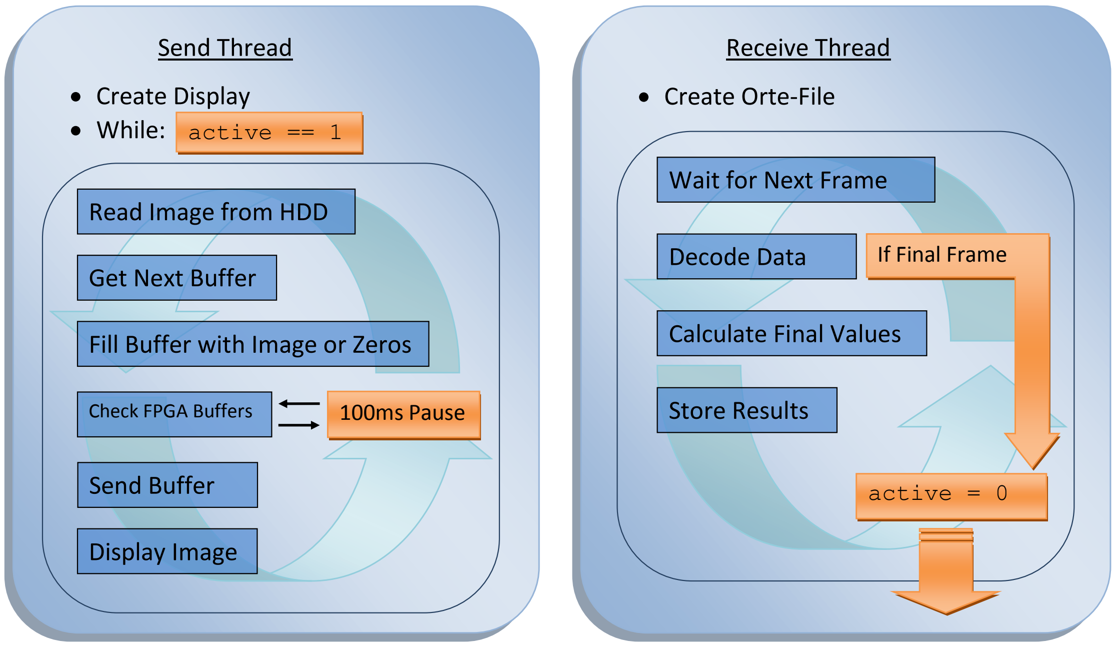


Figure 5.8: Scheme of the two threads that handle the data transmission.

The next step is to define the parameters of the algorithm and the image data that is to be processed. To receive the results of the Estimator algorithm sufficient memory has to be allocated by a special function called Fg_AllocMemEx(). The final data transfer then is controlled by the two functions Fg_sendImageEx() and Fg_AcquireEx(). These functions receive, beside others, pointers to the image data resp. pointers to the allocated memory for storing the results.

In the most simple case for each sent image one frame is received at the DMA-input channel and both DMA-channels are accessed alternately.

After the last image is transfered to the card and the final frame is received from the FPGA-card the DMA-channels can be closed by the function `Fg_stopAcquireEx()` and the allocated memory can be freed by `Fg_FreeMemEx()`. Finally, the frame-grabber structure is deleted by `Fg_FreeGrabber()` and the program is finished. This is only a short introduction to create a simple project in Visual C to control the frame-grabber. A more detailed description of all functions to control the hardware can be found in the documentation of the Runtime Environment on the Silicon Software drivers CD.

Data Management

Before sending images to the FPGA-card, it has to be initialized. The initialization is mainly done by defining the parameters of the DMA-channels. Therefore, the length and the width of the frames that are transmitted by the DMA-channel has to be defined and the required memory has to be allocated. At the receiving side, the height of the DMA-frame defines the count of the estimations that are transmitted in one frame. The width of this frame is a fix value and is defined by the count of the extracted values. In the final version of the implementation, the FPGA-card still transmits some values that are required for checking if the synchronization worked well. The 15 values that are transmitted in one estimation are listed in table 5.3. The values $I(x,y)$ and Q_{mid} both describe the pixel-value of the center pixel of the ROI. So these two fields must have the same value to signal a successive synchronization of the streams.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
gl_x	gl_y	$I(x,y)$	$ROInr$	I_{max}	μ_y	μ_x	$\overline{\Delta\mu_1^2}$	$\overline{\Delta\mu_2^2}$	$\overline{\sigma_y^2}$	$\overline{\sigma_x^2}$	Q	Q_{old}	$ROInr$	Q_{mid}

Table 5.3: Values returned by the frame grabber. Gray fields are values extracted from the difference image (1-4). White fields are values that are extracted from the ROI (5-15). The error is described by 32 bits. It is split on two pixels. Pixel 8 defines the integer value and pixel 7 defines the decimal value. Values with a bar describe mean values for x and y direction.

The main focus during the development of the controlling software, was on the performance of the data transmission between the card and the hosting computer. A fast FPGA-implementation provides no advantage if the image transfer to the FPGA-card and the receiving of the results takes more time than the actual calculations. Since the progress of both the receiving and the sending is controlled by the FPGA-card, the controlling-software has to wait until the card is ready to receive the next image and on the other side it has to wait until the next frame, containing the results, can be received. To avoid that one process has to wait for the other, each of the two tasks are realized in one independent thread. The two threads can work simultaneously on a modern multi-core workstation. This solution provides very fast data transmission to the FPGA-card.

The **send thread** is not only responsible for sending the data to the FPGA-card. Since a data stack can become very large it can take a lot of time to load it from the hard disk. Stacks can even exceed the size of the main memory of the computer.

To save initialization time and to reduce memory usage, the image data is initially loaded by the send thread: the data is read from the hard disk just before it is transmitted to the FPGA-card (to send the data to the card the function `Fg_sendImageEx()` is used). So only one image and not the whole stack has to be loaded into the main memory. Moreover, the time when the FPGA-card is still occupied with receiving an image is used to load the next image.

The last step of the sending process is to visualize the sent image. This is not really necessary but since the visualization function needs only few calculation time it is a nice presentation of the progress to the user.

The **receive thread** collects result-frames from the frame grabber and stores the final `Orte`-file that contains the molecule locations on the hard disk. For each separated ROI one estimation arrives at the output of the frame grabber. A number of estimations is summarized in one **result-frame** and returned by DMA to the PC. This *estimation-count* can be defined in the controller code and is set to 500. That means with each frame transmitted by the FPGA-card the results of 500 different estimations are transmitted.

The driver function `Fg_getLastPicNumberBlockingEx()` stores the result-frames in the previously allocated memory. This memory is managed as a ring buffer. If a frame is stored completely in the ring-buffer it can be used for further processing. The aforementioned driver function waits for the desired frame to be stored completely and returns the total count of received frames. So it can be checked how much frames still wait for further processing. The current data frame is returned by a pointer that contains the position of the first frame pixel.

Before the estimations can be stored, the received data has to be decoded. The received data is packed into an own `result`-class. The function of the `result`-class that stores the data decodes the results, reorders the values and combines the global coordinates of the ROI within the original image with the precise, estimated position of the spot in the ROI. The final order of the entries of the `Orte`-file is the same order as in the Matlab implementation (Tab. 5.1) to provide compatibility to further processing functions:

As mentioned above, the error of the localization is returned as a squared value split into two values. So these two values represent only a mean value for the localization error in x and y direction.

Before the final error value can be stored in the `Orte`-file, the two values have to be combined and the final error has to be calculated.

$$\Delta\mu = \sqrt{\frac{2^{16} \cdot \Delta\mu_2^2 + \Delta\mu_1^2}{2^{16}}} \tag{5.13}$$

The integer values of μ_1 and μ_2 received from the frame grabber are interpreted as float values in the calculations for the final value.

The end of the processing of a data stack is determined by the receive thread. Because the results of the calculations of one frame have a certain latency until they can be received from the FPGA-card, the processing pipeline has to be kept running after the last image is sent to the FPGA-card. This is done by sending empty frames to the card. Since in each frame at least the generated spot is detected, a final frame containing only the estimations of the generated signal is received after the last real signal is stored. This frame containing artificial estimations signals the end of the processing.

5.2.6 Integration in Matlab

Since the programming of an own GUI for the C++ code is quite a lot of work, an alternative was searched. The first idea was to integrate the controlling of the FPGA-card into the Matlab implementation. Therefore, the Matlab arrays have to be transformed in C++ compatible array structures. This can be done by a special glue-code called `mexfunction` [38]. The `mexfunction` is a small collection of C++-libraries provided by Matlab to pass the arguments of the Matlab environment to C++-compatible variables and vice versa.

The advantage of this approach is that in addition to the user interface all functions of the Matlab implementation like the visualization and the `multiframepoints`-functions can be reused. Further, the results are immediately usable in the Matlab environment for further calculations e.g. it is common to acquire multiple stacks of the same section of a probe, marked with different fluorophores and to combine them finally to one multicolor high resolution image.

The integration of the frame grabber card in Matlab was realized and the first usable environment of the FPGA-card for image-processing could be controlled by Matlab. However, a few things have caused problems:
One disadvantage of this implementation is the way Matlab accesses the image data. Before any calculation can be started, the whole image stack has to be loaded into a Matlab variable and so the whole image stack has to be loaded into the main memory. So the largest stack that can be processed has to fit completely into the main memory although only one image is required for the calculations at once. Due to the memory requirement of Matlab and the operating system itself image stacks of 300 MB size can already cause problems on systems containing 1 GB main memory. So a better solution is required.

5.2.7 Executable

The second implementation to make the frame-grabber usable without an own GUI is done as an executable. All parameters as well as the specification of the files that are to be analyzed have to be defined in an ini-file. Also the path to the `hap`-file that contains the Estimator algorithm for the frame grabber has to be specified. A readable ini-file is composed in the following way:

- Attributes are defined in []
- The value of the attribute is defined after '='
- The end of an attribute is defined by ';'

Two different image file types are allowed. The program is able to read `tif`-image stack and `kdf`-image stacks. Also multiple files can be processed as shown in figure 5.9.

```
#Initialization File

[Hapfilepath]
Hapfile=C:\\fastSPDM\\estimator16bit_VD1.hap;

[Threshold]
Threshold=80;

[Datafiles]
Datafiles=
C:\\Tiffs\\stack1.tif,
C:\\Tiffs\\stack2.tif;

[Cutoff]
Cutoff=20;

[Pixelsize]
Pixelsize=102;
```

Figure 5.9: Example of an initialization file for the executable. Two tif-stacks are defined. **Threshold** defines the minimum value of a signal above the background; The value defined for **Cutoff** is subtracted in order to decrease noise before the parameters are estimated; **Pixel-size** is only a scaling factor that defines the dimension of the detector pixels in nanometers.

Chapter 6

Results

This chapter summarizes the results of the measurements of this thesis using the final implementation of the Estimator algorithm. At first, the results of the Matlab simulation described in section 5.1.1 are presented. It comes out that the implemented algorithm provides an as good parameter estimation as the compared iterative least squares fits. Additionally, measurements with real data are described. The second part of this chapter shows the results of the measurements using the hardware implementation are compares them to the measurements presented in the first part. Further, a comprehensive timing analysis of the Matlab and the FPGA-based implementation is described and compared to the performance of common least squares methods. Finally, the total speedup compared to the former available algorithm is presented.

6.1 Performance of the Background Map

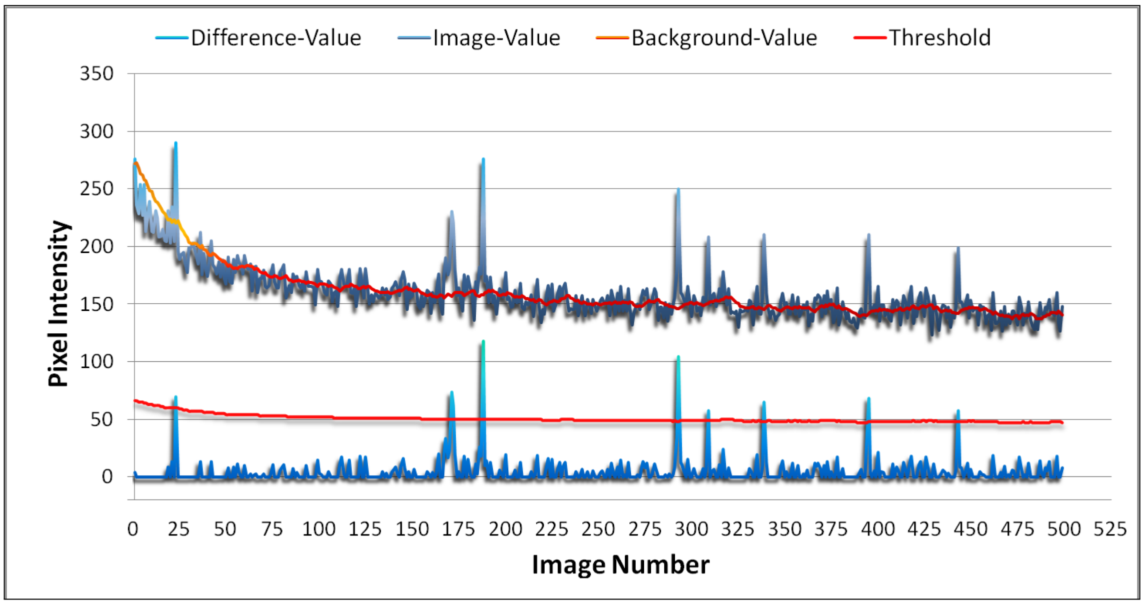


Figure 6.1: Temporal development of one pixel value, compared to the background map. The threshold is defined as $th = 4 \times \sqrt{BG}$.

The background map is recalculated for each image after the formula described in equation 4.7. Figure 6.1 shows the development over time of one pixel in the original image, the background map and in the resulting difference image. As it can be seen in the diagram, the background map achieves very well the requirements (Sec. 4.2.1). The red line follows the characteristics of the image pixel very fast, but suppresses the fluctuations of the noise. In this way, also weak signals can be detected as a glowing spot.

6.2 Results of the Simulation

As described in section 5.1.1 the algorithm was evaluated using simulated data. Therefore, a comprehensive Monte-Carlo-simulation was implemented in Matlab. In order to compare the parameter values calculated by the tested Estimator algorithm, the simulated spots are also analyzed by an iterative least squares fit from the Matlab library called `lsqcurvefit`. This function describes a Levenberg-Marquardt implementation of a least squares fit and fits a two-dimensional Gaussian distribution. Since the `lsqcurvefit` algorithm is influenced by the `cluster separator` and the appliance of the `cutoff`, the fit function is executed on a spot that is not yet preprocessed by these two tasks.

In addition to the Estimator and the Levenberg method, another analytic method, the FluoroBancroft, is also implemented and tested with the weakest background intensity of 50. It is also compared to the other methods. The performance of the FluoroBancroft algorithm is also shown in figure 6.2(a). However, as the diagram shows the localization accuracy of this algorithm is much worse than the results of the other methods. For this reason, the FluoroBancroft method is not further analyzed.

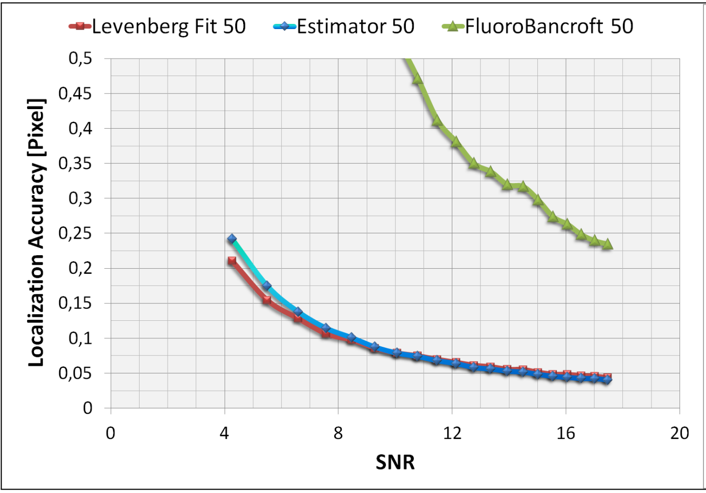
The simulation is basically split into two parts:
In the first part the localization accuracy of the Estimator algorithm is presented. The accuracy was measured for a small, medium and a high background intensity using 20 different signal intensities. The strength of the signals are spceified between $SNR = 4$ and $SNR = 18$.

In the second part of the simulation the SNR runs also from 4 to 16, but the intensity of the signal is fixed and the strength of the background is varied.

In addition to the localization accuracy, the error formula, resp. the calculated error is also evaluated. Therefore, the real localization accuracy (Equ. 5.8) is compared to the mean value of the mean calculated error. The calculated width of the signals is also presented and compared to the real value.

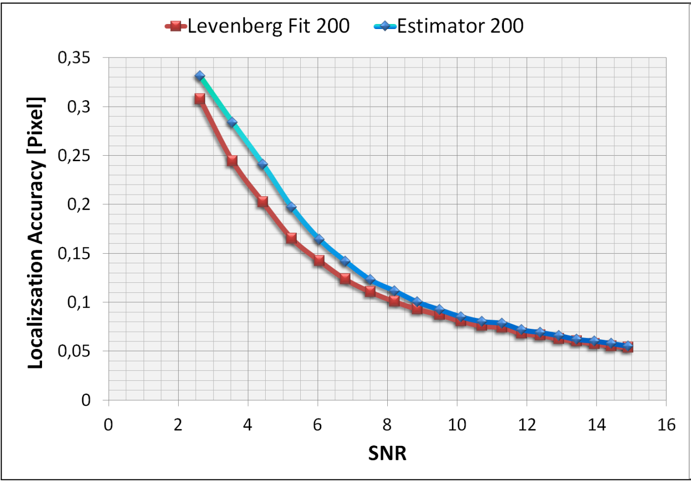
6.2.1 Simulation with Fixed Background Intensity

For each SNR, 4000 images containing exactly one spot are generated and analyzed. The results of this simulation are shown in figure 6.2(a)-6.2(c). The graphs show the mean localization accuracy in pixels for different signal to noise ratios compared to the results of the Levenberg fit method. As the diagrams show, for weak signals compared to the background, the accuracy of the Estimator algorithm is a little bit worse than the accuracy of the iterative fit. However, for higher signal intensities and little background, the accuracy of the Estimator becomes even better than the results of the Levenberg fit.



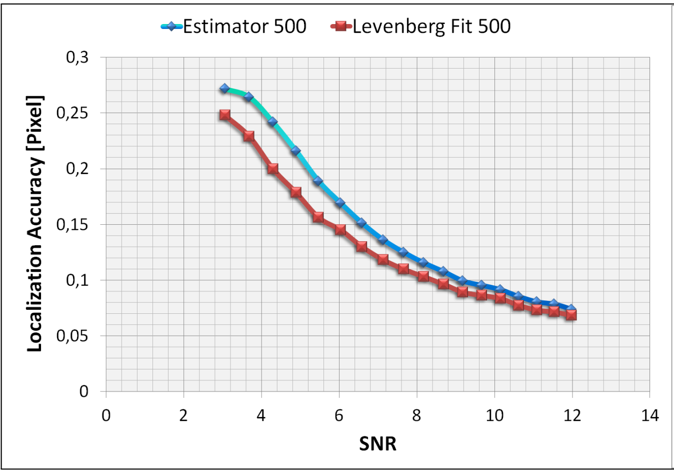
SNR	Estimator	Levenb.	FB
4,3	0,24	0,21	-
5,5	0,17	0,15	-
6,6	0,14	0,13	-
7,6	0,11	0,11	1,05
8,5	0,10	0,10	0,77
9,3	0,09	0,09	0,60
10,1	0,08	0,08	0,53
10,8	0,07	0,07	0,47
11,5	0,07	0,07	0,41
12,1	0,06	0,07	0,38
12,8	0,06	0,06	0,35
13,4	0,06	0,06	0,34
13,9	0,05	0,06	0,32
14,5	0,05	0,06	0,32
15,0	0,05	0,05	0,30
15,5	0,05	0,05	0,27
16,0	0,04	0,05	0,26
16,5	0,04	0,05	0,25
17,0	0,04	0,05	0,24
17,5	0,04	0,04	0,23

(a) Background intensity 50



Qges	SNR	Estimator	Levenberg
500	2,6	0,331	0,308
700	3,5	0,284	0,244
900	4,4	0,241	0,202
1100	5,3	0,197	0,165
1300	6,0	0,164	0,143
1500	6,8	0,142	0,123
1700	7,5	0,123	0,111
1900	8,2	0,112	0,101
2100	8,9	0,100	0,093
2300	9,5	0,092	0,087
2500	10,1	0,085	0,081
2700	10,7	0,080	0,076
2900	11,3	0,079	0,074
3100	11,8	0,072	0,068
3300	12,4	0,069	0,066
3500	12,9	0,066	0,063
3700	13,4	0,062	0,060
3900	13,9	0,060	0,058
4100	14,4	0,058	0,055
4300	14,9	0,055	0,054

(b) Background intensity 200



Qges	SNR	Estimator	Levenberg
900	3,1	0,272	0,248
1100	3,7	0,264	0,229
1300	4,3	0,242	0,200
1500	4,9	0,216	0,179
1700	5,5	0,189	0,156
1900	6,0	0,169	0,145
2100	6,6	0,151	0,130
2300	7,1	0,136	0,118
2500	7,7	0,125	0,110
2700	8,2	0,115	0,103
2900	8,7	0,108	0,096
3100	9,2	0,099	0,089
3300	9,7	0,095	0,087
3500	10,1	0,091	0,084
3700	10,6	0,085	0,077
3900	11,1	0,081	0,073
4100	11,5	0,079	0,072
4300	12,0	0,074	0,068

(c) Background intensity 500

Figure 6.2: Localization accuracy of the Estimator over SNR for different background intensities. Width of generated signals = 1.4 pixels. 4000 signals have been analyzed.

6.2.2 Simulation with Fixed Signal Intensity

In this simulation the signal intensity was fixed at an intensity of 3300 what means a maximum intensity of the signal of about 267. The different SNRs are realized by changing the background intensity of ≈ 4 at $SNR = 16$ to over 4000 at $SNR = 4$. Since a high background intensity value means a very noisy signal, the intensity of the generated signals has to be strong enough that they still can be detected in the whole *SNR* range. In this simulation also 4000 images containing one spot are generated.

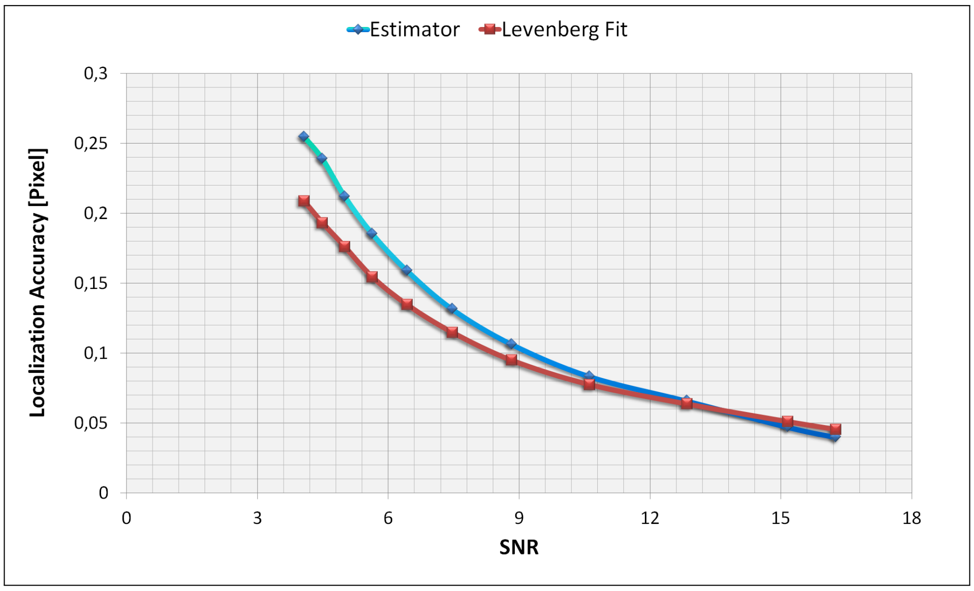


Figure 6.3: Localization accuracy over SNR using a fixed signal intensity of 3300. Signal width = 1.4 Pixel

As it is shown in figure 6.3 the localization accuracy is similar as in the first simulation for the different *SNR*s. The localization accuracy of the Estimator becomes significant better than the accuracy of the Levenberg fit at weak background intensities.

6.2.3 Evaluation of the Error Formula

The value of the localization error determines the confidence interval of the result of the calculations. So the mean calculated error (Equ. 3.26) has to be of about the order of the real localization accuracy displayed in figure 6.2(a). The mean value of the calculated error values is determined from the results of the simulation using a fixed background and compared to the real localization error (Fig. 6.4).
The diagram shows that the calculated error fits quite well to the real localization accuracy.

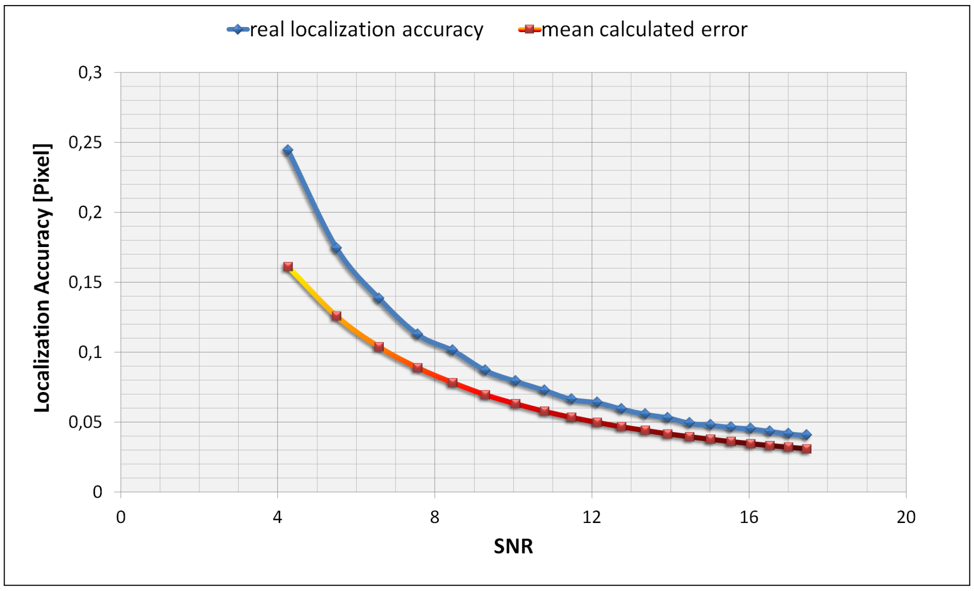


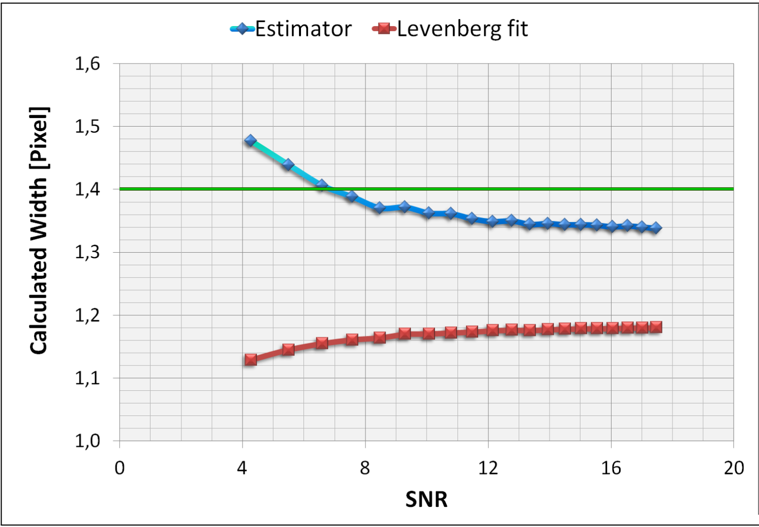
Figure 6.4: Mean value of the calculated error returned by the Matlab implementation compared to the real mean localization accuracy. The background intensity is 50. Spot width = 1.4 pixels

However, the error is specified a little bit below the real localization accuracy ($\approx$ 0.01 Pixel), but this is not a problem since the error defines only the 1σ confidence interval of the position. For weak signals and low SNRs the error is specified so small because in a ROI with such weak signals only few pixels in the ROI center differ from zero and so it is hard to define the error correctly. The most important thing was that the curve of the calculated error has the same shape as the real localization uncertainty. This shows that the dependencies of the parameters in the error formula are defined correctly.

6.2.4 Evaluation of the Calculated Width

In addition to the positions of the molecules and the error of the positions, the Estimator algorithm determines the width of the detected molecule signals.

The width of the generated spots in the simulation is fixed to 1.4 pixels. This value was determined by a measurement using data of real images (Sec. 6.3.1). The results of the calculations of the Estimator and the Levenberg algorithm using simulated data are shown in figure 6.5.



SNR	Estimator	Levenb.
4,3	1,48	1,13
5,5	1,44	1,15
6,6	1,41	1,16
7,6	1,39	1,16
8,5	1,37	1,16
9,3	1,37	1,17
10,1	1,36	1,17
10,8	1,36	1,17
11,5	1,35	1,17
12,1	1,35	1,18
12,8	1,35	1,18
13,4	1,34	1,18
13,9	1,35	1,18
14,5	1,34	1,18
15,0	1,34	1,18
15,5	1,34	1,18
16,0	1,34	1,18
16,5	1,34	1,18
17,0	1,34	1,18
17,5	1,34	1,18

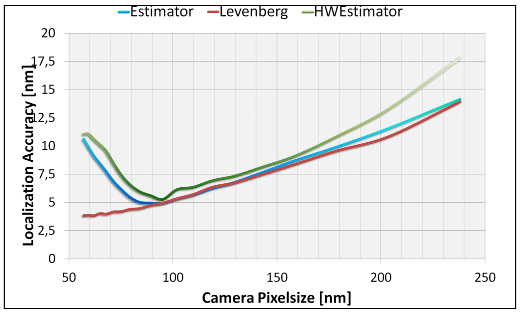
Figure 6.5: Calculated width of the detected spots compared to the real value of 1.4 pixels

As the diagram shows the Estimator determines the width of the spots very well. The difference of the real value and the calculated value is only of about 0.05 pixel. In contrast to the calculated value of the Levenberg algorithm which is always more than 0.2 pixels too low.

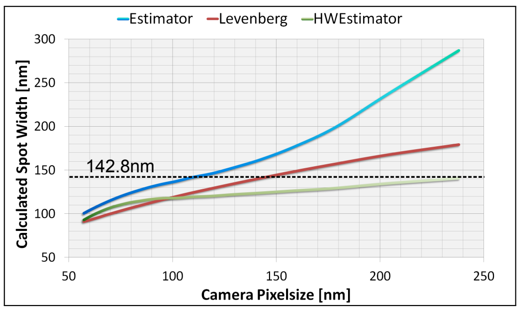
Influence of Different Pixel Sizes in the Detector

The setup of the simulation was once changed to measure the influence of the pixel size of the detector on the results of the Estimator. This was done to evaluate the performance of the Estimator for different microscopy setups. To simulate different detector sizes, the same amount of photons is distributed on a variable count of camera pixels. So the total intensity of the signal and the background is kept constant and the width of the generated signal is varied. In order to gain a certain statistic, 2000 spots are generated and analyzed for each camera pixel size. The final values shown in the diagrams are always mean values.

The measurement of the width in section 6.3.1 was made with a camera with an absolute pixel size of **PXS** = 102nm. This value can be used to calculate the resulting camera pixel size for different signal widths: $\mathbf{PXS}^* = \frac{102\text{nm}}{\sigma^*} \cdot 1.4$



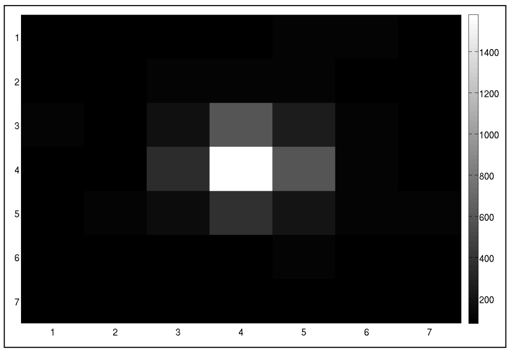
(a) Influence of the camera pixel size on the localization accuracy.



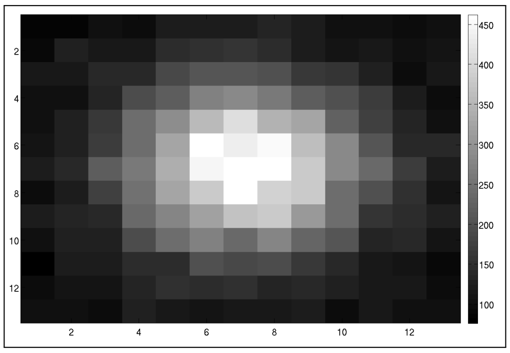
(b) Calculated width in nm for different camera pixel sizes. The real width of the generated spots is 142.8nm.

Figure 6.6: Results of the measurement of the influence of the camera pixel size on the localization accuracy

The diagram in figure 6.6(a) shows that the performance of the Estimator algorithm decreases rapidly below a camera pixel size of ≈ 95nm. As displayed in figure 6.7, if the camera pixels get smaller the intensity distribution is spread on much more pixels. So the intensity value of each single pixel gets lower and the relative strength of the noise increases. This may be a reason for the fast increase of the localization uncertainty installing smaller camera pixels.



(a) Pixel dimension = 238nm. Signal distributed on a 7x7 ROI



(b) Pixel dimension = 57.12nm Signal distributed on a 13x13 ROI

Figure 6.7: Intensity distribution of a signal having a total intensity of 2500 on different detectors

6.2.5 Timing Analysis of the Matlab Algorithm

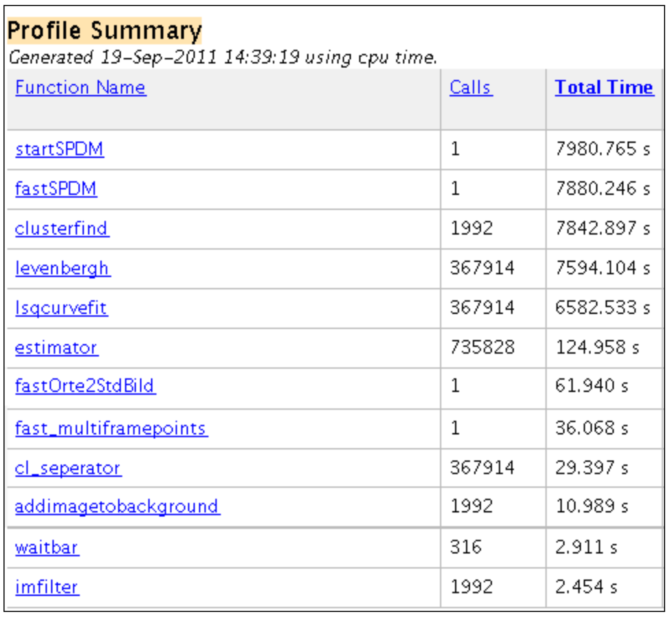
This section presents the results of the timing analysis of the Matlab implementation of the Estimator algorithm. During the development in Matlab, several ideas were implemented and the algorithm was sped up by each iteration.

All calculations of the simulations are executed on a HP laptop computer. The applied CPU is an Intel Core i5 450M [39] with a clock speed of $2 \times 2,4$ GHz. The installed main memory is 4 GB DDR3 having a clock speed of 1333 MHz.

As mentioned before, the calculation speed of an algorithm in Matlab depends strongly on the way it is described. As an example, the processing time of an implementation that uses for-loops for the processing of an image is compared to the processing time of a version implementing the same functionality by using Matlab native code style (Sec. 6.2.5).

For measuring the computation time, a whole stack of images is evaluated. The execution times of the individual steps are measured using the *Matlab profiler*. The Matlab profiler is a built-in profiling tool that measures the execution times of each step in the code.

The simulation is structured as follows:
To compare the execution times of the Estimator and the iterative least squares fit Levenberg every signal that is found in an image is processed by both functions. The data stack consists of 2000 images having a size of 288×320 pixels. Since the first 8 images are used to create the background map, 1992 images are searched for signals. In these 1992 images 367914 glowing spots are detected and processed. The results of this measurement are presented in figure 6.8.


Profile Summary
Generated 19-Sep-2011 14:39:19 using cpu time.

Function Name	Calls	Total Time
startSPDM	1	7980.765 s
fastSPDM	1	7880.246 s
clusterfind	1992	7842.897 s
levenbergh	367914	7594.104 s
lsqcurvefit	367914	6582.533 s
estimator	735828	124.958 s
fastOrte2StdBild	1	61.940 s
fast_multiframepoints	1	36.068 s
cl_seperator	367914	29.397 s
addimagetobackground	1992	10.989 s
waitbar	316	2.911 s
imfilter	1992	2.454 s

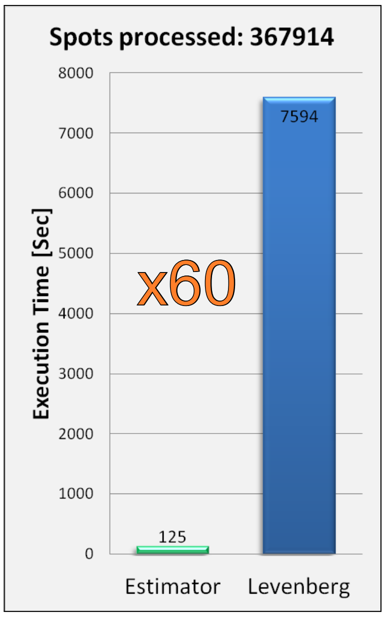


Figure 6.8: Timing measurement using the Matlab profiler[1]. Estimator compared to the iterative Levenberg fit. Total calculation speedup of a factor ×60 measured.

[1]startSPDM=GUI, fastSPDM=contains the other functions, clusterfind=finding signals over threshold, levenbergh(lsqcurvefit)=iterative fit, estimator=new non iterative algorithm, fastOrte2StdBild=Creation of the localization image, fast_multiframepoints=combines spots detected on following images at the same position to one

The diagram shows a speedup of the processing speed of the Estimator of approximately 60 times compared to the performance of the Levenberg fit. This is an enormous acceleration of data processing while maintaining the localization accuracy. Furthermore, this speedup is measured just for the calculation of the parameters of the small section of the signal (the ROI). If the execution time of the entire new algorithm is compared to the computation time of the Matlab program that was used before this diploma thesis started, this speedup is much larger. The execution time of the old Matlab code is at almost 38000 seconds for analyzing the same data stack. This means an acceleration by a factor of about **200** for the processing of the entire image stack, depending on the number of detected spots.

As the diagram on the left side of figure 6.8 illustrates, the parameter estimation takes up the greatest part of the computation time of the whole algorithm. This means that the total processing time for an entire data stack depends mostly on the number of detected spots.

Even the functions for generating the localization image (`fastOrte2StdBild`) and the combination of the points which glow for more than one frame (`fast_multiframepoints`), require less time than the total execution time of the Estimator function. The function that filters the entire image, before the spots are searched (Sec. 5.1) is surprisingly quick, and requires even less time than updating the progress bar. Finally, an overview of the fraction of the execution times of the single functions is shown in figure 6.9.

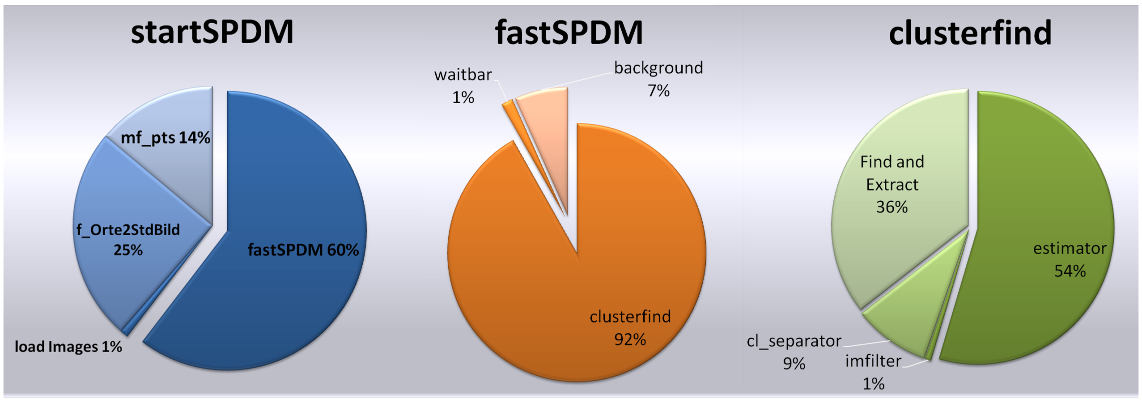


Figure 6.9: Execution times of the single tasks of the Matlab functions.

Since the profiler produces additional calculation effort, the total processing time of the final Estimator algorithm is measured again without the Levenberg algorithm, that was included only for comparison of the calculation speed, using the Matlab stopwatch `tic-toc`. For the data stack containing 367914 signals the entire data processing requires **148 ± 4 seconds**. The same algorithm using the iterative Levenberg method to determine the center of the spots instead of the Maximum-Likelihood-Estimator needs ≈**7822 seconds** and detects 366915 spots. Compared to the old algorithm that was used before this thesis started this is an even larger speedup. What means that also the image preprocessing and the way the spots were detected is much slower realized in the old version.

The old algorithm needs about ≈**37935 seconds** for processing this image stack and found only 113077 signals.

stronger signal, cl_separator=Separates overlapping signals, addimagetobackground=backgroundhandling, waitbar=Displays the progress, imfilter=filter for detecting signals.

In the remainder of this chapter, the execution time of the Matlab implementations are compared in a chart (Fig. 6.17) that also contains the analysis of the time measurements of the hardware implementation.

Execution Time of Loop Based Calculations Compared to Matlab Style

As mentioned before, the programming style has a big influence on the execution speed of calculations in Matlab. In order to show this, a little simulation was created, that measures the execution time of the background handling function `addimagetobackground` implemented as described in section 5.1 and the execution time of a second function that implements the same functionality in a nested for loop (Code in appendix A.1) that runs once over each pixel of every image.

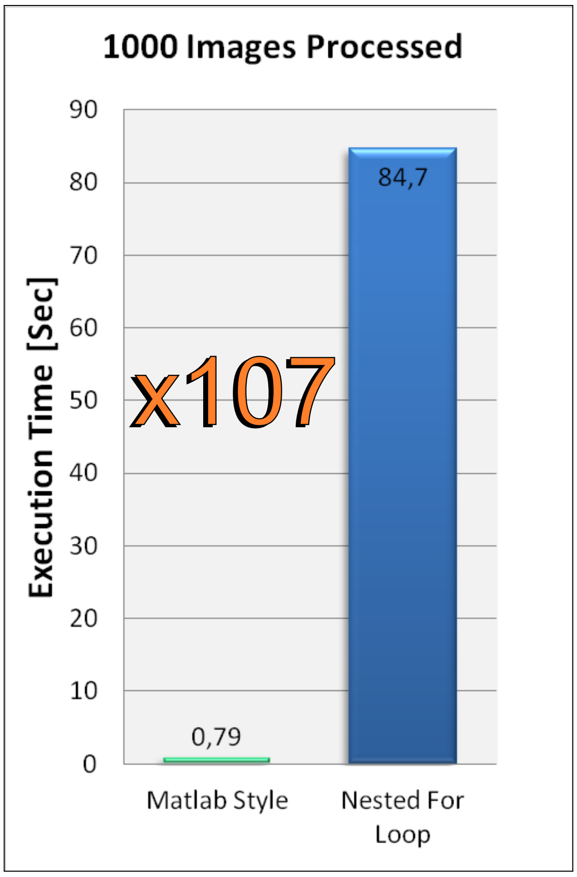


Figure 6.10: Execution time of a loop-based implementation of the background handler compared to the original function that is written in a common Matlab style. 1000 background maps of size 160x160 are calculated.

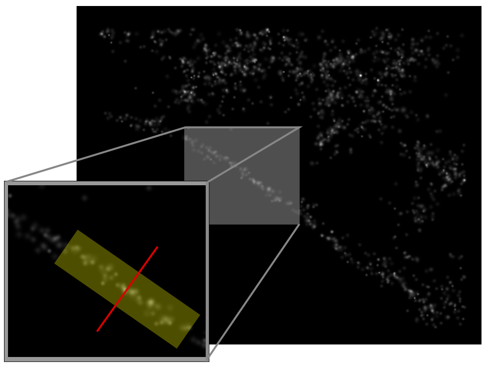
As shown in figure 6.10 the implementation using the common Matlab style is over 100 times faster than a nested for-loop description. This shows, that the calculation speed of an algorithm in Matlab strongly depends on the correct description of the functionality e.g. by avoiding for-loops if possible.

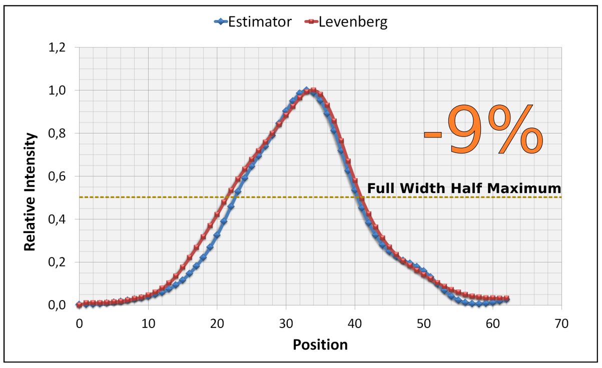
6.3 Results of Calculations Using Real Data

In section 6.2 the results of comprehensive measurements using simulated data are presented. But as promising as these results are, the algorithm still has to prove its performance at real data. The evaluation of the localization accuracy of an algorithm using real data is slightly complicated because the real positions of the molecule signals in an acquired image are not known. So how can the accuracy of a calculated value be determined?

The real position cannot be obtained from the image. However, the accuracy of the estimated position of a molecule can be determined by comparing it with respect to the position of other molecules.

For instance, if it is possible to localize molecules on a very thin straight structure, the width of the resulting line in the final image can be used in order to estimate or to compare the localization accuracy. The linewidth measurement is applied on a localization image displayed using the Gaussian blurring method described in section 4.2.5. The results of this measurement are presented in figure 6.11(b). As the diagram shows, the width of the line from the Estimator algorithm is slightly smaller: $\approx 9\%$. This proves that the estimator provides also very good results using real data.


(a) Section of an localization image containing a line like structure



(b) Full widths at half maximum are compared. Estimator: 17.75px / Levenberg: 19.43px

Figure 6.11: Linewidth measurement using real data.

The images of this measurement contain strong signals and the background is very low. Therefore, the localization accuracy of the spots can be assumed to be very high. The width of the line is determined as a mean value over its entire length (Fig.6.11(a)).

6.3.1 Measurement of the Mean Width of Real Signals

In the Matlab simulation, a realistic value for the width of the molecular signals is required. Consequently, the dimension of a typical signal has to be determined. To do this, the mean value of the width of about 20000 estimations is measured. The maximum pixel value (Imax) of all spots of this measurement is in the range of 200-300 which corresponds to an SNR of about 10. The resulting width is $\sigma = 1.36$ pixels. Regarding diagram 6.5, this value has to be corrected to a higher value. So, the value for the real width of a typical detected spot is:

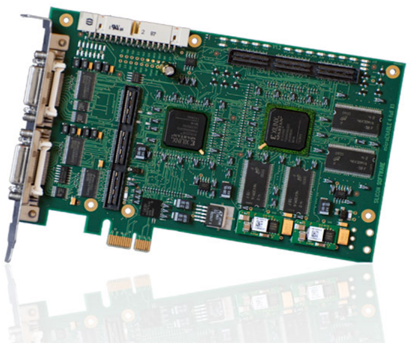
$$\sigma_{x,y} \approx 1.4\text{Pixels} \tag{6.1}$$

This value is used in the Matlab simulation as width of the generated signals.

6.4 Results of the FPGA Implementation

This section summarizes all measurements using the final hardware implementation of the Estimator algorithm, further called Hardware Estimator, or simply HWEstimator. In order to evaluate the performance and localization accuracy, the images of the Matlab simulation described above are stored and are reprocessed by the FPGA-based implementation. Additionally, the performance of the hardware with real data is tested. For this reason, the linewidth measurement of section 6.4.3 is repeated using the hardware implementation.

In the remainder of this chapter the results of the timing measurements using the FPGA implementation are presented. Additionally, the final speedup compared to the Matlab implementations is shown.


(a) Frame grabber card: microEnable IV VD1 - CL

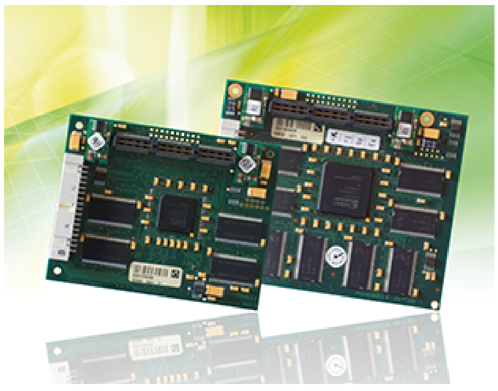

(b) Pluggabble subboards Pixelplant PX100 and the larger PX200

Figure 6.12: Frame Grabber cards from Silicon Software used to implement the Hardware Estimator. Source: Silicon Software.

6.4.1 Resource Utilization and Properties of the FPGA-Card

The FPGA that is utilized to implement the HWEstimtator on the frame grabber microEnable4 VD1-CL is a Xilinx XC3S 1600E FPGA. On the additional PixelPlant PX200 is a Xilinx XC3S 4000 FPGA installed. The characteristics of these FPGAs are listed in table 6.2. As it can be seen, the PixelPlant-FPGA provides about twice as many logic resources as the base-card. The FPGAs support clock-rates up to over 300 MHz. In order to meet all constraints of VisualApplets the clock-rate of the FPGAs is limited by Silicon Software to 63 MHz. Since the clock of the FPGA defines the pixel-clock the theoretical throughput is 63 Million Pixels per second or a capacity of 126 MB/s, assuming 16 bits per Pixel.

BOARD	FPGA	Logic Cells	CLBs	Slices	BRAM	Dedicated Multipliers
mE4 VD1	XC3S 1600E	33192	3688	14752	648K	36
PX200	XC3S 4000	62208	6912	27648	1728K	96

Table 6.1: Characteristics of the utilized FPGAs

CLK 63 MHz	VD1	PX200
Flip-Flops	15249 (51%)	23429 (42%)
Logic	21425 (72%)	24301 (43%)
B-Ram	36 (100%)	33 (34%)

Table 6.2: Total hardware usage of the hardware implementation.

The final implementation of the Hardware Estimator is distributed on the two FPGAs of the two cards. The total hardware usage is listed in table 6.2. As the table shows, the logic of the algorithm also is distributed equally on the two FPGAs. So the algorithm is basically divided in two parts.

The one part that contains the background handling and the spot detection and extraction is implemented on the base card. The other part contains the whole logic for the calculations on the extracted ROI and is located on the PixelPlant.
The filling ratio of the FPGA of the base card mE4 VD1-CL is very high: 72% of the logic elements and even 100% of the intern Block-RAM cells are used. The high value for the Block-RAM usage has its origin in the buffers for synchronization. These buffers are designed as large as possible to realize a higher number of detectable spots per image. Furthermore, this guarantees a more stable execution of the algorithm when larger numbers of signals are detected.

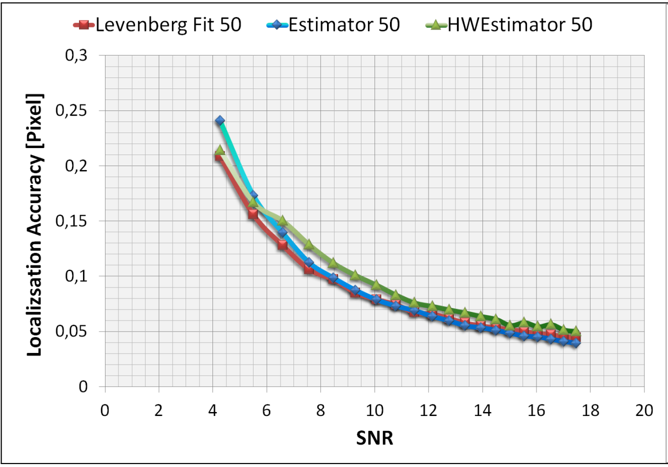
The FPGA of the PixelPlant offers much more resources than the FPGA installed on the base card. It is filled only to a degree of about 43%. Typically, each degree below 80% describes a comfortable level to ensure a successful building process. Nevertheless, the execution of the placing and routing routine often failed because timing constrains were not met. It did not became clear why this happened in about 90% of the attempts. A reason for the failure could be the very high number of operators (Overall 184) of the VisualApplets design. However, it succeeded finally and the design can successfully be used.

6.4.2 Results of the Hardware Using the Data of the Matlab Simulation

The localization accuracy of the Hardware Estimator is determined using simulated data. The pictures that are used for this measurement are the images from the Matlab simulation.

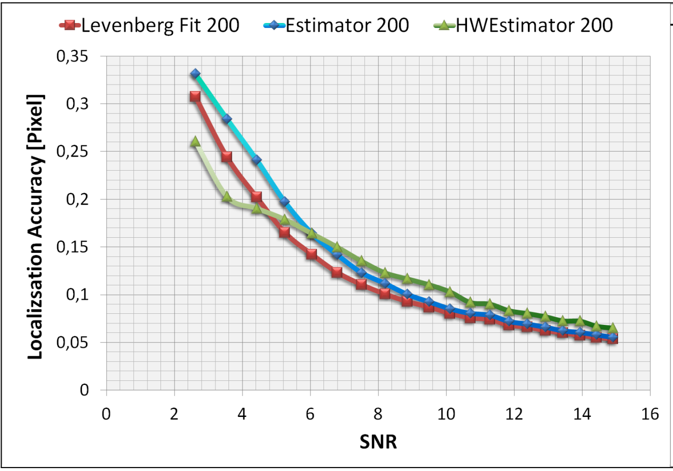
Simulation with Fixed Background Intensity

The results of the performance of the Hardware Estimator are presented in Figures 6.13(a)-6.13(c). As the diagrams show, the localization accuracy of the FPGA-implementation is almost as high as the accuracy of the Matlab implementation or the Levenberg fit method. In each diagram the localization accuracy passes the limit of 1/10 pixels at an SNR of 10.



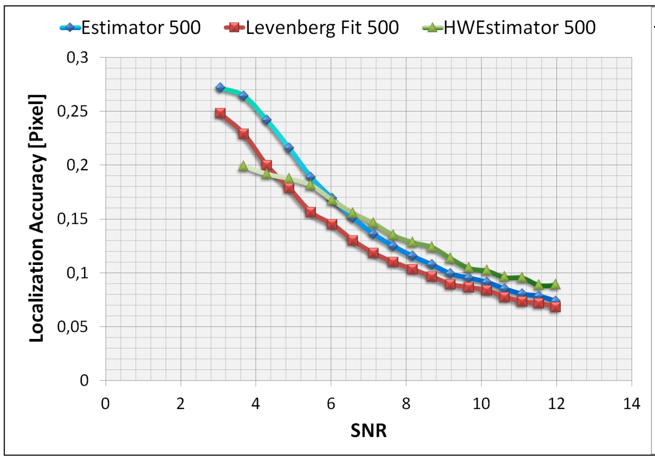
SNR	HWEstim.	Estimator	Levenb.
4,265	0,214	0,244	0,209
5,499	0,167	0,175	0,156
6,587	0,150	0,138	0,128
7,567	0,129	0,113	0,106
8,464	0,112	0,101	0,097
9,293	0,101	0,087	0,085
10,067	0,092	0,079	0,079
10,794	0,083	0,073	0,074
11,483	0,076	0,066	0,067
12,138	0,073	0,064	0,066
12,763	0,070	0,059	0,062
13,361	0,067	0,056	0,058
13,937	0,064	0,053	0,055
14,492	0,061	0,049	0,053
15,028	0,056	0,048	0,052
15,546	0,058	0,046	0,050
16,049	0,054	0,045	0,048
16,538	0,056	0,044	0,047
17,013	0,051	0,042	0,045
17,477	0,050	0,041	0,045

(a) Background intensity 50



SNR	HWEstim.	Estimator	Levenb.
2,617	0,301	0,252	0,266
3,547	0,202	0,274	0,231
4,422	0,192	0,242	0,204
5,251	0,182	0,199	0,164
6,039	0,163	0,165	0,142
6,790	0,153	0,138	0,122
7,508	0,137	0,123	0,111
8,197	0,122	0,112	0,103
8,859	0,113	0,101	0,093
9,497	0,107	0,096	0,088
10,112	0,100	0,084	0,080
10,708	0,094	0,080	0,076
11,284	0,092	0,077	0,071
11,844	0,085	0,071	0,068
12,387	0,080	0,069	0,066
12,916	0,078	0,065	0,062
13,430	0,079	0,060	0,059
13,932	0,073	0,059	0,057
14,422	0,067	0,057	0,055
14,900	0,065	0,055	0,053

(b) Background intensity 200



SNR	HWEstim.	Estimator	Levenb.
3,053	0,199	0,272	0,248
3,679	0,207	0,264	0,229
4,290	0,195	0,242	0,200
4,885	0,187	0,216	0,179
5,465	0,179	0,189	0,156
6,032	0,170	0,169	0,145
6,585	0,157	0,151	0,130
7,127	0,143	0,136	0,118
7,656	0,134	0,125	0,110
8,175	0,128	0,115	0,103
8,683	0,123	0,108	0,096
9,181	0,114	0,099	0,089
9,670	0,107	0,095	0,087
10,149	0,104	0,091	0,084
10,619	0,097	0,085	0,077
11,082	0,093	0,081	0,073
11,536	0,090	0,079	0,072
11,982	0,088	0,074	0,068

(c) Background intensity 500

Figure 6.13: Localization Accuracy of the hardware implementation over SNR for different background intensities compared to the software implementation and the iterative Levenberg fit. Width of generated signals = 1.4 Pixel

Evaluation of the Error

The evaluation of the error is done as described in section 6.2.3. The same generated signals are analyzed. The mean value of the calculated error values is compared to the real localization accuracy of the measurement in diagram 6.13(a). These two values are shown in figure 6.14 over several SNRs.

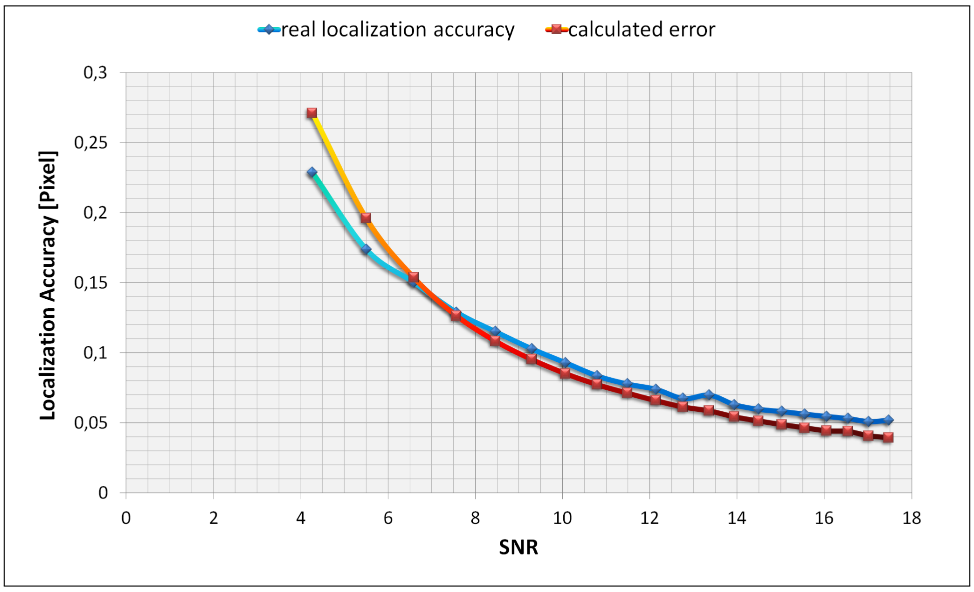


Figure 6.14: Mean value of the calculated error returned by the hardware implementation compared to the mean real localization accuracy. The background intensity is 50. Signal width = 1.4 Pixel

Compared to the values of the Matlab version (Fig. 6.4), the error returned by the Hardware Estimator is closer to the real values, than the calculated value from the Matlab version.

Evaluation of the Calculated Width

As the Matlab implementation, the hardware implementation also returns the width of the detected spots. As it can be seen in figure 6.15, the width returned by the Hardware Estimator is smaller than the real width of 1.4 pixels.

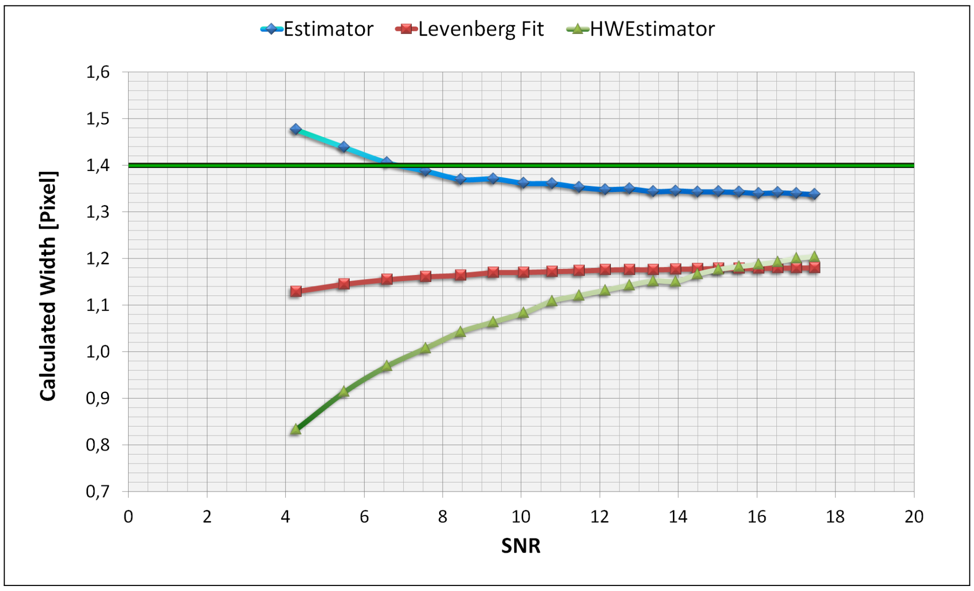


Figure 6.15: Calculated width returned by the hardware implementation compared to other methods. Real width of the created spots is 1.4 pixels.

Although the equations for calculating the width are the same, the Matlab implementation provides much better values for the width of the detected spots. This difference occurs because the Matlab implementation runs a second time over the ROI and calculates the width before the ROI is processed by the `cluster-separator` and the `cutoff`. This is not realized in hardware. In order to save resources, calculations of the parameters are implemented only once, because the second run over the ROI requires a complete second implementation of the calculations for the width - but as far as the precise value of the width is not needed, this implementation will be satisfying.

6.4.3 Results of the Hardware Using Real Data

In order to give an impression of the localization accuracy of the Hardware Estimator using real data, the linewidth measurement described in section 6.3 is repeated. Also, the same section containing the straight structure is processed by the FPGA-card in this measurement. It has to be remarked that the measurement itself influences the result, because it is slightly complicated to measure the structure in exactly the same direction. This error is reduced by averaging over the entire length of the structure.

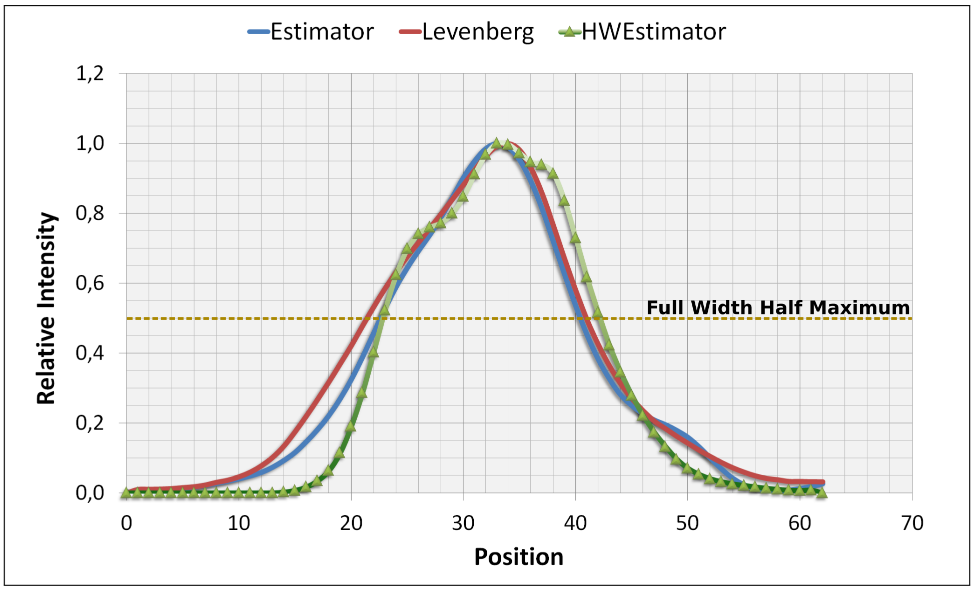


Figure 6.16: Result of the linewidth measurement using the FPGA-card. The intensity values are normed to the maximum intensity. The width is measured at the half of the maximum. Width Estimator: 17.75 px, Levenberg: 19.43 px, HWEstimator: 19.41 px

As it can be seen in figure 6.16, the performance of the Hardware Estimator is well comparable to the other methods. The stated values of the width are measured in pixels. The analyzed image is created using pixels that are equal to 5nm distance. So the linewidths can be transformed to:

- Estimator: 88.75nm
- Levenberg: 97.15nm
- HWEstimator: 97.05nm

The measured widths of the structure are much wider than the localization accuracy of a single spot. So, this measurement gives only an idea of the real performance of the single methods. Unfortunately, no record of a thinner structure was available.

6.4.4 Analysis of the Execution Time

The measurement of the computation speed of the hardware implementation is done using the same image stack that has been used already for the time measurements using the Matlab implementation of the Estimator (Sec. 6.2.5).

Since the execution time in hardware cannot be measured for each step of the algorithm, only the total processing time for processing the entire image stack can be compared. The final result of this measurement can be analyzed in figure 6.17. As the diagram shows, the hardware implementation provides a further speedup of the image processing by a factor of **16**. This means an acceleration of the calculations compared to the common iterative fit methods of a factor of over **800**. However, if the acceleration of the data processing is compared with respect to the algorithm that was available before this thesis was started, the total acceleration is more than a factor of **4000**. Furthermore, if the total computing speed of the MaLiang method from section 3.2.2 is transfered to the calculation of 366000 signals, the final execution time is still 2 times of the execution time of the Estimator algorithm. So the Hardware Estimator even beats the fastest algorithm in localization microscopy presented so far.

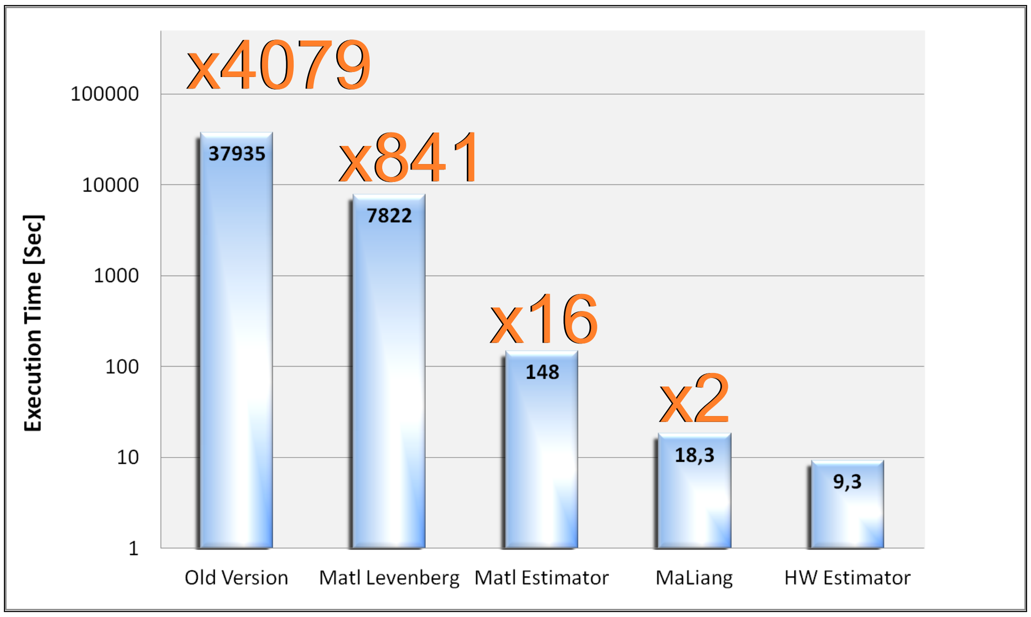


Figure 6.17: Speedup of the hardware implementation of the Estimator compared to the implementation in Matlab, the Matlab implementation with the parameter estimation replaced by the iterative Levenberg method and the former Algorithm that defines the baseline for the acceleration. In the measurement a data stack of 2000 images, having a resolution of 288x320, are processed.

Algorithm	Old Matlab Alg.	Matl. Levenb.[1]	Matl. Estim.	MaLiang Alg.[2]	HWEstimator
Execution Time	37935 Sec	7855	148	18.3	9.3
Detected Signals	113077	366915	367914	366000	365166

The image data that were used for these measurements are representative for common image stacks and contain a lot of signals. Thus, the results of this measurement are qualified to be generalized. This means that the analysis of localization image data has been accelerated

[1] *Matlab Levenberg* contains all the processing steps of the Estimator algorithm except for the parameter calculation, that is replaced by the Levenberg method.

[2] Values for *MaLiang Algorithm* are calculated valued. No Implementation was available.

by the hardware implementation to about 10 seconds. Looking at the recording time for the images of about 2 minutes, this FPGA-based implementation already realizes the data analysis in real time.

Timing Analysis of the Single Tasks

The tasks, that are executed during the measurement are the loading of the images from the hard disk, the sending of the images by PCI-express to the frame grabber card, the processing of the images by the FPGA, and the receiving and storing of the data that are returned.

As it can be seen in figure 6.18, the total time required for the execution depends strongly on the count of spots that are detected in an image. In all measurements of figure 6.18 the same image stack with 288×320 pixels is processed. In order to vary the count of detected spots, only the threshold is changed.

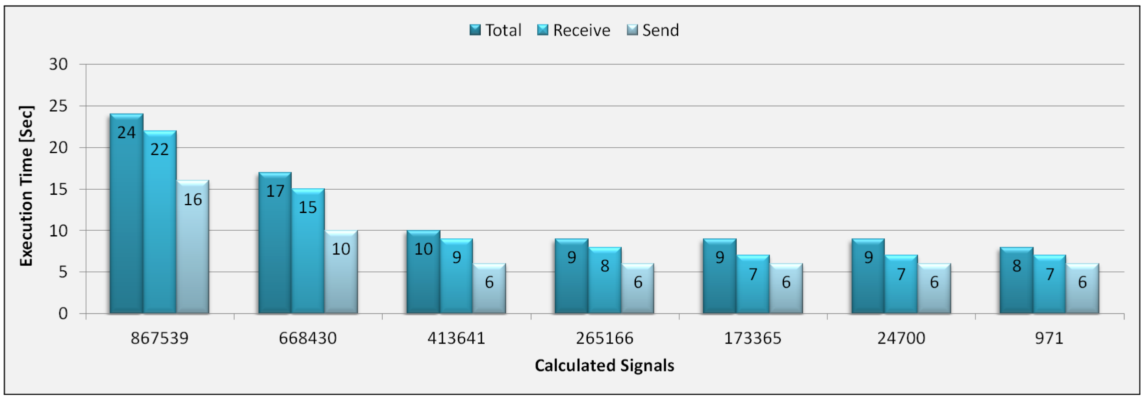


Figure 6.18: Execution times of different tasks of the hardware implementation. The time required for the send- and the receive thread compared to the total execution time are shown for different counts of found signals. The signal count is varied by changing the threshold. Values are rounded.

The diagram shows that the time required for sending the data to the frame grabber card is independent from the amount of found signals up to a certain limit. Besides the sending of the data, the send thread also reads the image data from the hard disk. Since the data amount that has to be read and sent is always the same, the duration should be independently from the number of signals that are found. However, if the number of found signals exceeds a certain limit the execution time of the send thread increases strongly. This occurs because the input stream has to be throttled in order to avoid an overflow of the buffers in the FPGA.

Furthermore, the total execution time mainly depends on the time it takes for storing the data. With an increase of detected signals the amount of data that has to be stored on the hard disk also increases. The storage of the results affects strongly the execution time what means a further decrease of the processing time can be realized by installing a faster hard disk e.g. a fast solid state disk (SSD).

Considering the total data amount of 2000 image having a dimension of 288x320 pixels and 2 bytes per pixel, 368.64 MByte are sent to the frame grabber. Regarding the measured time of 6 ± 0.3 seconds that is required to transmit the data, the total data rate can be determined to 61.44 ± 3 MByte per second. This data rate defines a realistic value for the reading capacity of the installed hard disk.

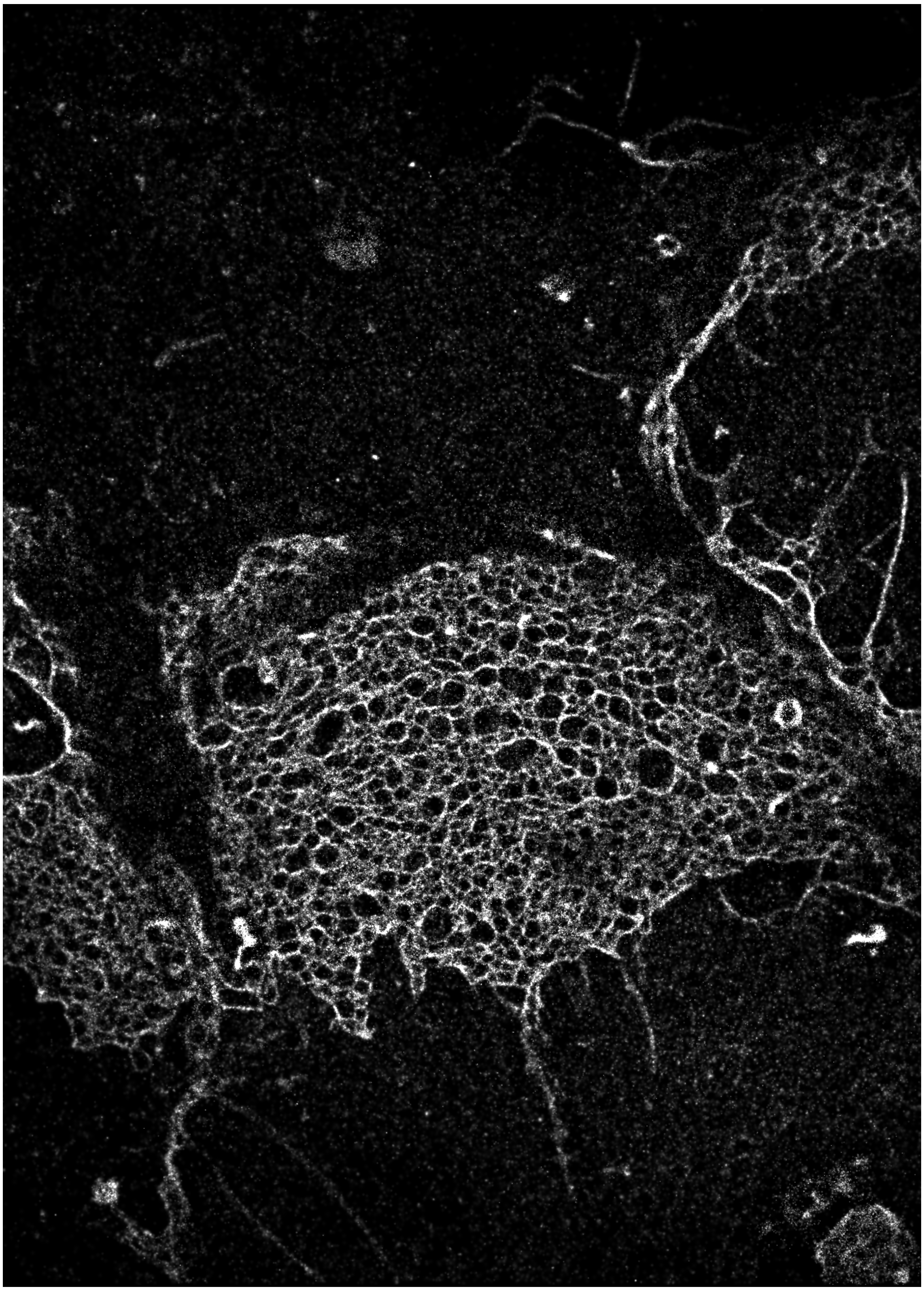


Figure 6.19: Localiztion image of a cell membrane, generated from the data stack that was used for the timing analysis in section 6.2.5 and 6.4.4. The computations took 9 seconds using the Hardware Estimator. 265166 signals were detected. Source: [40].

Chapter 7

Conclusion

The data summarized in the result chapter represent extraordinary good results. The algorithm that was developed in this thesis achieves and also outperforms the requirements that were claimed before. The new approaches for the noise and background handling that were developed in order to reach fast execution speed and hardware compatibility, show very good characteristics to provide high localization accuracy of the Maximum-Likelihood Estimator. Further, the method of maximum likelihood provides excellent characteristics to find the precise position of a molecule with sub-pixel accuracy. As it turns out, the Estimator algorithm even outperforms the localization accuracy of iterative least squares methods for low background intensities. This was hardly expected for this new approach that was basically developed to accelerate the existing data processing.

This thesis basically illustrates two key achievements:
First it shows that a non-iterative, analytical algorithm can provide as good results as the iterative least squares fit method but with far less calculation effort. This could not be expected before, because normally there is always a compromise between calculation speed and accuracy in solving mathematical problems. This can also be seen at the implementation of the analytical method FluoroBancroft that was implemented to compare the estimator to another non iterative method. But as it turned out, the implementation of the FluoroBancroft missed the expectations and was therefore not analyzed further. In addition, this thesis proves that a large pipeline, which has been realized in an FPGA is able to perform very special calculations as they exist in the image processing. Generally, the idea of a pipeline in hardware is to let data-streams like image-streams propagate value by value through the calculations. Consequently, in every clock cycle one value enters the pipeline and each pipeline stage executes its task simultaneously. So normally the amount of data that enters the pipeline is equal to the amount of data that is returned by the implementation.

By the distribution of the algorithm into two parts (one for the calculations on the whole images and the other part for the calculations on the smaller sections, the ROIs) it became possible to extract very special data from small sections of a large two-dimensional image what changes this balance of in and output. Usually, this would mean a larger synchronization problem but the tool VisualApplets of Silicon Software realizes the synchronization of the different pipeline streams perfectly.

The results of the timing measurements reveal that the goal of real-time image-processing is already reached by the Matlab implementation of the Estimator. However, this does not mean that the further increase in performance offered by the hardware implementation will not be needed. Future fluorophores and high performance cameras will permit much higher frame rates and data volumes that will make the application of hardware-accelerated systems unavoidable. Furthermore, it was demonstrated that it is already possible today to realize real-time data-processing, both with higher frame-rates and larger images. So, the calculation speed of the Matlab implementation of the Estimator is already high enough to support localization microscopy with a tool that provides high-speed image-processing for fast data analysis and that can also be used on any workstation PC running Matlab. Further, the final implementation on the Estimator shows that the challenges of the next generation cameras that will provide much higher frame rates and larger images, can be solved today. The measurement shows that the Hardware Estimator even outperformes the fastest algorithm in this field published so far of a factor of about $\times 2$.

At the moment, the Matlab version of the Estimator is already frequently applied to analyze the acquired image data of the localization microscope in the Kirchhoff Institute of Physics in Heidelberg. Additionally, the formulas and basic ideas of the Estimator have been presented and have already been published on the hardware conference FPL in September 2011 [41].

Chapter 8

Perspective

In this chapter further improvements and possible additional features for the Estimator algorithm are described. Also, an outlook for the later application of the algorithm is given. Furthermore, this chapter summarizes a few features that could further improve the performance of practivability of the Matlab resp. the hardware implementation of the Estimator algorithm.

8.1 Special Hardware Support in Matlab

The later versions of Matlab provide support for the Matlab Parallel Computing Toolbox$^{\text{TM}}$ [30], a special library for hardware supported computations. This library contains functions that support multi-threaded computing on multiple CPU cores, computer clusters or even distributed computing in a computer cloud as well as support for calculations on NVIDIA graphics cards resp. GPUs. Since the focus of this thesis is on the implementation of the FPGA-based algorithm, the acceleration of the Matlab-version by special hardware usage could not be further analyzed. However, an additional increase of the calculation speed of the Matlab-version can be expected; especially by executing the calculations on the ROIs in parallel.

8.2 Tools and Plug-ins for Existing Data Processing Programs

In order to achieve a more widespread usage of the fast algorithm, it would surely be nice to develop plug-ins that implement the Estimator algorithm for existing programs. For instance, the program ImageJ is a widespread program to analyze image data that provides support for additional self-made plug-ins. The program is able to read image data stacks, which are used in localization microscopy. As soon as these data stacks are available in the program it was surely easy realizable to create an own plug-in that implements the Estimator algorithm for fast data processing within the ImageJ-image viewer. It was also thinkable to use the ImageJ-viewer as interface of the hardware implementation. In addition to the ImageJ viewer, several other programs used for image acquisition in microscopy provide interfaces to integrate self-written code. Thus, there is a large range of programs that could be extended by the fast Estimator algorithm.

8.3 Integration of the Estimator in the Existing Data Acquisition Software

The next step to improve the data analysis of localization microscopy would possibly be an integration of the new algorithm into the actual image acquisition software. This would provide analysis of the data at the moment it is available on the computer without storing the data on the hard drive. Thereby, it would be possible to generate localization microscopy images within few minutes without any interaction of the user. It would only need one push of a button to create images of biological structures with a resolution that exceeds the diffraction limit. But this integration would mean a major investigation of the existing program code and might mean a redesign of the entire program, what would clearly exceed the limits of this thesis.

One can see that there is still much work to improve the data analysis of localization microscopy and thereby, to achieve a higher practicability of this great technique. It has been very interesting to get introduced in this recent field of science and it will remain very exciting to follow the future development of this high-resolution visible-light microscopy method and what further progress can be achieved thereby.

Appendix A

Code Sections

A.1 Loop Implementation of the Background Handler

In section 6.2.5 the execution times of the original background handler and a second implementation containing for-loops to realize the same calculations are compared. The code of the second implementation is listed below:

```
function [diffimg,meanbg,newBM]=slowbackground(oldBM,img,BGF)

  meanbg = round(mean(mean(img)));
  cutoff = sqrt(meanbg);

  [SizeY SizeX] = size(img);
  diffimg = zeros(SizeY,SizeX);

  for x = 1:SizeY
    for y = 1:SizeX

      diffimg(y,x) = img(y,x) - oldBM(y,x);
      newBM(y,x) = oldBM(y,x)+min(diffimg(y,x),cutoff)/BGF;

      if(diffimg(y,x) < 0)
          diffimg(y,x) = 0;
      end

    end
  end

end
```

Listing A.1: Matlab code for the background handling function in loop-style:

For every image the nested loop construction runs once over every pixel of the image and calculates the difference image Also, the new background map is generated and at last all negative values of the difference image are set to zero.

Appendix B

Setting up a Visual C++ Frame-grabber Project

In future the requirements of the frame grabber can change. So if the software for the frame grabber has to be adapted, this chapter gives an instruction how to create a new project in Microsoft's Visual C++.

Following program installations are required:

Silicon Software Runtime 5.1 [1]

Microsoft Visual C++ 2008 Express or later

Tiff-library LibTiff

All software can be downloaded for free.

To create a compilable project in Visual C++ all required files and libraries have to be included. If the basic program code (**HWEstimator.cpp** and textbfHWEstimator.hpp) already exists, the option `create Project from existing files` can be used. The new project has to be created as a **console project**.

After specifying the workspace folder and the name for the project the additional include folder containing the header files has to be defined. The include folder contains all header files from Silicon Software and from the tiff-library.

Required Header Files

To compile the program successfully several header files (.h) are required. The following files have to be included:

B.0.1 Required Libraries

The next step is to include the precompiled libraries. Here also the libraries provided by Silicon Software and the tiff library are needed. These library files also have to be added to the project: The easiest way to include these libraries is to simply add the .lib files to the project.

[1]32bit and 64bit windows support. Software is downloadable on the Silicon Software homepage for free.

msinttypres\inttypes.h	**fgrab_define.h**	**clser.h**
msinttypres\stdint.h	**fgrab_prototyp.h**	**tiff.h**
gbe.h	**fgrab_struct.h**	**tiffio.h**
gbe_error.h	**sisoboards.h**	**tiffvers.h**
os_funcs.h	**SisoDisplay.h**	**tiffconf.h**
os_type.h	**sisoIo.h**	

clserme3.lib	**gbelib.lib**
clserme4.lib	**iolibrt.lib**
display_lib.lib	**libtiff.lib**
fglib5.lib	

B.0.2 Own Sources and Classes

Several classes to deal with the different data structures were developed. The `CTimer` class is only required for the timing measurements presented in the results chapter. These files have also to be added to the project:

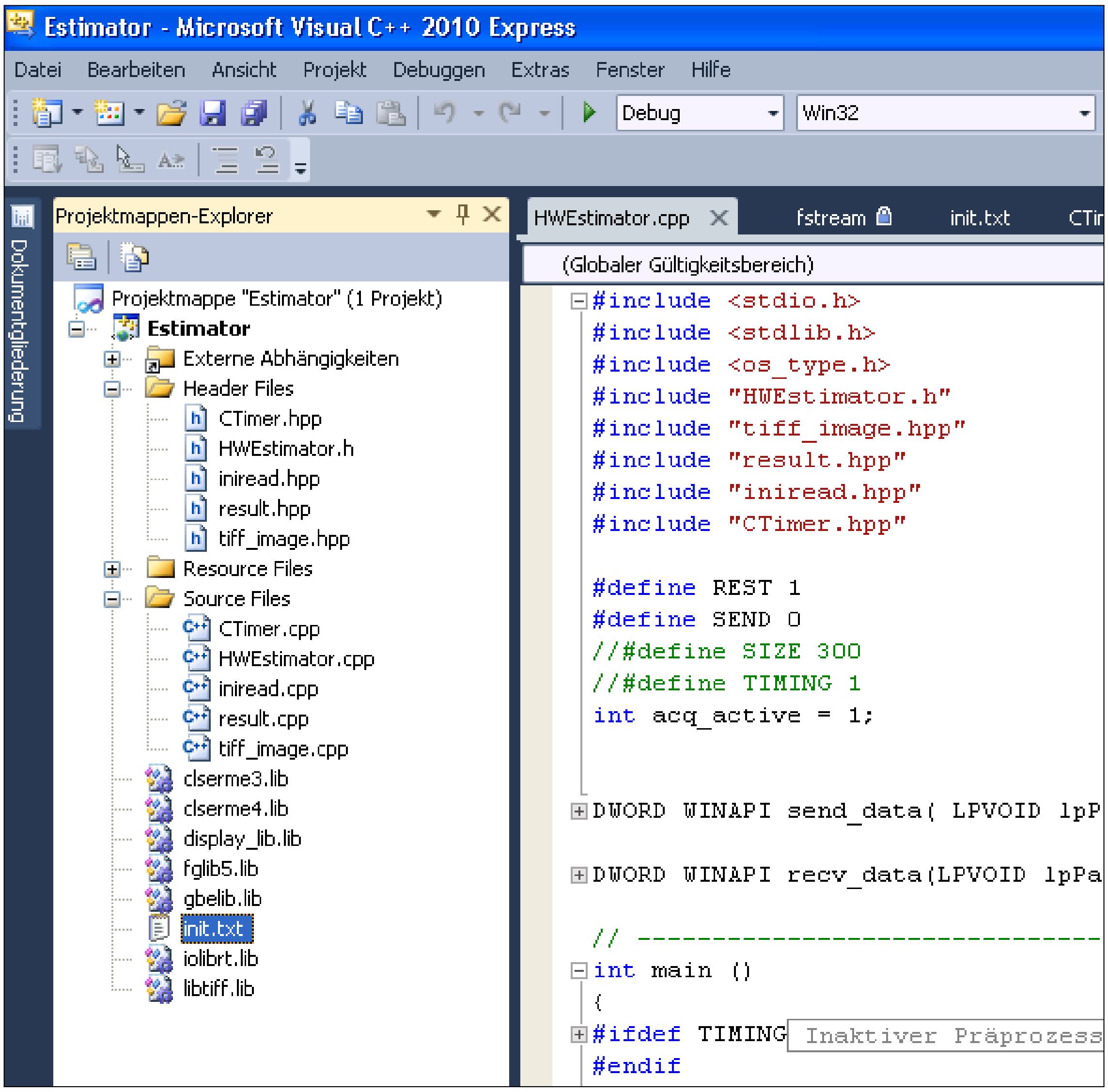
tiff_image.cpp/.hpp

result.cpp/.hpp

CTimer.cpp/.hpp

iniread.cpp/.hpp

The final setup of the project is shown in figure B.1.



```
#include <stdio.h>
#include <stdlib.h>
#include <os_type.h>
#include "HWEstimator.h"
#include "tiff_image.hpp"
#include "result.hpp"
#include "iniread.hpp"
#include "CTimer.hpp"

#define REST 1
#define SEND 0
//#define SIZE 300
//#define TIMING 1
int acq_active = 1;

DWORD WINAPI send_data( LPVOID lpP

DWORD WINAPI recv_data(LPVOID lpPa

// --------------------------------
int main ()
{
#ifdef TIMING Inaktiver Präprozess
#endif
```

Figure B.1: Final setup for compiling the Estimator. Additionally, the directory containing the header files of the Silicon Software driver functions has also to be added to the include folders of the project.

Appendix C

Setting up a Visual C++ Frame-grabber Project for usage in Matlab

Matlab provides certain functions to access c++ code by matlab functions. This can be done with the implementation of a `mex_function()`.

The `mex_function()` has to be compiled in Windows as a dynamic linkable library (dll). Using this dll-file, Matlab can access the function and execute the c code.

The translation of the Matlab arrays into c++ data array is realized by the functions `mxGetPr()` and `mxCreateDoubleMatrix()`.

`mxGetPr()` returns a pointer to the data in an existing matlab array. This function is used to read the image data from Matlab into a common integer array, that is further passed to the frame grabber. Additional functions are available that read the dimensions of the Matlab array containing the images.

The function `mxCreateDoubleMatrix()` creates a new Matlab array, that is, for instance, filled with the data that is returned by the controller code of the Hardware Estimator.

The Mex-Function has a defined structure, as shown in listing C.1. In addition to the include files of the Silicon Software environment and the Hardware Estimator that are listed in appendixB the header files of the Matlab environment must also be included. They can be found in the Matlab tree.

Listing C.1: C++ code for the mexFunction.cpp:

```
---------------------------------------------------------------

#include "mexFunction.h"
#include "vasFgProject.h"
#include "HWEstimator.hpp"
#pragma comment(lib, "libmx.lib")
#pragma comment(lib, "libmat.lib")
#pragma comment(lib, "libmex.lib")
#define IMG_IN prhs[0]
#define PXS_IN prhs[1]
#define TH_IN prhs[2]
#define CUTOFF_IN prhs[3]

#define ORTE_OUT plhs[0]
```

```
void mexFunction( int nlhs ,mxArray* plhs[] ,
                  int nrhs , mxArray *prhs[] )
-------------------------------------------------------------
```

To achieve a working compiling process, further adjustments in visual C++ are required:

1. Generate a new empty Win32 as DLL application and select *Export Symbols*
2. Create a cpp file *mexFunction.cpp* and a *mexFunction.h* header file.
3. Add include directories for Matlab, SiliconSoftware and HWEstimator files.
4. Add the *Preprocessor Definition: MATLAB_MEX_FILE*
5. Select for *Code Generation* the *Multi-threaded Debug* runtime library
6. Add the library directories of Matlab and Silicon Software and load also the .lib files of Silicon Software and the libtiff into the Project as described in appendix B.
7. Add the module definition file *mecFunction.def* to the project.

Listing C.2: mexFunction.def:

```
-------------------------------------------------------------
LIBRARY         "HWEstimator"
EXPORTS
    mexFunction
-------------------------------------------------------------
```

Listing C.3: mexFunction.h:

```
-------------------------------------------------------------

#include "matrix.h"
#include "mex.h"

#define MEX_FUNCTION_EXPORTS
#ifdef MEX_FUNCTION_EXPORTS
#define MEX_FUNCTION_API __declspec(dllexport)
#else
#define MEX_FUNCTION_API __declspec(dllimport)
#endif

MEX_FUNCTION_API void mexFunction(int nlhs, mxArray* plhs[],
                                  int nrhs, mxArray* prhs[]);
-------------------------------------------------------------
```

The final program is compiled into the dll and can be accessed from Matlab by an empty function that is named with the same name as the dll that must be stored in the same directory.

Another tutorial to generate a mexFunction for Matlab can be found in [42].

Appendix D

Tools

D.1 KdftoTiff

The KDF image format is an old image format that is still used in the work group of Prof. Cremer from the university of Heidelberg, to store all localization image data. Since this format is not supported by most of the image analysis tools, a small program was developed to convert KDF-image stacks into the more convenient tif-image format.

`KdftoTiff` can simply be accessed by right-clicking on a file and open it with the executable. The final tif-image stack then is stored with the same name in the folder containing the original KDF image source. Alternatively the program can be accessed from the console. There it is possible to define an alternative out put folder and new filename by an optional second input argument:

```
$ ./KdftoTif ~/data/03ls_a.kdf
or
$ ./KdftoTif ~/data/03ls_a.kdf ./newTiff.tif
```

To execute the program in Microsoft Windows XP 32bit several .dll files are required:

display_lib.dll

jpeg62.dll

msvcp100.dll

msvcr100.dll

libtiff3.dll

These files have to be stored in the folder containing also the executable `KdftoTiff.exe` or in the `Windows\System32` folder.

Appendix E

Calculations of the Formulas for the Estimator

The likelihood function of a two dimensional Gaussian distribution without background factors is defined by:

$$L=\prod_{j=1}^{N}\frac{A}{2\pi\sigma_x\sigma_y}exp\left(-\frac{1}{2}\left(\frac{x^{(j)}-\mu_x}{\sigma_x}\right)^2-\frac{1}{2}\left(\frac{y^{(j)}-\mu_y}{\sigma_y}\right)^2\right) \tag{E.1}$$

After application of the logarithm on this function the product turns into a sum what is more convenient for the later calculations:

$$l=-\frac{1}{2}\sum_{j=1}^{N}\left(\frac{(x^{(j)}-\mu_x)^2}{\sigma_x^2}+\frac{(y^{(j)}-\mu_y)^2}{\sigma_y^2}\right)-N\ln\sigma_x-N\ln\sigma_y-const \tag{E.2}$$

With this function the system of likelihood equations becomes to:

$$\frac{\partial l}{\partial\mu_x}=\sum_{j=1}^{N}\frac{(x^{(j)}-\mu_x)^2}{\sigma_x^2}, \tag{E.3}$$

$$\frac{\partial l}{\partial\sigma_x}=\frac{1}{\sigma_x^3}\sum_{j=1}^{N}(x^{(j)}-\mu_x)^2-\frac{N}{\sigma_x} \tag{E.4}$$

$$\frac{\partial l}{\partial\mu_y}=\sum_{j=1}^{N}\frac{(y^{(j)}-\mu_y)^2}{\sigma_y^2}, \tag{E.5}$$

$$\frac{\partial l}{\partial\sigma_y}=\frac{1}{\sigma_y^3}\sum_{j=1}^{N}(y^{(j)}-\mu_y)^2-\frac{N}{\sigma_y} \tag{E.6}$$

These four equations can be separated into two independent systems of linear equations. Each system can be solved and the resulting estimators for the parameters are the equations used for the Estimator algorithm:

$$\mu_x=\frac{1}{N}\sum_{j=1}^{N}x^{(j)} \qquad \mu_y=\frac{1}{N}\sum_{j=1}^{N}y^{(j)} \tag{E.7}$$

$$\sigma_x = \sqrt{\frac{1}{N}\sum_{j=1}^{N}(x^{(j)}-\mu_x)^2} \qquad \sigma_y = \sqrt{\frac{1}{N}\sum_{j=1}^{N}(y^{(j)}-\mu_y)^2} \tag{E.8}$$

To gain the error of the estimated parameters the normal matrix has to be determined. The elements of the normal matrix are calculated by:

$$N_{ij} = \frac{\partial^2 l}{\partial a_i \partial a_j} \qquad \text{with} \qquad a_0 = \mu_x, a_1 = \mu_y, a_2 = \sigma_x, a_3 = \sigma_y \tag{E.9}$$

For the given problem of a two dimensional Gaussian distribution it has following shape:

$$\mathbf{N} = \sum_{j=1}^{N} \begin{pmatrix} -\frac{1}{\sigma_x^2} & 0 & \frac{2(x^{(j)}-\mu_x)}{\sigma_x^3} & 0 \\ 0 & -\frac{1}{\sigma_y^2} & 0 & \frac{2(y^{(j)}-\mu_y)}{\sigma_y^3} \\ \frac{2(x^{(j)}-\mu_x)}{\sigma_x^3} & 0 & \frac{1}{\sigma_x^2} - \frac{3(x^{(j)}-\mu_x)^2}{\sigma_x^4} & 0 \\ 0 & \frac{2(y^{(j)}-\mu_y)}{\sigma_y^3} & 0 & \frac{1}{\sigma_y^2} - \frac{3(y^{(j)}-\mu_y)^2}{\sigma_y^4} \end{pmatrix} \tag{E.10}$$

By application of the following simplifications

$$\sum_{j=1}^{N} \frac{x^{(j)}-\mu_x}{\sigma_x^3} = \frac{N}{\sigma_x^3}\left(-\mu_x + \frac{1}{N}\sum_{j=1}^{N} x^{(j)}\right) = \frac{N}{\sigma_x^3}\left(-\mu_x + \mu_x\right) = 0 \tag{E.11}$$

and

$$\begin{aligned} \sum_{j=1}^{N} \frac{1}{\sigma_x^2} - \frac{3(x^{(j)}-\mu_x)^2}{\sigma_x^4} &= \frac{N}{\sigma_x^2} - \frac{3N}{\sigma_x^4}\left(\frac{1}{N}\sum_{j=1}^{N}(y^{(j)}-\mu_y)^2\right) \\ &= \frac{N}{\sigma_x^2} - \frac{3N}{\sigma_x^4}\left(\sigma_x^2\right) = -\frac{2N}{\sigma_x^2} \end{aligned} \tag{E.12}$$

the normal matrix finally becomes diagonal:

$$\mathbf{N} = \begin{pmatrix} -\frac{N}{\sigma_x^2} & 0 & 0 & 0 \\ 0 & -\frac{N}{\sigma_x^2} & 0 & 0 \\ 0 & 0 & -\frac{2N}{\sigma_x^2} & 0 \\ 0 & 0 & 0 & -\frac{2N}{\sigma_y^2} \end{pmatrix} \tag{E.13}$$

The covariance matrix of a diagonal matrix can easily be calculated. It is needed to get the errors of the parameters a_0-a_3.

$$\mathbf{C} = \begin{pmatrix} -\frac{\sigma_x^2}{N} & 0 & 0 & 0 \\ 0 & -\frac{\sigma_x^2}{N} & 0 & 0 \\ 0 & 0 & -\frac{2N}{\sigma_x^2} & 0 \\ 0 & 0 & 0 & -\frac{2N}{\sigma_y^2} \end{pmatrix} \tag{E.14}$$

The error is given by the square root of the variance, and so, the errors are defined by:

$$\Delta\mu_{x,y} = \frac{\sigma_{x,y}}{\sqrt{N}} \qquad \Delta\sigma_{x,y} = \frac{\sigma_{x,y}}{\sqrt{2N}} \tag{E.15}$$

This forumulas are the same as in the one dimensional case.

Appendix F

Erklärung (Statement)

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

December 18, 2011

Manfred Kirchgessner

Bibliography

[1] E. Abbe, “Beiträge zur theorie des mikroskops und der mikroskopischen wahrnehmung,” *Archiv für Mikroskopische Anatomie*, vol. 9, no. 1, pp. 413–418, Dec. 1873.

[2] L. De Broglie, “Recherches sur la théorie des quanta,” *Annales de Physique*, vol. 10, jan 1925.

[3] M. Knoll and E. Ruska, “Das elektronenmikroskop,” *Zeitschrift für Physik A Hadrons and Nuclei*, vol. 78, pp. 318–339, 1932, 10.1007/BF01342199. [Online]. Available: http://dx.doi.org/10.1007/BF01342199

[4] S. W. Hell and J. Wichmann, “Breaking the diffraction resolution limit by stimulated emission: stimulated-emission-depletion fluorescence microscopy,” *Opt. Lett.*, vol. 19, no. 11, pp. 780–782, Jun. 1994. [Online]. Available: http://dx.doi.org/10.1364/OL.19.000780

[5] *ALICE Technical design Report.* Cern, 2004, ch. 13.2.3 FPGA coprocessor/ detector interface.

[6] L. Rayleigh, “On the theory of optical images with special reference to the microscope,” *Philos Mag*, vol. 5, no. 42, pp. 167–195, 1896.

[7] M. Chalfie, Y. Tu, G. Euskirchen, W. W. Ward, and D. C. Prasher, “Green fluorescent protein as a marker for gene expression,” *Science*, vol. 263, no. 5148, pp. 802–805, Feb. 1994. [Online]. Available: http://dx.doi.org/10.1126/science.8303295

[8] R. Heim, A. B. Cubitt, and R. Y. Tsien, “Improved green fluorescence.” *Nature*, vol. 373, no. 6516, pp. 663–664, Feb. 1995. [Online]. Available: http://dx.doi.org/10.1038/373663b0

[9] J. Goedhart, L. van Weeren, M. A. Hink, N. O. Vischer, K. Jalink, and T. W. Gadella, “Bright cyan fluorescent protein variants identified by fluorescence lifetime screening.” *Nature methods*, vol. 7, no. 2, pp. 137–139, Feb. 2010. [Online]. Available: http://dx.doi.org/10.1038/nmeth.1415

[10] M. Minsky, “Microscopy apparatus,” US Patent 3013467, 1961.

[11] P. Brakenhoff; G. J. ; Blom, P.; Barends, “Confocal scanning light microscopy with aperture immersion lenses,” *J. Microsc.*, vol. 117, pp. 219–232, 1979.

[12] T. C. C. Cremer, “Considerations on a laser-scanning-microscope with high resolution and depth of field,” *M1CROSCOPICA ACTA*, vol. 81, no. 1, pp. 31–44, 1987.

[13] T. J. F. Nathan S. Claxton and M. W. Davidson, “Laser scanning confocal microscopy.”

[14] V. Westphal and S. W. Hell, “Nanoscale Resolution in the Focal Plane of an Optical Microscope,” *Physical Review Letters*, vol. 94, no. 14, 2005. [Online]. Available: http://dx.doi.org/10.1103/PhysRevLett.94.143903

[15] T. A. Klar and S. W. Hell, “Subdiffraction resolution in far-field fluorescence microscopy,” *OPTICS LETTERS*, vol. 24, no. 14, pp. 954–956, Jul 1999.

[16] K. I. Willig, “Sted microscopy in the visible range,” Ph.D. dissertation, Ruperto-Carola University of Heidelberg, 2006.

[17] P. Lemmer, M. Gunkel, D. Baddeley, R. Kaufmann, A. Urich, Y. Weiland, J. Reymann, P. Müller, M. Hausmann, and C. Cremer, “Spdm – light microscopy with single molecule resolution at the nanoscale,” *Applied Physics B*, vol. 93, p. 1–12, 2008.

[18] M. J. Rust, M. Bates, and X. Zhuang, “Sub-diffraction-limit imaging by stochastic optical reconstruction microscopy (STORM).” *Nature methods*, vol. 3, no. 10, pp. 793–795, Oct. 2006. [Online]. Available: http://dx.doi.org/10.1038/nmeth929

[19] E. Betzig, G. H. Patterson, R. Sougrat, O. W. Lindwasser, S. Olenych, J. S. Bonifacino, M. W. Davidson, J. Lippincott-Schwartz, and H. F. Hess, “Imaging Intracellular Fluorescent Proteins at Nanometer Resolution,” *Science*, vol. 313, no. 5793, pp. 1642–1645, Sep. 2006. [Online]. Available: http://dx.doi.org/10.1126/science.1127344

[20] R. Zondervan, F. Kulzer, S. B. Orlinskii, and M. Orrit, “Photoblinking of rhodamine 6g in poly(vinyl alcohol): Radical dark state formed through the triplet,” *J. Phys. Chem. A*, vol. 107, no. 35, pp. 6770–6776, 2003.

[21] J. Schuster, F. Cichos, and C. von Borczyskowski, “Blinking of single molecules in various environments,” *Optics and Spectroscopy*, vol. 98, no. 5, pp. 712–717, May 2005. [Online]. Available: http://dx.doi.org/10.1134/1.1929057

[22] J. Folling, M. Bossi, H. Bock, R. Medda, C. A. Wurm, B. Hein, S. Jakobs, C. Eggeling, and S. W. Hell, “Fluorescence nanoscopy by ground-state depletion and single-molecule return,” *Nature Methods*, vol. 5, no. 11, pp. 943–945, Sep. 2008. [Online]. Available: http://dx.doi.org/10.1038/nmeth.1257

[23] T. B. McAnaney, W. Z. andCamille F. E. Doe, N. Bhanji, S. Wakelin, D. S. Pearson, P. Abbyad, X. Shi, S. G. Boxer, and C. R. Bagshaw, “Protonation, photobleaching, and photoactivation of yellow fluorescent protein(yfp 10c): A unifying mechanism,” *Biochemistry*, vol. 44, no. 14, pp. 5510–5524, 2005.

[24] R. M. Dickson, A. B. Cubitt, R. Y. Tsien, and W. E. Moerner, “On/off blinking and switching behaviour of single molecules of green fluorescent protein.” *Nature*, vol. 388, no. 6640, pp. 355–358, Jul. 1997. [Online]. Available: http://dx.doi.org/10.1038/41048

[25] G. Willis and J. Watrous, “Mosfet transistor and method of fabrication,” United States Patent, Tech. Rep., Oct 1976.

[26] *Embedded Processor Block in Virtex-5 FPGAs*, Xilinx, Feb 2010.

[27] P. D.-I. W. H. Glauert, *VHDL tutorial*, Lehrstuhl für Rechnergestützten Schaltungsentwurf Universität Erlangen-Nürnberg, Paul-Gordan-Str. 5 91052 Erlangen, Aug 2011. [Online]. Available: http://www.vhdl-online.de/tutorial/

[28] D. K. Tala, *Verilog Tutorial*, http://www.asic-world.com, May 2011. [Online]. Available: http://www.asic-world.com

[29] "Matlab - the language of technical computing," website, Aug 2011. [Online]. Available: http://www.mathworks.de/products/matlab/

[30] "Parallel computing toolbox," website, Aug 2011. [Online]. Available: http://www.mathworks.com/products/parallel-computing

[31] P. D. S. Heinrich, "Vorlesung/Übungen numerische methoden in der physik ws 2010/2011," Oct 2010. [Online]. Available: http://itp.tugraz.at/LV/sormann/NumPhysik/Skriptum/kapitel4.pdf

[32] S. Brandt, *Data Analysis*, 3rd ed. Springer, 1999.

[33] Z. Shen and S. B. Andersson, "Precise localization of fluorescent probes without numerical fitting," in *Biomedical Imaging: From Nano to Macro, 2007.* 4th IEEE International Symposium on, april 2007, pp. 252–255.

[34] ——, "Bias and precision of the fluorobancroft algorithm for single particle localization in fluorescence microscopy," *Signal Processing, IEEE Transactions on*, vol. 59, no. 8, pp. 4041 –4046, aug. 2011.

[35] T. Quan, P. Li, f Long, s Zeng, Q. Luo, P. N. Hedde, G. U. Nienhaus, and Z.-L. Huang, "Ultrafast, high-precision image analysis for localization-based super resolution microscopy," *Optics Express*, vol. 18, no. 11, pp. 11 867–11 876, 2010.

[36] D. Baddelay, M. B. Cannel, and C. Soeller, "Visualization of localization microscopy data," *Microscopy and Microanalysis*, 2010.

[37] B. Delaunay, "Sur lá sphère vide," *Bulletin of Academy of Sciences of the USSR*, vol. 7, no. 6, pp. 793–800, 1934.

[38] *MEX-files Guide*, MathWorks, Oct 2011. [Online]. Available: http://www.mathworks.de/support/tech-notes/1600/1605.html

[39] *Intel® Core^{TM} i5-450M Processor*, 2010. [Online]. Available: http://ark.intel.com/products/49022/Intel-Core-i5-450M-Processor-%283M-cache-2_40-GHz%29

[40] R. Kaufmann, "Entwicklung quantitativer analysemethoden in der lokalisationsmikroskopie," Ph.D. dissertation, University of Heidelberg, 2011.

[41] F. Gruell, M. Kirchgessner, R. Kaufmann, M. Hausmann, and U. Kebschull, "Accelerating image analysis for localization microscopy with fpgas," in *Field Programmable Logic and Applications (FPL), 2011 International Conference on*, sept. 2011, pp. 1 –5.

[42] dip, "C++ with matlab tutorial," online lecture, Oct 2011. [Online]. Available: http://www.engineering.uiowa.edu/~dip/lecture/C++_with_Matlab.pdf