Sebastian Krell, BSc

Design and Implementation of Real-time Histogramming for LiDAR

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Telematics

submitted to

Graz University of Technology

Supervisor

Ass.Prof. Dipl.-Ing. Dr.techn. Christian Steger

Institute for Technical Informatics (ITI)

Advisor

Ass.Prof. Dipl.-Ing. Dr.techn. Christian Steger
Dr. Norbert Druml (Infineon Technologies Austria AG)

Graz, May 2018

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis dissertation.

Date

Signature

Kurzfassung

Light Detection and Ranging (LiDAR) ist ein bildgebendes Verfahren, welches eine dreidimensionale Repräsentation der Umgebung abbilden kann. In den letzten Jahren gewann es immer mehr an Beliebtheit, sowohl in der Unterhaltungs- als auch in der Industrieelektronik. Es deckt ein breites Spektrum an Anwendungsfällen ab und wird sowohl vom Boden als auch aus der Luft betrieben. Es können sowohl topographische Vermessung als auch Dokumentation von lokalen Szenerien erstellt werden. Die Signalanalyse als auch -verarbeitung wurden in den letzten Jahren so weit vorangetrieben, dass auch Eigenschaften wie Reflektivität oder Geschwindigkeit von Objekten klassifiziert werden können. Es können auch strukturelle Schwächen in Betongebäuden gefunden und analysiert werden. Durch den immer kleiner werdenden Formfaktor findet diese Technologie auch Einzug in die Robotik und den autonomen Fahrassistenzsystemen. Diese Bereiche benötigen auch Echtzeit Signalverarbeitung der gelieferten Daten, da Entscheidungsfindungen System- und Sicherheitskritisch sind.

Diese Masterarbeit präsentiert ein Echtzeit-Datenverarbeitungssystem inklusive Signalanalyse eines 1bit diskreten LiDAR Systems. Dies ist möglich unter der Verwendung der Xilinx Artix 7 FPGA Plattform. Hardware-beschleunigte Komponenten sind mit VHDL entwickelt und in einem Artix 7 FPGA umgesetzt. Diese Art der Hardwareintegration ermöglicht high-speed Datenverarbeitung. Ein integrierter MicroBlaze Mikroprozessor wird verwendet, um die entwickelten Komponenten zur Laufzeit zu steuern. Durch die Analyse der abgetasteten Lichtreflexionen können verschiedene Attribute wie relative Reflektivität, Pulsbreite und Distanz zum Objekt extrahiert werden. Die daraus resultierenden Daten werden danach mittels Ethernet und USB an eine PC Software übertragen. Dieses Hardware-beschleunigte System ermöglicht zusätzliche Informationsextraktion und gleichzeitige Datenreduktion bei gleichzeitiger Beibehaltung von höchstem Datendurchsatz.

In dieser Masterarbeit wird ein bereits bestehender Prototyp eines LiDAR Systems verwendet, um Messdaten mit dem erstellten System zu generieren. Der Prototyp besteht aus einem Empfangsboard mit 32 Avalanche Photo Dioden (APDs), einem mikroelektromechanischem Spiegel, Empfangsoptik und einer Steuerplatine. Sämtliche Komponenten des Prototyps wurden von der Firma Infineon Technologies entwickelt. Die Empfangsleitungen der 32 APDs werden zu den Anschlüssen des Artix 7 FPGA geführt und danach von dem in der Masterarbeit entwickelten System abgetastet. Das präsentierte System ist ansonsten unabhängig von der verwendeten Hardware und kann in jeglichem FPGA-basierten LiDAR System verwendet werden. Mit den bestehenden Hardwarelimitierungen kann das implementierte System 40 000 Messungen pro Sekunde verarbeiten. Dies entspricht, angewendet auf den Prototypen, 300 Bildern pro Sekunde.

Das erarbeitete System wird unter Verwendung von verschiedenen Materialien mit unterschiedlicher Reflektivität evaluiert. Sowohl das Rauschen in den resultierenden Daten

als auch das Sensor Übersprechverhalten werden dabei analysiert. Extrahierte Attribute aus dem Signal werden verwendet um die Qualität der produzierten Bilddaten zu verbessern. Ein weiterer LiDAR Prototyp wird verwendet um ein möglichst störendes Umfeld zu generieren und das System darin zu testen. Die Signalmittelung wurde dabei evaluiert und zeigt sehr stabile Resultate.

Abstract

Light Detection and Ranging (LiDAR) is a 3D imaging technique to generate a representation of the surrounding environment. It has become very popular during the last year in consumer and industrial electronics. It covers a wide variety of applications, ranging from airborne to terrestrial assemblies. It is used for topographical survey as well as documentation of local scenes. Signal analysis and processing of LiDAR data have advanced so far, that nowadays features like reflectance and velocity of objects can be determined. It is even used for finding structure integrity problems of concrete buildings. Due to the decreasing form factor of LiDAR systems, it is also applied to robotic and automotive areas for navigation assistance and similar systems. The application fields of automated driving and robotics require real time data analysis since fast decision making may be crucial for functional safety.

This thesis presents a novel real-time signal analyzing system capable of extracting features out of 1bit LiDAR data. This is accomplished by using the Xilinx Artix 7 platform. Hardware-accelerated components are designed in VHDL and implemented into the Artix 7 FPGA. The hardware-integrated solution allows high speed processing. An integrated MicroBlaze micro controller is used for configuration of the developed components during run-time. Feature extraction in form of pulse width, position and assumed relative reflectance is accomplished by analyzing the incoming digitalized light reflections. Additionally, a new technique for signal averaging is performed. The resulting data is then transmitted over Ethernet or USB to PC software. This powerful hardware system enables feature extraction while reducing required data throughput and still achieving essential speed-up in hardware.

In this work an existing prototype is extended in order to also gather real-world data with the proposed system. The prototype consists of a receiver board containing 32 Avalanche Photo Diodes (APDs), a microelectromechanical mirror system (MEMS), receiver optics and an controller board. All of these components are developed by Infineon Technologies. The 32 Low Voltage Differential Signal (LVDS) lines are fed into the Artix 7 FPGA and are then controlled by the system developed in this work. The presented algorithms can be applied to any generic LiDAR system with FPGA unit. In the existing system, the resulting work is capable of processing 40 000 measurements per second. With system specific settings, this corresponds to 300 FPS.

The system is tested with materials of different reflectivity at a variety of distances. Noise levels and cross talk phenomena are analyzed, while extracted features are used to improve the image quality. A second LiDAR prototype is used to create a noisy environment. Under these conditions, the signal averaging is analyzed and shows stable results.

Danksagung

Diese Diplomarbeit wurde am Institut für Technische Informatik an der Technischen Universität Graz durchgeführt. Der praktische Teil der Arbeit konnte bei Infineon Technologies Austria AG in Graz absolviert werden. Allen voran möchte ich mich bei all jenen bedanken, die am Entwicklungsprozess dieser Arbeit beteiligt waren.

Ich möchte mich für die Bemühungen und die Unterstützung meines Betreuers Ass.Prof. Dipl.-Ing. Dr.techn. Christian Steger bedanken. Durch hilfreiche Anmerkungen und Ratschläge hat er nicht nur zur Qualitätsverbesserung dieser Arbeit beigetragen, sondern hat auch dadurch erst einen positiven Abschluss möglich gemacht. Weiters bin ich dankbar für die Unterstützung und Hilfe meines Betreuers Dipl.Ing. Dr. Norbert Druml bei Infineon Technologies. Durch ihn habe ich nicht nur die Möglichkeit erhalten, eine Masterarbeit im Bereich LiDAR durchzuführen, sondern konnte darüber hinaus noch viel tiefer in diese Thematik eintauchen. Dank der stets konstruktiven Ratschläge und der professionellen Unterstützung konnte diese Arbeit erfolgreich zum Abschluss gebracht werden.

Weiters möchte ich mich bei meinen Freunden und Studienkollegen bedanken. Ohne die anhaltenden Ermutigungen und auch Ablenkungen wäre das Studium nicht zu so einem bemerkenswerten Erlebnis geworden. Hierbei freut mich besonders, dass aus Studienkollegen mit der Zeit Freunde fürs Leben wurden. Besonderer Dank gilt meiner Lebensgefährtin Theresa, der ich mehr zu verdanken habe als ein paar Zeilen fassen könnten. Danke für die unablässige Unterstützung und dein endloses Verständnis.

Schließlich möchte ich auch noch einen großen Dank meiner Familie aussprechen, ohne deren Geduld und letzten Endes auch finanzielle Unterstützung mein Studium nicht möglich gewesen wäre. Obwohl das Studium eine beträchtliche Distanz zwischen uns gebracht hat, so hat es dem Zusammenhalt nicht geschadet. Und ein Besuch im schönen Niederösterreich brachte immer wieder den notwendigen Ausgleich, durch den ich mein Studium mit Erfolg abschließen konnte. Vielen Dank.

Graz, im Mai 2018 Sebastian Krell

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In the last years, the interest in automated driving has increased enormously. Due to technological progress, basic sensors and algorithms are already available and automated driving is not a futuristic idea anymore. But in order to accomplish reliable decision making by these algorithms, reliable data of the surroundings is required. Ideally, the sensors are able to reconstruct the entire environment of the car. State-of-the-art implementations therefore work with multiple sensors in order to generate a 360 degree coverage surrounding the car. In Figure 1.1, the sensor equipment of a level 3 automated driv-

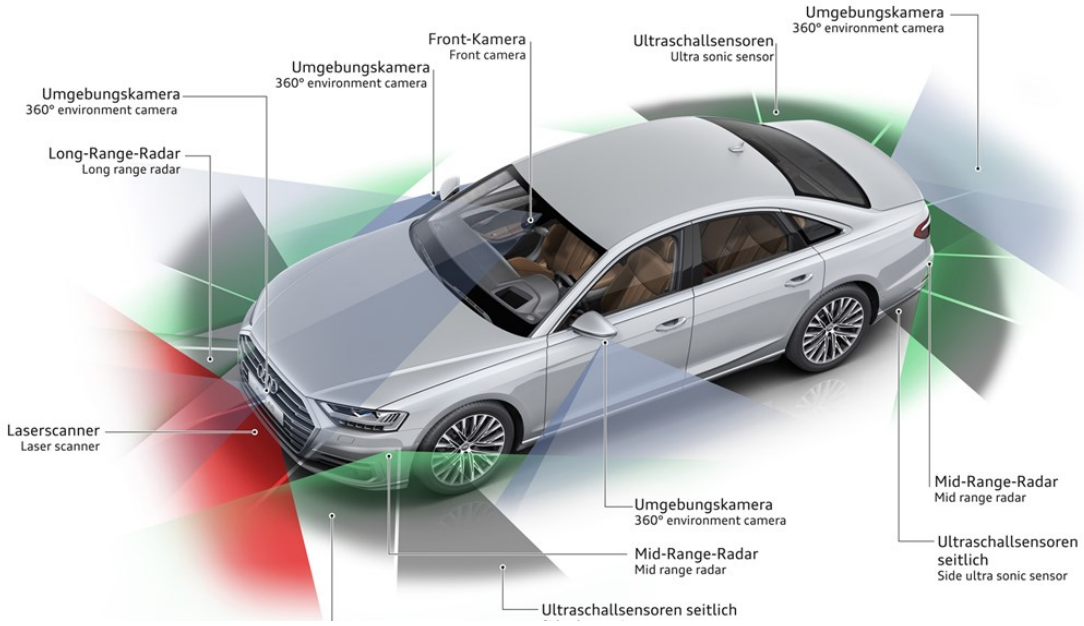


Figure 1.1: Sensor equipment of an Audi to cover 360 degree around the car [AG.17].

ing car is illustrated. As it can be seen, multiple different sensors are used to generate multiple sources of data for analyzing. The sensors used are radar, camera, ultra sonic and Light Detection and Ranging (LiDAR). These are mounted on different sides of the car and cover a different, sometimes overlapping area. For ensuring the functionality of automated assistance functionality, the system requires both redundancy and diversity. Further, the system needs robust sensor data.

In order to be capable of processing all the sensor data and fuse them, the noise of the sources needs to be kept as low as possible. Another key aspect is the ability of processing the data in real time. In this thesis, a real-time data processing LiDAR system is presented. This system reduces the data to the essential information as well as increases the Signal-Noise Ratio (SNR) through signal averaging. All these processing steps are performed in real-time on a Field Programmable Gate Array (FPGA). Figure

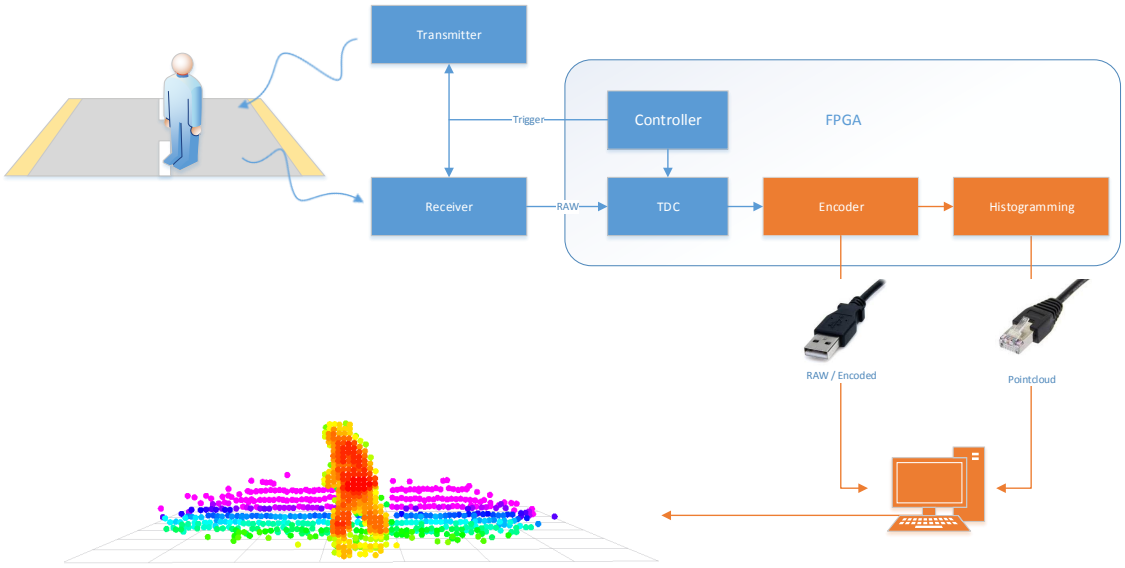


Figure 1.2: Overview of the LiDAR system implemented in this thesis. Orange blocks are extended to an already existing system.

1.2 illustrates an overview of the system presented in this thesis. Blue components are already present in the existing prototype while orange colored modules are designed and implemented in this work. Generally, as in most existing LiDAR systems, a transmitter illuminates a scene, while a receiver samples the reflected light. The location of objects in the scene can be computed with a time-to-digital converter (TDC). The developed encoder then extracts all necessary information. This is passed on to the histogramming module, which performs signal averaging in order to increases SNR. Both components can transmit their data via an interface to a post-processing system. It is then possible to generate a 3D representation of the observed scene.

1.2 Objectives

This thesis focuses on the signal processing of 1bit LiDAR system. The signal processing techniques are designed and implemented for real-time processing and evaluated in a working prototype. The main goals of this work are:

- Extension of an existing 1bit LiDAR prototype implementation by adding encoding and histogramming to the signal processing path.
- Development of a data compression encoding on a FPGA meeting real-time constraints.
- Transfer and visualization of the gathered encoded data-stream
- Development and implementation of a histogramming module to increase the SNR.
- Provision of a software capturing and displaying histogrammed data.
- Evaluation of feature extraction of 1bit LiDAR data.

The implementation of the encoding and histogramming module will evaluate the usability of feature extraction on 1bit LiDAR data. In case of succeeding, LiDAR systems without analog-to-digital converter (ADC)s and therefore with lower costs could be constructed. Due to the real-time computation and the lower data rate, algorithms can provide decision making in automated driving systems faster.

1.3 Outline

The structure of this thesis is as follows. In Chapter 2, the theoretical part of this work is covered. The current state-of-the-art of LiDAR signal processing is described. The principle of time-of-flight as well as its current applications is presented. Additionally, different methods of feature extraction and their interpretation are explained. Chapter 3 contains the requirements for the work in this thesis. Further, it gives an overview of the components this work is split into. It also describes the structure of the processing algorithms used in detail. The workflow of the development as well as implementation related details are explained in Chapter 4. It also presents the implemented modules and their behavior. In Chapter 5, the results of this thesis are documented and discussed. It analyzes the implemented system regarding its performance and its limits. Finally, Chapter 6 summarizes the results while suggestions for further work based on this thesis are presented.

Chapter 2

Related Work

This chapter starts with an introduction to LiDAR technology and gives an overview of related topics. First, the concept of 3D imaging based on time of flight is explained. Next, possible features are listed and analyzed that can be extracted out of these measurements. Finally, certain state-of-the-art applications and currently investigated research topics are introduced.

2.1 Laser ranging principle

Light detection and ranging (LiDAR), also known as laser detection and ranging (Ladar), relies on the time-of-flight principle (see Figure 2.1). Therefore, light pulses are emitted by the illumination unit of the LiDAR, reflected by the scene, travel back to the LiDAR and are detected by its receiving unit. The time t between emission and receiving of the light pulses is measured. By knowing the propagation speed of light c, the distance r can be derived according to Formula 2.1 [CC13].

$$r = \frac{c \cdot t}{2} \tag{2.1}$$

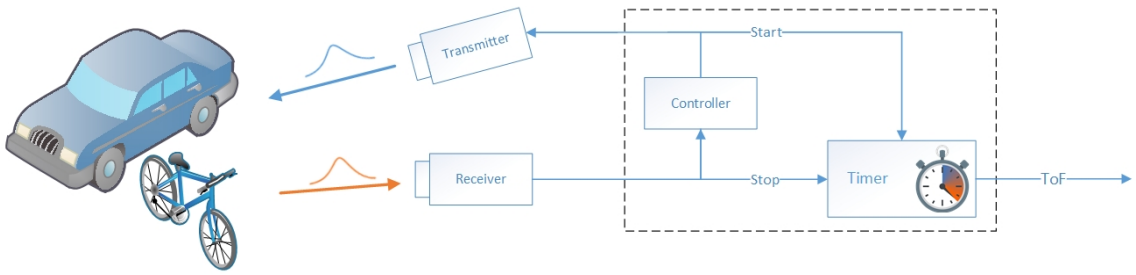


Figure 2.1: Direct time-of-flight principle.

2.2 LiDAR Systems - State-of-the-Art

In the following Section, different state-of-the-art technologies for LiDAR systems are introduced. These include different technologies regarding the illumination unit, the receiving unit and signal processing approaches. Additionally, LiDAR systems can be distinguished by their field of application.

2.2.1 Field of Application

Depending on the field of application, mainly three different types of LiDAR systems are distinguished. These are: terrestrial, airborne and mobile LiDARs. Terrestrial LiDARs are mounted on a tripod and usually perform a 3D scanning with one range measurement and two angle measurements. Airborne LiDARs are mounted on an aircraft and mobile LiDARs on a ground-based vehicle. Usually, airborne and mobile LiDARs perform two dimensional scanning with one line scan in one scan angle. [UPb]

2.2.2 Time-of-Flight Measuring

In general, there exist three different approaches for the time-of-flight measurement. For the first one, *pulsed modulation*, discrete light pulses are emitted and the time-of-flight is directly measured by measuring the passing time between emission and receiving of the light pulses. With the second approach, *continuous wave modulation*, a continuous waveform is modulated onto the emitted light signal and the phase shift between outgoing and incoming signal is measured. A pseudo-random number sequence is encoded with the third approach, *pseudo-random number modulation*, and then an autocorrelation is performed with the received signal. [CC13]

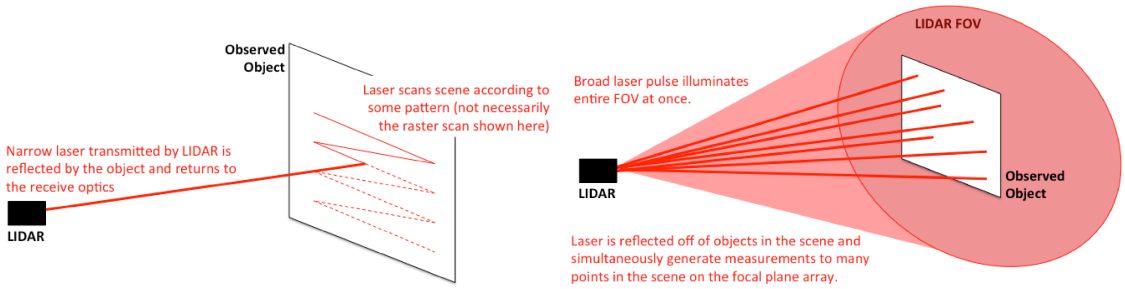
2.2.3 Illumination/Sensing

Depending on the illumination and sensing technology, two different types of LiDAR system can be distinguished. These types will be described more in detail in the following Section.

Scanning LiDARs

Scanning LiDARS have a narrow laser beam that is swept over the field of view in order to scan the scene. This approach is illustrated in Figure 2.2a. Either a single detector or an array of detectors is used to receive the reflected light. A system of mirrors, lenses or similar devices enables this system to sweep the narrow laser beam. By knowing the direction of the laser beam and the measured time-of-flight, a 3D point cloud can be obtained.

An advantage of scanning LiDAR systems is that they provide a high resolution at high precision. A typical application field is tracking of objects. While the system of



(a) Example for a scanning LiDAR system. (b) Example for a detector array LiDAR system.

Figure 2.2: Two different illumination/sensing technologies for LiDAR systems [CC13].

mirrors provides a high precision of the direction of the laser beam on the one side, these moving parts can wear out or lead to hardware problems on the other side. [SKKK]

Flash LiDARs

Flash LiDARs illuminate the entire scene at once, as it is illustrated in Figure 2.2b. The receiving unit is formed by a more-dimensional array of detectors. This means, that each pixel measures the time-of-flight of the reflected light it senses. With this method, a 3D image is obtained by combining the 2D pixel information with the measured time-of-flights. An advantage of detector array LiDARs over scanning LiDARs is that they are less failure-prone since they do not have any moving parts. However, since several detectors are used for the receiving unit, the complexity of calibration is increasing. [SKKK]

2.2.4 Receiver

For detection of the light pulses, APDs are used. APDs are fast photo diodes that produce electrons via photo-electric effect and use the avalanche breakdown for internal amplification. These APDs can be driven in Geiger mode or linear mode. In this Section, these two different types for receiving are explained and their advantages and disadvantages are discussed.

Linear Mode

When an APD is used in linear mode, the via photoelectric effect produced current is proportional to the incoming photons. Most systems working with APDs in linear mode use a single laser and a single receiver that are swept over the field-of-view by reflecting and refracting elements [UPa]. These specifications yield in the fact that data is acquired sequentially.

Usually, ADCs are used to convert the output of the APDs into a stream of samples. In the first step, this stream is analyzed by separating noise and signal in order to detect the received echo signals, hence the targets. Therefore, a threshold is used that distinguishes

between noise and a target. In the next step, the temporal position of these detected targets is determined and subsequently the distances to these targets can be derived. Additionally, information such as signal strength and reflectance of the target are estimated. A characteristic of LiDARs using linear APDs is a high distance resolution.

Geiger Mode

LiDARs in Geiger mode use APDs that are biased above their breakdown voltage. This means, that already a single photons triggers the APD. A comparison between linear and Geiger mode APDs can be seen in Figure 2.3. Another difference to linear mode is that a pixel can be triggered just once per laser pulse. As a result,the distance resolution depends on the recovery time of the APD. If the first target reflects enough photons, this target will be detected and targets near behind the first are ignored, since the APD already broke through. Another characteristic resulting from the effect of being triggered by a single photon is that no information like signal strength can be extracted. However, an advantage of Geiger mode LiDARs is that they can be used for very high range applications thanks to their high sensitivity, resulting from single photon triggering. [UPa] This high sensitivity

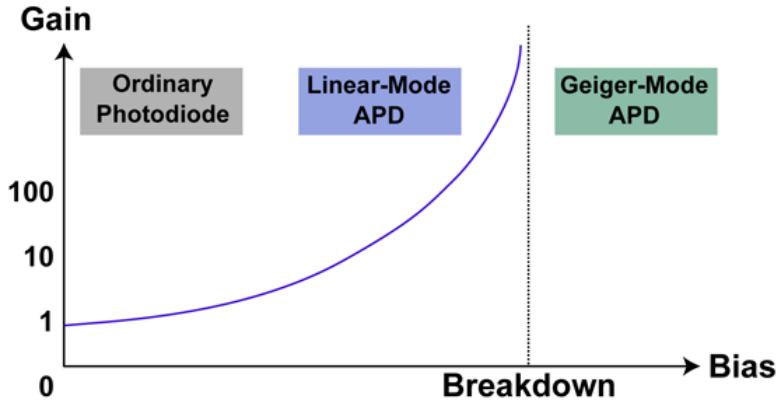


Figure 2.3: Comparison of ordinary photo diode, linear mode and Geiger mode APDs [Lab].

also leads to issues regarding solar background. It is an important point to minimize this solar background as far as possible. Therefore, different considerations play a role. First of all, the wavelength used for the system has a high impact and has to be taken into consideration. Further methods for reduction of solar background are listed from Stoker et al. [SANW16]:

- minimization of the system aperture
- implementation of a bandpass filter in the receive path
- reduction of the instantaneous field of view of the detector
- minimization of the range gate duration
- operation at night

2.2.5 Signal Processing

In this Section, different aspects and technologies regarding signal processing for feature extraction and classification are introduced. The processing steps can happen in real-time, but also be applied on offline data after the measurements. The possible signal processing steps are highly dependent on the type of data available through one measurement.

Analog Discrete vs Full-Waveform Return LiDARs

Mainly two different kinds of LiDARs exist regarding the waveform of the return signal, analog discrete return and full waveform LiDARs. These are illustrated in Figure 2.4. With analog discrete return, each reflection is represented by a discrete peak at a certain time. Usually, TDCs are used for these kinds of LiDARs. The electric signal generated by photo-electric effect of the APDs triggers a TDC. By knowing the time when a peak returns, the range to the detected object can be calculated. The only information that can be obtained from the peak, is that there was a reflection. But no information about the signal strength of the return peak or its pulse width is gained.

Contrary, full-waveform LiDARs receive a continuous waveform of the return signal instead of discrete peaks. Usually, this is achieved by the usage of APDs in linear mode and fast ADCs for digitization. Full-waveform return signals provide additional information beside the distance, such as signal strength or pulse width. With these signal properties, features like reflectance, classification of targets' surface or object classification - just to mention a few of them - can be extracted. [UPa]

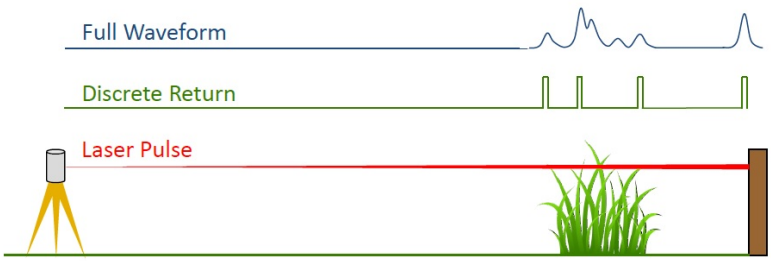


Figure 2.4: Illustration of analog discrete return and full-waveform return LiDARs [HGF13].

Feature Extraction with Full-Waveform Return LiDARs

As assumed by Ullrich and Pfennigbauer [UP11], the system response $s_E(t)$ of a received signal is a convolution of the emitted laser pulse $s_R(t)$ with the target signature $T(t)$ [PWWU14]:

$$s_E(t) = s_R(t) * T(t) \tag{2.2}$$

This is also illustrated in Figure 2.5. The red line represents the shape of the laser pulse $s_R(t)$, the black line the target signature $T(t)$ and the green line the system response

$s_E(t)$. However, if it is possible to extract the target signature by a reverse convolution, properties about the detected target could be derived. This deconvolution is the current aim of full-waveform analysis. In general, there exist two kinds of approaches for this reverse convolution. The first one is a rigorous deconvolution. A disadvantage of this approach is that it is prone to noise. Another idea for the extraction of the backscattering properties is the attempt to reconstruct the received signal by the combination of basic functions. The most popular approach for this is Gaussian decomposition.

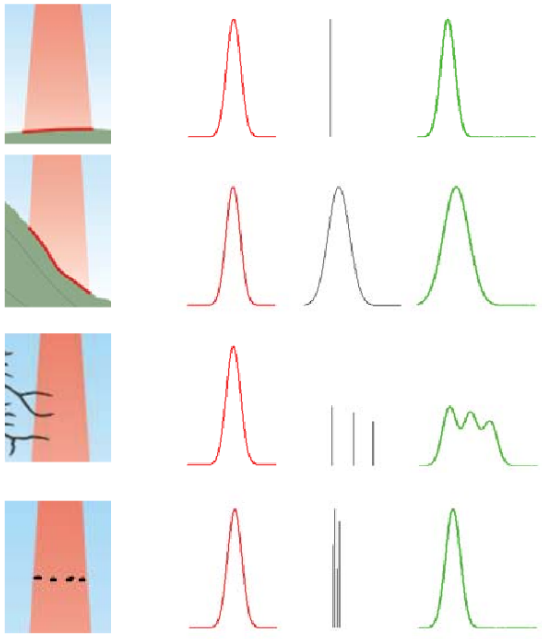


Figure 2.5: The system response (green line) of a received signal can be described as convolution of emitted laser pulse (red line) and target signature (black line) [UP11].

2.3 State-of-the-Art

For this Section, five different papers are presented that cover current research topics regarding LiDAR technology. These cover different techniques of feature extraction and their use. Since this work implements a histogram and analyzes the data of a discrete LiDAR system, the main focus on the presented papers is on signal processing. Full waveform LiDARs are more and more used in the research field since these provide the most feature-rich signals. But the underlying physical properties of these signals are also valid for discrete LiDAR systems.

2.3.1 Calibration for Reflectance

In this Section, the work of Hartzell et al. [HGF13] is presented. The authors aim to develop templates for system response waveforms empirically. These templates should cover the dynamic range of terrestrial laser scanners when Gaussian fitting appears inapplicable. As a result, it should be possible to estimate the reflectance of an object by fitting the measured system response to a matching beforehand derived template.

Experimental Setup

The used hardware consists of a VZ-400. This is a tripod laser scanner from Riegl. In order to be capable of recording full waveform of the backscattered laser beam, a firmware upgrade was accomplished by the manufacturer. The most important specifications of the VZ-400 are summarized in Table 2.1.

	Longe Range Mode	High Speed Mode
Effective Pulse Rate (Hz)	42000	122000
Max. Range (reflectivity 90%)(m)	600	800
Max. Range (reflectivity 29%)(m)	280	160
Beam Divergence ($mrad$)	0.3	0.3
Angular Resolution (°)	0.0005	0.0005
Laser Wavelength (nm)	1550	1550

Table 2.1: Specifications of the VZ-400

Along with the VZ-400, the authors encountered two main issues. The first one is that very few points are captured of the response pulse. This effect results from the short pulse width of the outgoing signal on the one hand, and the low digitization rate on the other hand. A typical waveform of such a response pulse is illustrated in Figure 2.6. The second problem arises from the fact that the response pulse does not describe a Gaussian distribution. Hence, Gaussian fitting is not very applicable for this laser scanner. Further, the user receives hardly any information about the internal properties of the scanner. Therefore, three 12"x 12"reflectance targets out of spectralon are used in order to measure with known reflectance and geometric configurations. At a wavelength of 1550 nm, these three targets have a reflectance of 99%, 62%, and 30%.

Measurement Performance

Several thousand measurements are performed at different distances (2m - 260m) in order to cover the dynamic range of the VZ-400 comprehensively. Then, the authors use cubic spline fit to align the waveforms. Thus, a 3D representation of the system response is achieved for each return pulse amplitude. This can be seen in Figure 2.7b, while Figure 2.7a illustrates a sample of a splined response pulse. All aligned waveforms for a certain distance are combined to obtain an averaged waveform that is representative for this

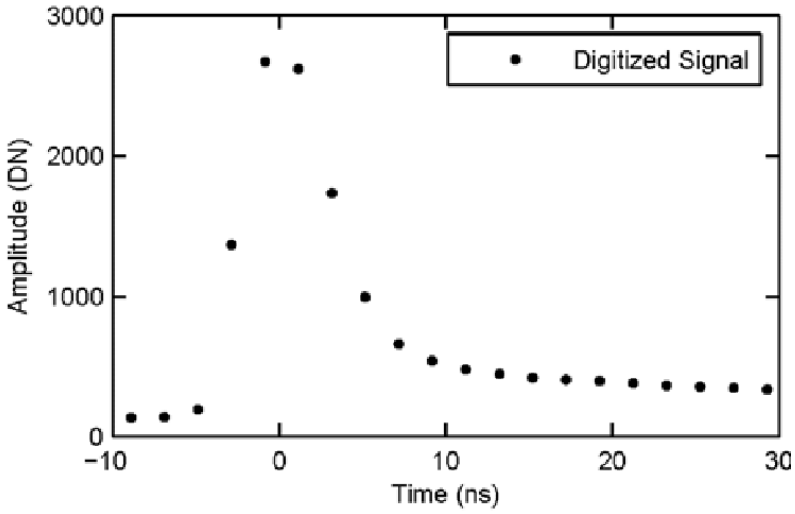
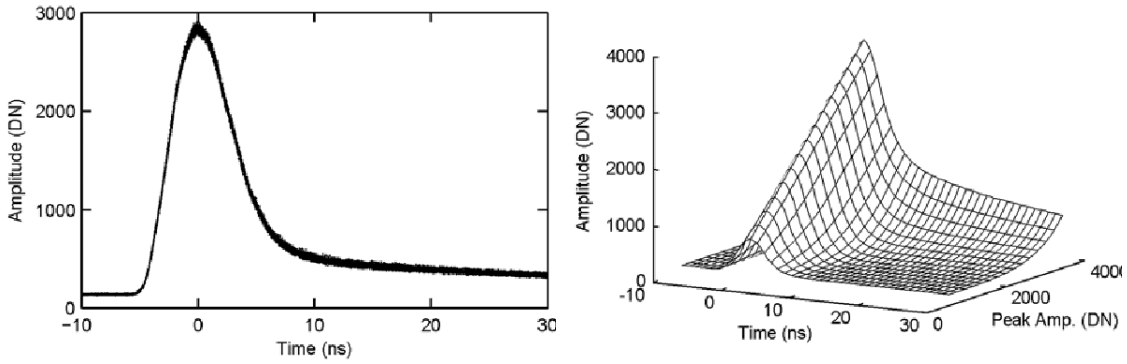


Figure 2.6: A typical waveform of a response of the VZ-400 [HGF13].

distance. All averaged waveforms over the different distances are then combined and form a template for a certain reflectance.



(a) Sample system response at certain distance and reflectance.

(b) Alignment of several waveforms to obtain 3D representation.

Figure 2.7: The collected measured waveforms are aligned using a cubic spline fit in order to receive a 3D representation for each response pulse [HGF13].

Additionally, not only the distance was varied, but also the incident angle. The measurements were performed at incident angles of 0°, 20°, 40°, and 60°. This topic is covered more in detail in the next section.

Incident Angle

Hartzell et al. [HGF13] observed interesting effects regarding the incident angle. When observing the response waveform at the same distance with different incident angles, they expected the amplitude to decrease along with an increasing angle. This expectation was only met for distances above 160 m, but not for shorter distances.

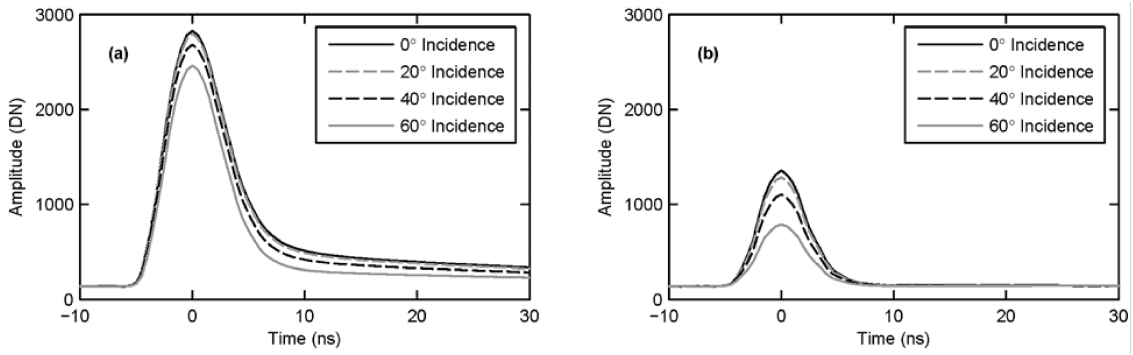


Figure 2.8: Waveforms of backscattered signals with different incident angles (on the left side with a distance of 32 m, on the right side with 160 m). The amplitude does not change as much, regarding to the changing incident angle, as expected [HGF13].

Figure 2.8 illustrates this effect. On the left side, the response pulse for different incident angles at a distance of 32m is shown, while the right side presents the results for a distance of 160m. As explanation, the authors refer to [PWKJ07]. The authors of this paper claim, that the spectralon targets do not act as perfectly diffuse lambertian surfaces. Further, Hartzell et al. cite the manufacturers of the VZ-400 who explain that non-linearity occurs in the process of signal detection for shorter ranges resulting from a stronger returned echo signal.

A second unexpected observation is the fact that an increasing incident angle does not change the pulse width of the returned pulse waveform remarkably. This phenomenon is explained by the authors with the small divergence of the VZ-400.

Absolute Reflectance

The returned peak amplitude of common laser scanners is usually uncalibrated. Therefore, it does not provide information about the reflectance of the object that was hit by the laser beam. With the VZ-400 from Riegl however, the manufacturers provide two information about the recorded system response that they call *calibrated amplitude* and *relative reflectance*. The first one is a value provided in DB and is depending on the distance to the target. The second one, *relative reflectance*, describes the ratio between the calibrated amplitude of the measured target to the calibrated amplitude measured to a white reflectance target at the same distance. Hence, a relative reflectance value is obtained.

Hartzell et al. instead aimed to get an estimation of the absolute reflectance of a

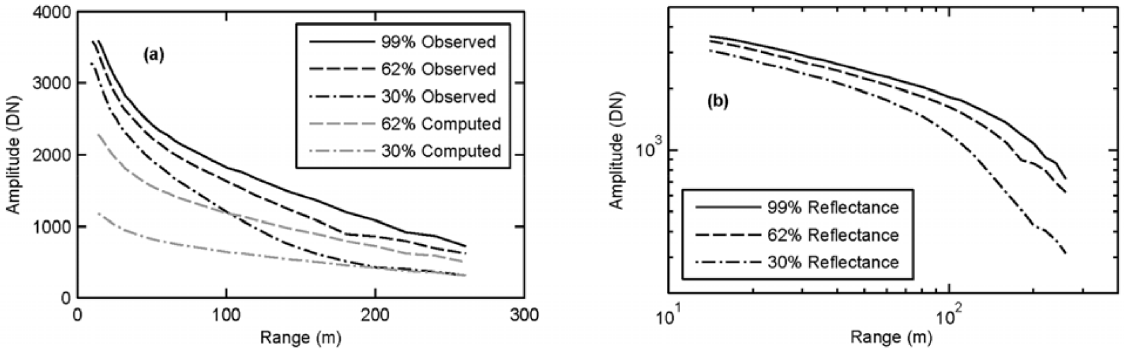


Figure 2.9: Amplitude peaks for the three different reflectance targets covering the entire distance range. The black lines on the left side show the measured results for the three targets while the gray lines show estimated peak values for reflectances of 30% and 62% depending on the measured values for 99% reflectance. The image on the right side shows the same results but plotted on logarithmic axes [HGF13].

target. Therefore, they interpolated the peak amplitude values over the entire distance range. Additionally, this was performed for all three reflectance targets. The reason for this can be seen in Figure 2.9 on the left side. The black lines show peak amplitudes for all distances for the three different reflectance targets. For the gray lines instead, the expected peaks for the reflectances of 62% and 30% were calculated by taking 62% and 30% of the measured peaks for the reflectance of 99%. As it can be seen in the left image, for distances lower than 200 m, the real measured peak amplitudes for reflectances of 62% and 30% are higher than the calculated/expected ones. The right side of Figure 2.9 shows the same results as the left side, just on logarithmic scaled axes.

2.3.2 Fog classification

Pfennigbauer et al. [PWWU14] investigated how different technologies perform in an turbid environment with an attenuating and scattering media. Therefore, they performed their measurements in a fog chamber. The used LiDAR is a Riegl VZ-1000. For this investigation, it is assumed that attenuation and the backscatter coefficient are constant within the medium. Hence, the backscatter profile can be described as following:

$$T(t) = a\delta(t_0) + b\sigma(t - t_0)e^{\frac{-(t-t_0)}{\tau}} \tag{2.3}$$

In this equation, the first part describes the portion of diffuse reflection within the backscatter cross section a at time zero. This means, this part handles the time when a turbid medium is entered. Since the measurements for this experiment are performed only *inside* the fog chamber, this part can be neglected. In the second part of the equation, the distributed reflectance inside the medium is handled, where τ denotes the penetration depth and b the backscatter factor.

Experimental Setup

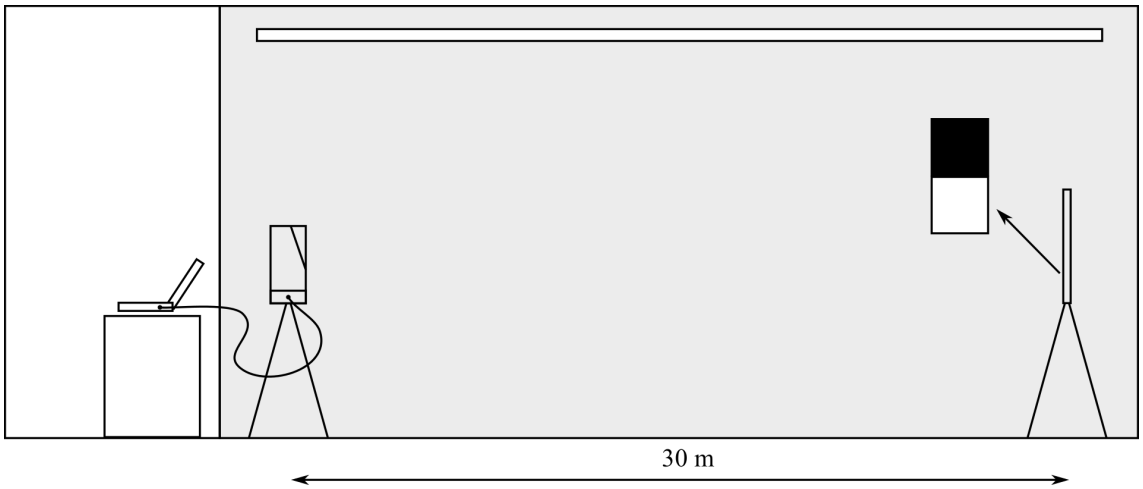


Figure 2.10: Setup of the experiment for measuring inside a fog chamber [PWWU14].

The measurements were performed inside a fog chamber that is capable of producing spatially homogeneous fog remaining constant over time. The target was formed by a flat board consisting of a black and a white surface providing a reflectance of 3% and 100% at the LiDARs wavelength. The target and the LiDAR were both fix mounted at a distance of 30m. The LiDAR was working with a frequency of 70 kHz in line scanning mode and with 2.5 lines per second. Figure 2.10 shows the setup.

Results

The measurements were performed once without fog for reference results, once with a lower density of fog with 40m visibility and once with a higher density with a visibility of 10m. The left column in Figure 2.11 shows the results for the absence of fog. The range is 30m, and the results for amplitude and reflectance show the difference between the black and the white surface of the target. The pulse shape deviation is as expected approximately zero.

The second column shows the results for the lower density of fog with a visibility of 40m. Having a look at the range results, the target at a distance of 30m is still detected, but almost every measurement point detects another target at a distance of 2.5m. These points result from the echoes of the fog. The results for the amplitude show that the echoes of the fog have a higher amplitude than the ones from the target. This is a plausible result since the fog causes the reflections at a shorter distance than the target board. The reflectance results appear the same as the results without fog, because the atmospheric attenuation has been taken into account. However, it can also be seen that the reflectance of the fog is as high as the reflectance of the black surface of the target, while the white surface has still a higher reflectance than the fog. A look at the pulse shape deviation demonstrates that the deviation of the fog echoes is highly increased in comparison to the

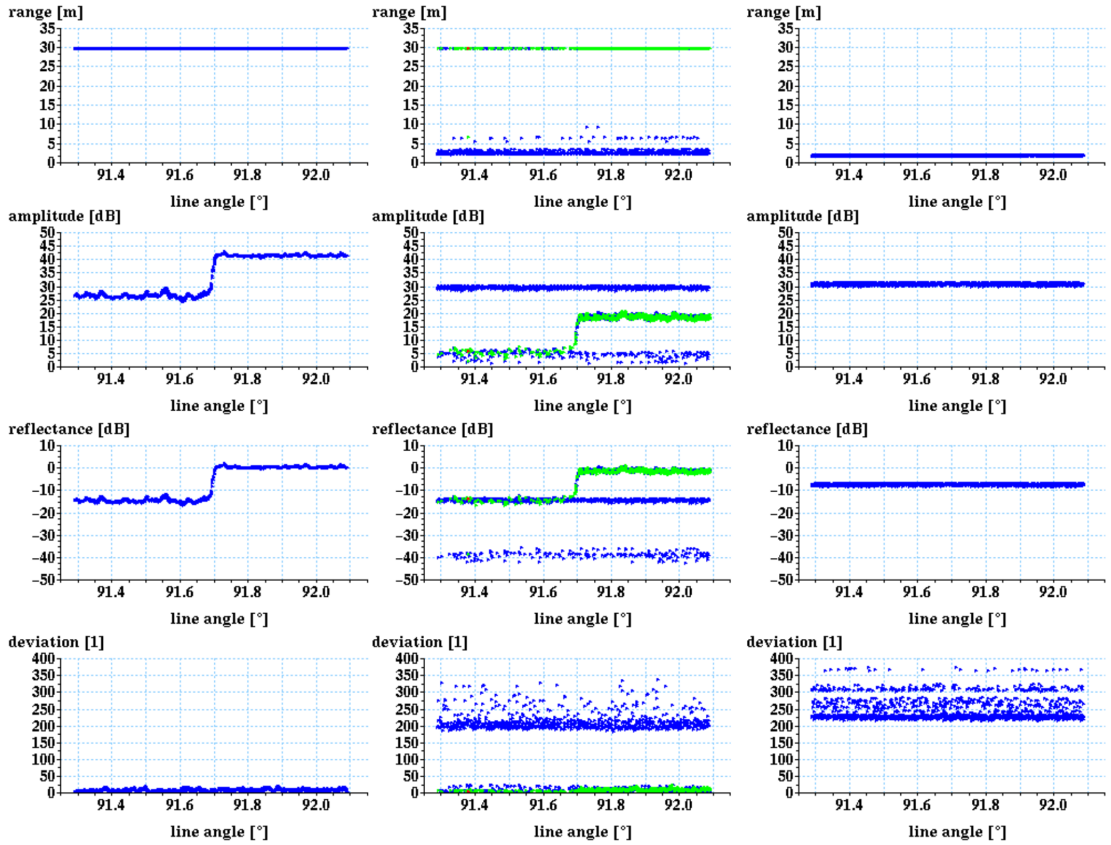


Figure 2.11: Results of the experiment in a turbid medium. The left column shows the results without fog, the middle column for a fog density with 40 m visibility and the right column with 10 m visibility. The blue points show first hit targets, the green points second targets.[PWWU14]

deviation of the target. This circumstance can be used to distinguish between fog and target.

The third column presents the results of the higher fog density with a visibility of 10m. Now, the atmospheric attenuation is too high to still allow a detection of the target at the distance of 30m. In this case, all measured results come from the fog echoes.

In the next step, Pfennigbauer et al. [PWWU14] analyzed the measured waveforms with the aim to estimate the visibility range in a turbid environment. Therefore, they recorded 1000 waveforms of the fog at different visibility ranges over time. This is illustrated in Figure 2.12. Their approach is to determine the temporal distance D_{COM} between the rising edge T_{re} of the waveform and its center of mass T_{COM}.

$$D_{COM} = T_{COM} - T_{re} = \frac{\sum t_i s_i}{\sum s_i} - T_{re} \tag{2.4}$$

s_i refer to the sampling values and t_i to the sampling instances, while T_{re} denotes the position where the rising edge has reached the half amplitude of the peak.

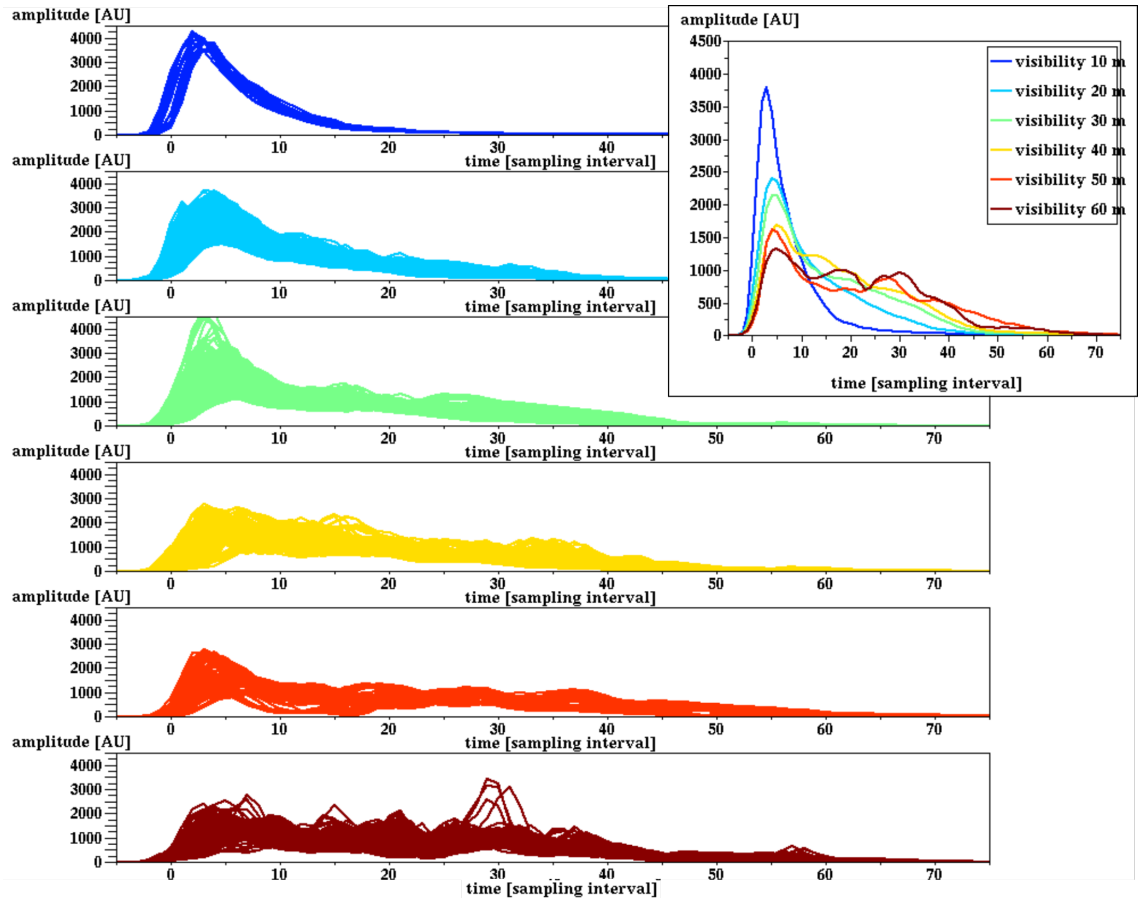


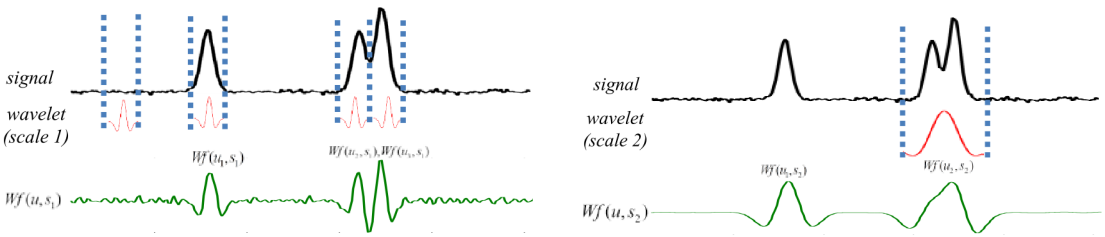
Figure 2.12: Recording of the waveform of fog at different visibility ranges [PWWU14].

2.3.3 Wavelet-Based Echo Detector

In 2012, the work of Wang [Wan12] covered the investigation of overlapped and weak return signals. Weak and overlapped waveforms may occur when multiple targets are within a single measurement at different distances. While weak signals are hard to detect because of their low SNR, overlapped signals are the harder to separate the more coincident their peaks are. Hence, within this work a wavelet-based echo detection algorithm was developed that should outperform common zero crossing detectors.

A continuous wavelet transformation (CWT) is a decomposition of a signal into wavelets as base function. Therefore, a chosen wavelet is compared to the target signal at different scales and also at different positions by shifting the wavelet. For each position and scale, a wavelet coefficient (WC) is calculated that indicates the similarity between the signal and the wavelet. Hence, applied to full-waveform analysis, each occuring WC peak implies an echo from a target in the received waveform. This is illustrated in Figure 2.13.

In Figure 2.13a and 2.13b, two different scales for the wavelets were applied. In the left image, the scale matches quite precisely with the received waveform in that all three



(a) CWT at scale s_1 (b) CWT at scale s_1

Figure 2.13: Example for echo detection using continuous wavelet transformation [Wan12].

targets are detected at the correct position. Unlike the right image, where the scale is too big and hence only two targets are detected. For this work, the author chose an Gaussian wavelet as mother wavelet while the scale factor was determined by the detection results.

Experimental Setup

For this project, the author has decided to simulate waveforms aiming to investigate the limitations of the proposed detector. Therefore, he describes a waveform as following, where a Gaussian function is chosen to represent a return echo:

$$w(t) = \sum_{k=1}^{m} g_k(t) + n \tag{2.5}$$

$$g(t) = A \cdot exp(-\frac{(t-\mu)^2}{2s^2}) \tag{2.6}$$

$w(t)$ is the simulated waveform, m the number of echoes, n the noises, μ the time domain location and s is the echo width. Different SNR levels were chosen for the experiment by increasing the power of the echoes. 1000 waveforms are generated for each SNR level. For the evaluation of the results, the following three rates are considered: the rate of the correctly detected echoes *CR1*, the rate of the missing echoes *MR1* and the rate where too many echoes are detected *RR1*.

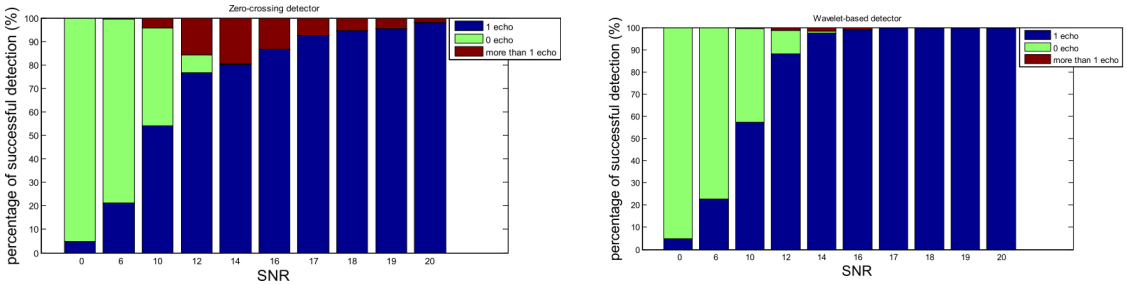
$$CR1 = \frac{number \quad of \quad detecting \quad 1 \quad echo}{1000} \cdot 100\% \tag{2.7}$$

$$MR1 = \frac{number \quad of \quad detecting \quad 0 \quad echo}{1000} \cdot 100\% \tag{2.8}$$

$$RR1 = \frac{number \quad of \quad detecting \quad more \quad than \quad one \quad echo}{1000} \cdot 100\% \tag{2.9}$$

Results

The results of the wavelet-based detector are compared to the results for the same test data fed to a zero-crossing detector. A zero-crossing detector calculates the first derivative of the waveform and detects echoes at the zero-crossings of the derivative.



(a) Zero-crossing detector. (b) Wavelet-based detector.

Figure 2.14: Results for echo detection with different SNR levels for the zero-crossing and wavelet-based detectors [Wan12].

Figure 2.14 shows the results for both detectors. The results show that the wavelet-based detector reaches a *CR* of 100% at a lower SNR level than the zero-crossing detector. Further, the wavelet-based detector hardly detects too many echoes unlike the zero-crossing detector.

2.3.4 TOF-LIDAR signal processing using the CFAR detector

In 2016, Ogawa and Wanielik [OW16] presented their research in the field of detecting low SNR targets. The authors state that there are two main approaches for increasing the sensitivity for detection. The first approach is to improve the optical hardware, such as the laser diodes, the photo detectors and advanced scanning systems. The second approach is about applying signal processing techniques before the point detection for increasing the SNR. The common technique therefore is filtering, such as low-pass, band-pass filtering or coherent integration. At the end of signal processing, a threshold is applied for distinction between noise and targets. Therefore, a Constant False Alarm Rate (CFAR) is used in many cases. This is a mechanism that calculates the threshold dynamically depending on the background conditions for achieving an improved detection performance at a lower SNR. Additionally, Ogawa and Wanielik propose an algorithm that combines the features of integrating signals and the CFAR approach.

CFAR Detector

Figure 2.15 shows the basic setup of a CFAR operator. It consists of cells under test, guard and training cells. The cell under test refers to the point at which evaluation is performed whether there is a target or not. The training cells are used to evaluate the background level by analyzing its intensities. Therefore, either the mean or the median of the intensities is calculated. The guard cells are used as range margin in order to isolate

the training cells from the intensity of an existing object at the cell under test. This cell

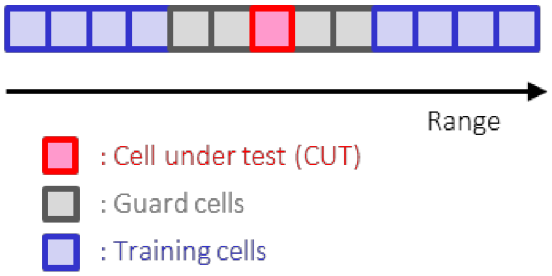


Figure 2.15: Basic design of a CFAR operator [OW16].

configuration is then implemented as sliding window operator. Finally, by evaluating the calculated mean/median values of the training cells, the background level is estimated and further the threshold is chosen above the background level. As outlined by Ogawa and Wanielik [OW16], this technique allows proper detection results in clutter, interference and other uncertain environments.

Extended CFAR Detector

In the work of [OW16], the authors propose an extended CFAR detector. Their idea is to combine coherent integration with the CFAR detector. Therefore, they extend the basic CFAR sliding window with neighbor cells in a second dimension. This is illustrated in Figure 2.16.

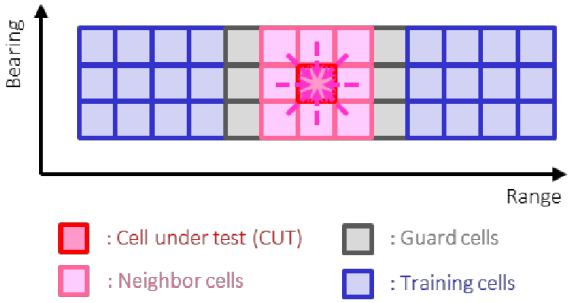


Figure 2.16: Design of the extended CFAR operator [OW16].

The neighboring cells are placed around the cell under test. This defines the region for intensity integration. Additionally, a numerical value is added to each cell that specifies the weight of the integration. For thresholding, the intensity is then evaluated with median value according to the corresponding weights.

This approach makes use of the fact that the pulse width of an echo is broader than it is high for the used LiDAR. Hence, an integration of the intensity over neighboring cells is performed. This is illustrated in Figure 2.17.

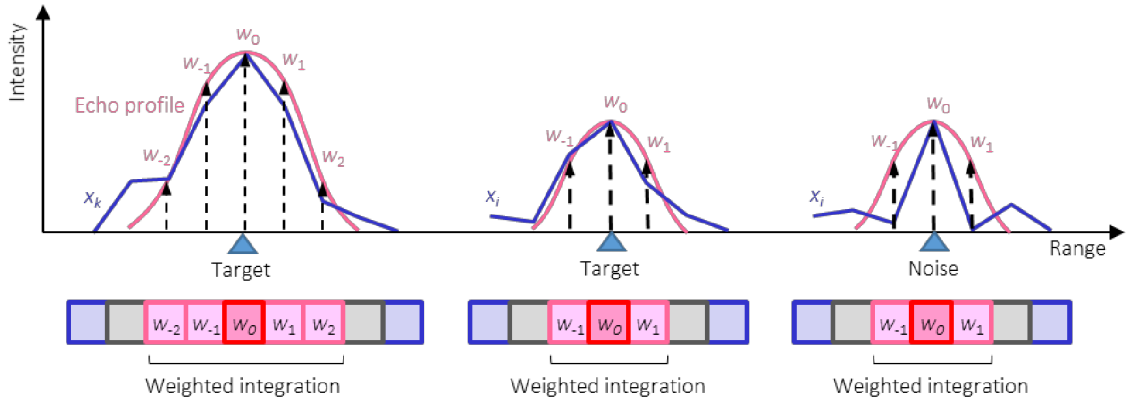


Figure 2.17: Intensity integration over neighboring cells [OW16].

Target Detection

Example results are presented in Figure 2.18. The above image shows the raw intensity values in combination with a constant threshold. In this case, no target is detected since all intensity values are beneath the threshold. In the lower image, the blue line indicates

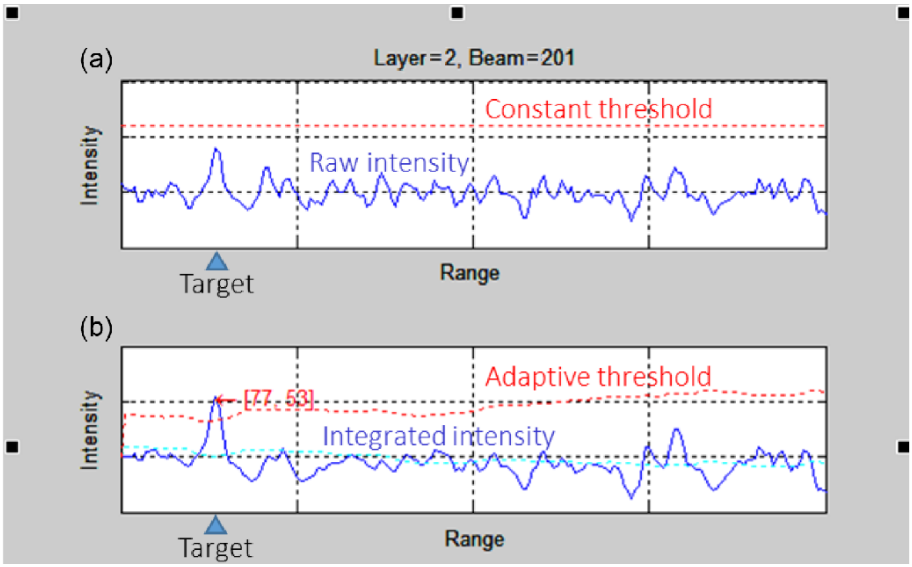


Figure 2.18: Example results showing the difference between a constant and an adaptive threshold for the detector [OW16].

the integrated intensity values and red dashed line the calculated, adaptive threshold. In this case, the target is detected. By comparing both images, it can be seen that the SNR is increased by integrating the intensities. Further, the adaptive threshold provides more accurate results than the constant one.

Experimental Setup

The experimental setup can be seen in Figure 2.19. The LiDAR is mounted fixed behind the windshield inside of the car. The target object has a width of 0.5 m and a height of 1.75 m. Its surface has a diffuse reflection of 10 % in the near infra-red band of the used LiDAR. The distance between the target and the car is altered in 5 m steps, while the measurement is performed for 30 s at each distance. Finally, the raw data is processed with the conventional and the proposed detector and the results are compared.

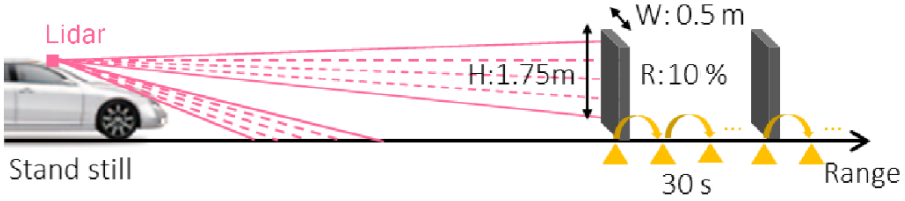


Figure 2.19: Experimental setup [OW16].

Results

First, the authors of [OW16] have a look at the Receiver Operation Characteristic (ROC) curves for both detectors, the conventional and the proposed one, at a distance of 60 m. These can be seen in Figure 2.20. The x-axis shows the false positive rate *FP*, a value that indicates the ratio between the number of false detections and the number of bins without target. The y-axis shows the true positive rate *TP*, so the ratio between the number of detected targets and the total number of bins. It can be clearly seen in Figure 2.20, that the proposed detector shows an improved performance in comparison to the conventional detector.

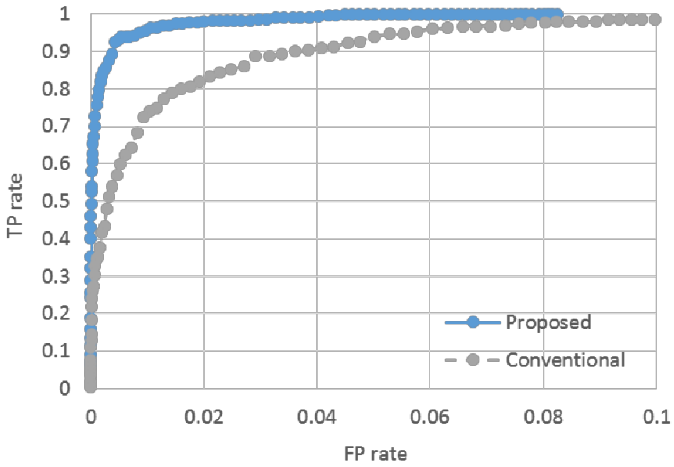


Figure 2.20: ROC curves for both detectors [OW16].

Figure 2.21 shows, that the TP rate could be improved enormously for distances be-

tween 40 m and 60 m with the proposed detector. As a result, targets at higher distances can be detected with the proposed detector that would get lost in the noise resulting from a too low SNR with a conventional detector. This is further shown in Figure 2.22, where a

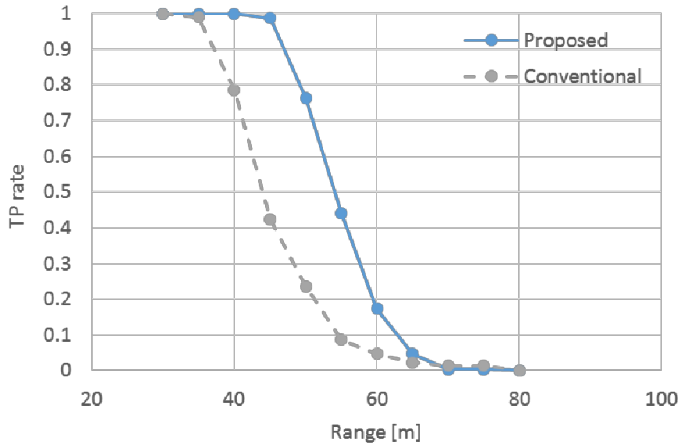


Figure 2.21: TP rates for both detectors at varying ranges [OW16].

scene consisting of a street, a building, grass and a pedestrian is recorded. It can be seen, that the proposed detector is able to detect the building at far higher distances. Further, unlike the conventional detector, it detects the pedestrian and also the grass.

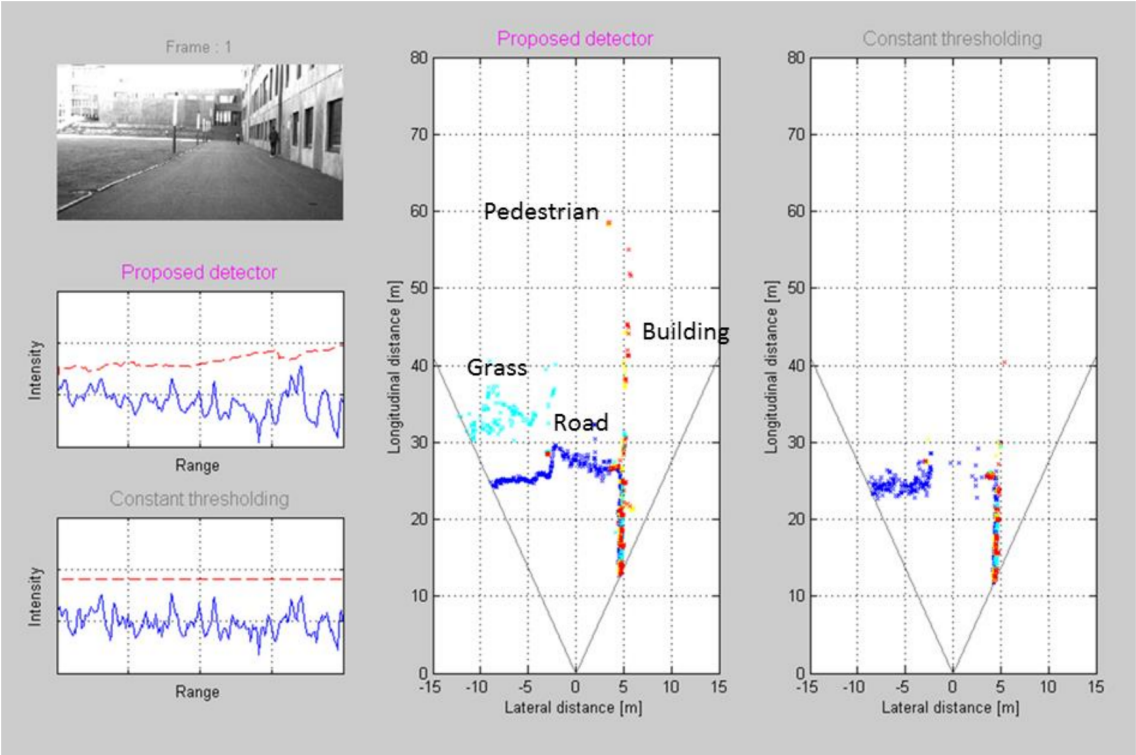


Figure 2.22: Recording of a scene for comparison of the conventional and the proposed detector [OW16].

Chapter 3

Design

In this Chapter, the design is discussed that was developed in this work.

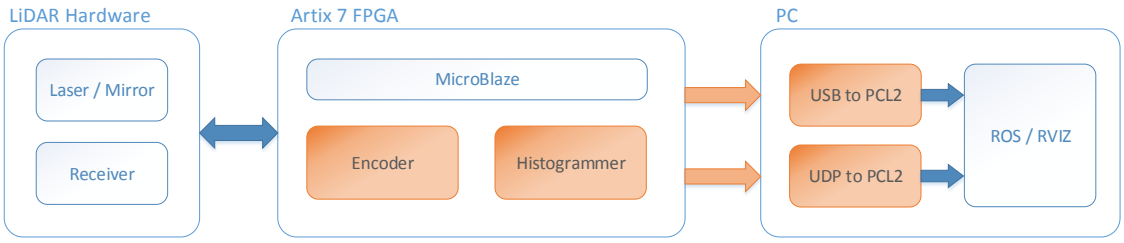


Figure 3.1: Overview of the LiDAR system and its interconnections. Orange components and interfaces are designed and implemented in this work.

In Figure 3.1 an overview of the designed system is illustrated. It can be roughly separated into 3 domains: the LiDAR prototype hardware, the Artix 7 FPGA and the software used to display the results. Orange outlined components are designed in this work and described in detail in the following subsections. This includes the modules *Encoder* and *Histogrammer*, as well as the interfaces to the PC. These receive the data from an existing LiDAR prototype. On software side the programs *USB to PCL2* and *UDP to PCL2* are designed.

First, the requirements of the resulting system are defined. Next, used frameworks as well as the existing LiDAR prototype is described. Also the two main modules *Histogrammer* and *Encoder* are discussed in detail. Last, the software design for presenting the resulting data is presented.

3.1 Requirements

An existing LiDAR platform is extended in a way that it will suit the needs for automated driving in real world. The key aspects of the resulting system are listed here:

Hardware-accelerated Computation
Signal processing algorithms are implemented on a FPGA. This ensures real-time data processing, which is crucial for tasks like collision detection and other real time decision making.

Data Visualisation
Produced point cloud data should be transferred over User Datagram Protocol (UDP) to a PC software. The software solution should be able to visualize and also store the received data.

Debug Connectivity
Aside of the UDP connection, an Universal Serial Bus (USB) interface should be used to transfer raw captured data out of the LiDAR system. This data should also be captured and stored by a PC software. With this data, different off-line post processing steps can be performed to evaluate the real-time processing or apply different filters.

The FPGA should perform different tasks in real-time. These tasks have to be implemented efficiently and provide additional data from the reflected light. In order to be able to evaluate these steps, it should also be possible to alter their configuration during runtime. The following task should be implemented:

Multiple Hit Receiver
The resulting system should be capable of receiving, analyzing and storing multiple reflections of one light beam in the scene. This way, it is possible to identify semi-transparent objects and filter false-positive detections.

Feature Extraction
When a light pulse is reflected by an object in the scene, the pulse shape is altered. In order to be able to analyze and possibly extract object properties out of the pulse, the duration of the received pulse should be recorded.

Histogramming
By performing a histogram on multiple scans of the same scene, it is possible to increase the SNR. Different algorithms should be compared and the most suitable should be implemented.

3.2 Existing Platform

In this section, the existing platform used for this thesis is presented. Therefore the hardware and software components of the prototype are discussed in detail.

3.2.1 LiDAR System Hardware

The LiDAR system consists of multiple independent hardware components connected by a controller board. These can be seen in Figure 3.2 and are described in the following

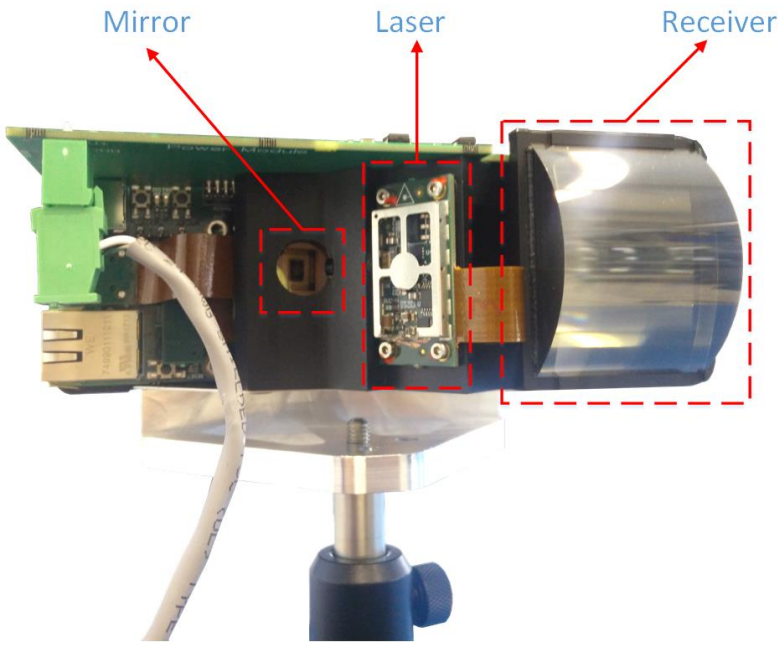


Figure 3.2: The existing LiDAR prototype and its components.

subsections.

Laser Module

The laser module is used to produce the required light-pulses for a measurement. It produces light with a wavelength of 905nm and a Full Width Half Maximum (FWHM) of 12ns. With a pulse-repetition-rate of 40kHz, this component mainly restricts the available time frame for the computation.

Micro-electro-mechanical System (MEMS) Mirror

Unlike most available MEMS mirrors on the market, the mirror used operates on a specific oscillation frequency. In order to be able to trigger the laser module, when the mirror is in the correct position, the frequency is measured by the controlling system.

Receiver Board

The receiver board, as seen in Figure 3.3, contains 32 APDs. Their signals are then amplified and fed into a comparator logic. For each channel, one Low Voltage Differential Signal (LVDS) is then provided for further analysis by the controlling system.

The threshold for the comparator logic, as well as other parameters, can be configured over a Serial Peripheral Interface (SPI). Since 32 APDs are provided, 64 signal lines are

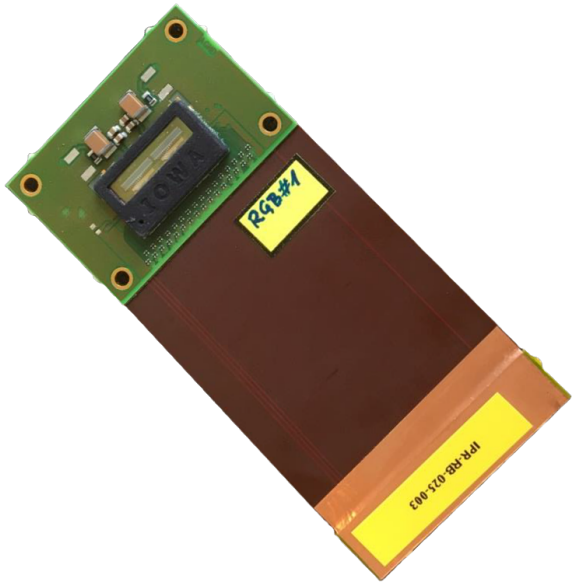


Figure 3.3: Receiver board with 32 APDs and LVDS lines.

used. LVDS makes a high speed transmission possible and is more error-prone than single transmission lines.

Artix Development Board

An Artix 7 development board for the signal-processing and needed controlling mechanism is used. Specifically the micromodule TE0712-02 featuring an Artix-7 FPGA, 1GByte DDR3 memory and 100MBit Ethernet from Trenz Electronic was chosen and comes with following specifications:

- Xilinx Artix-7 XC7A200T-2FBG484C
- 32 MByte (256 MBit) Quad-SPI Flash-memory
- 1 GByte 32-Bit DDR3 SDRAM
- 10/100 MBit Ethernet PHY in RMII Mode
- MAC address EEPROM
- Low Jitter Phase Lock Loop (PLL) (Silicon Labs Si5338)
- 158 FPGA in- and outputs (78 differential pairs) over board to board connector
- Plug-on Modul with 2 x 100-Pin and 1 x 60-Pin High-Speed connectors

The used FPGA Artix-7 has following internal resources:

- Logic Cells: 215,360

- Slices: 33,650
- CLB Flip-Flops: 269,200
- Maximum Distributed RAM (Kb): 2,888
- Block RAM/FIFO w/ ECC (36 Kb each): 365
- Total Block RAM (Kb): 13,140
- Clock Resources CMTs (1 MMCM + 1 PLL): 10
- I/O Resources Maximum Single-Ended I/O: 500
- Maximum Differential I/O Pairs: 240

A basic Very High Speed Integrated Circuit Hardware Description Language (VHDL) implementation of the controlling logic is also present. This implementation is capable of triggering the laser module, monitoring the mirror frequency, reading out the LVDS lines of the receiver and transmitting the data over the UDP interface. It also has a SPI bus for setting the threshold on the receiver.

Controller Board

The controller board connects all of the before mentioned components. It also features 2 push-buttons and a Future Technology Devices International Ltd. (FTDI) chip D300, responsible for the USB communication. These features are wired to and controlled by the FPGA.

3.2.2 System Software

The interfaces from and to the receiver board as well as the connection to the FTDI USB chip and the UDP peripheral are directly connected to the FPGA. This way, these communication lines can be configured and controlled by the FPGA.

The FPGA is programmed in VHDL through the Xilinx Integrated Development Environment (IDE) Vivado. Starting point of this work is a basic implementation on the FPGA, capable of single-hit measurements and fundamental UDP communication.

MicroBlaze

MicroBlaze is a micro controller developed by the company Xilinx. It is fully implemented in VHDL and therefore a so called soft-core CPU. It is a 32-bit RISC micro controller and uses, depending on its configuration, between 700 and 2000 slices.

The firmware for the controller can be programmed in C and is put into the block ram of the FPGA. Due to its flexibility, multiple different components can be connected through VHDL to it. Xilinx provides multiple different libraries which can be used in the firmware.

Advanced eXtensible Interface (AXI) Bus

The AXI bus is especially prevalent in Xilinx's Zynq devices. It is providing the interface between the processing system and programmable logic sections of the chip. It consists of one master and multiple slaves. Each slave has an address range and can provide functionality like writing or reading this memory. It is also possible to provide different commands over this interface.

Most of the components are interconnected via an AXI bus. This way, the MicroBlaze is able to control these and exchange data. Contrary to this, the USB dataflow is established without configuration or controlling by the FPGA. As soon as the encoder transmits data into the USB FIFO, this data is gathered by the external FTDI chip. This provides also higher data throughput because it is not limited by the MicroBlaze clocking.

Clocking

In the existing implementation, 3 clocks are used to control the system:

100 MHz
This clock is mainly used for the AXI bus and components connected to it. It is generated by an clock wizard IP by Xilinx.

625 MHz
This clock is used for the input lines from the receiver, described in 3.2.1.

125MHz
The LVDS lines are serialized by a factor of 10. Serialization is done by the Serialize Input Intellectual Property (IP) from Xilinx. Serialized data is presented in $\frac{1}{10}$ of the driving clock (625 MHz). Since the lines are sampled with dual data rate, the resulting clock is 125MHz.

$$c_{data} = \frac{c_{ser} \cdot 2}{s_{factor}} = \frac{625MHz \cdot 2}{10} = 125MHz \tag{3.1}$$

3.3 Histogramming

In this work, histogramming denotes the process of saving measured peaks into their corresponding distance bins for further processing. By accumulating multiple measurements of the same point in the scene, a histogram of received reflections can be built.

This section first describes the principle of the histogramming and its benefits. Next, different algorithm approaches are discussed and analyzed. In the last section, a confidence factor for measurements extracted out of the histogramming is proposed.

In this work, the histogramming is implemented in VHDL, in order to be able to compute it in realtime on the FPGA hardware. This is accomplished by generating a independent intellectual property in Vivado.

3.3.1 Principle

The existing system scans the scene in lines. With each shot line, 32bit-streams can be received from their corresponding APDs. Each APD measures the reflections of the scene at one specific point. When measuring the same point multiple times, by shooting the same line and reading out the same APD, these measurements can be combined to a histogram for this part of the scene. Since noise is assumed to have a uniform distribution in the signal, even the noise exceeding the threshold in power can be filtered out.

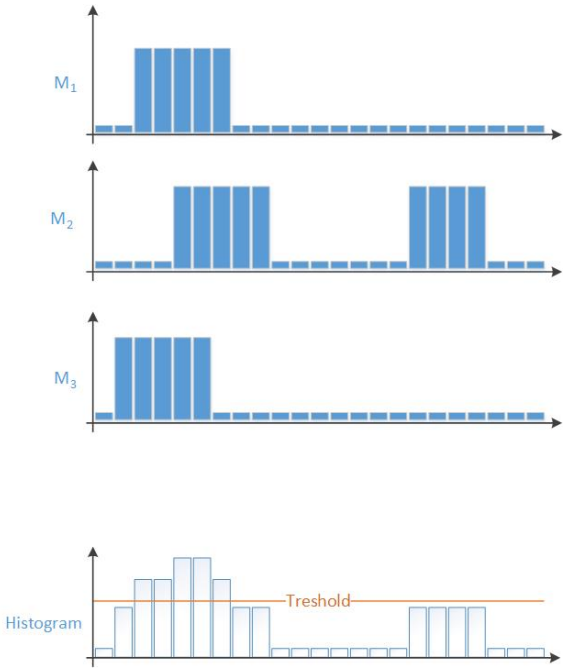


Figure 3.4: Illustration of 3 measurements being accumulated into one histogram.

In Figure 3.4, three already binned measurements are illustrated. Measurements M_1 and M_3 contain one reflection with a specific pulse width while M_2 contains 2 reflections. These have do be taken from independent shots of the same point in the scene. After accumulating all measurements of the same point and applying an arbitrary threshold, the most present points are extracted. As it can be seen, the second reflection of measurement M_2 is below threshold and therfore considered noise and ignored. Since the noise occurred as only measurement at this exact distance, it is unlikely it represents a valid object.

One drawback of the histogramming approach is the time delay induced. Since the measurements are taken independently, during the time between them, objects in the scene can move.

3.3.2 Algorithm Evaluation

For the histogramming, a suitable algorithm has to be chosen. The algorithms are analyzed by their memory consumption, needed computation time and its limitations. Primarily,

following three different approaches were analyzed:

Frame first

The first approach is called frame first. In this setting, the LiDAR will first measure all vertical lines in a scene. The whole scene will then be saved in memory before the next frame gets shot. This way, the desired number of frames is captured before the histogram is calculated over the saved data.

One advantage is that this approach is independent of the point shot sequence since it will analyze only complete frames. A drawback is the high memory consumption, since all measurements need to be preserved over multiple frames. The computation time is considerably low because the measurements can already be combined during sampling the next points.

Histogram first

Histogram first prioritizes the histogram over the frame. This means that the same points are shot multiple times before the next point in the scene is measured. It is then possible to calculate the histogram for one point very quickly. The used memory can then be reused for the next measurements.

With this principle, a faster histogramming is possible. One drawback is, that the frames per second (FPS) are dropping due to the time that is needed, before all points in a scene are shot. Also, a specific shot-sequence needs to be implemented.

Dynamic histogram

The final algorithm describes a combination of before mentioned principles. Each measurement of a point is considered individually, as in histogram first. For each shot, the measurements are saved in memory. When the desired number of measurements is reached, the histogram is calculated. Contrary to histogram first, the measurements are not accumulated and then saved in memory, but the shots are saved individually. So, after each shot, a new histogram can be calculated. When a point is measured again, the oldest values are discarded, the new ones saved and the histogram recalculated out of all measurements. With this dynamic approach the FPS drop is decreased.

The dynamic histogram is chosen to be implemented.

3.3.3 Memory Consumption

The chosen algorithm, dynamic histogram, needs the most memory compared to the other approaches. One measurement results in a bit-stream of data, consisting of *1* for light, *0* for no light. Every bit-stream needs to be saved as a whole multiple times. After recording the desired number of streams for one point, the histogram can be built by accumulating them and extracting the reflections in the scene.

The exact amount of memory which is required to store the data depends on different variables, as defined in the following list:

Number of lines L
The number of lines which have to be shot to build up one full frame. Since the used LiDAR system scans one line at the time, it needs to shoot the scene multiple times to get a full representation of the scene. The number of needed shots depends on the Horizontal Field of View (HFoV). This parameter itself depends on the maximum deflection of the used MEMS. In the existing system, the HFoV is set to 30 deg with 128 lines, in order to get a horizontal resolution of 12 cm.

Number of APDs A
Each shot measures multiple points in one line. The used system has 32 APDs built in, so each shot gives distance values for 32 different points. This variable is set constant in the resulting system, since this parameter is hardware specific.

Stream length S_l
Since the system must be capable of detecting multiple reflections of one shot, the whole stream needs to be stored. The length of the bit-stream depends on the maximum distance that is required. Each bit in the stream represents one sample of an APD at a given time. The time difference between two samples is defined through the sampling frequency f_s, which is set to 1.25 GHz. The desired maximum distance d_{max} is 500m.

$$c = 299,792,458\frac{m}{s} = 0.2998\frac{m}{ns} \tag{3.2}$$

$$\Delta t_{sample} = \frac{1}{f_s} = \frac{1}{1.25 \cdot 10^9} = 0.8ns \tag{3.3}$$

$$\Delta d_{sample} = \Delta t_s \cdot c = 0.2398m \tag{3.4}$$

$$S_l = \left\lceil \frac{d_{max}}{\Delta d_{sample} \cdot 2} \right\rceil = 4171 \tag{3.5}$$

As stated in equation 3.3, one bit in the bit-stream represents a time-difference Δt_{sample} of 0.8ns. This correlates to a traveled light distance of 0.2398 meters (equation 3.4). All samples, starting from 0m to the maximum distance of 500m, can then be represented with a stream-length of 4171 bits, as stated in equation 3.5. The denominator of 2 is used to take the light's way back into account.

Measurements per Histogram M_h
For one full histogram, multiple measurements of the same point need to be saved. The exact number of accumulated measurements will be evaluated by experiments with the resulting system.

For the resulting memory needed, a simple product of the stated parameters is calculated (3.6).

M_h	Memory		
	bit	*kByte*	*MByte*
1	17084416	2135.55	2.14
2	34168832	4271.10	4.27
3	51253248	6406.66	6.41
4	68337664	8542.21	8.54
5	85422080	10677.76	10.68
10	170844160	21355.52	21.36
15	256266240	32033.28	32.03
20	341688320	42711.04	42.71

Table 3.1: Needed memory for different numbers of measurements per histogram M_h.

$$Memory = S_l \cdot L \cdot A \cdot M_h \tag{3.6}$$

A compromise between noise reduction and performance needs to be found with a reasonable M_h value. As it can be seen in Table 3.1, the approach with saving the full stream with each measurement would use up much memory enormously. As stated in section 3.2.1, beginning with 5 M_h, it would use up more memory than provided in the Artix 7 as block ram.

A workaround using DDRAM is considered. The development-board from Trenz has 1GB DDRAM3 built in. But accessing and configuring this memory is much more complicated and slower than using block ram.

In the end, another way of storing and organizing the data is used. Since the meaningful part of a measured signal consists of the captured reflections, all other parts can be omitted. When the APDs do not recognize any light, the generated bitstream holds no information. More on the memory consumption after implementing this approach can be read in section 3.4.2.

3.3.4 Confidence

When the results of the histogramming are delivered to other systems, a confidence value for the measured points is desired. In this work, a new confidence value for LiDAR systems is induced.

The value is generated with respect to the measured peak width, its time of occurrence and its position relative to the other measurements of the same point. In an ideal system with a stationary scene and LiDAR, it is considered that measurements of the same point in the scene always result in the same received signal form. So the reflections of one object at one point in the scene occur always at the exact same moment. Therefore, when accumulating multiple measured peaks of the same point, they should all be of same length and at the same time index.

Due to object movements, ambient noise and other disturbance sources, the time of recording the reflection can differ at each shot. This means, the exact location of the object in the scene, which generated the reflection, can not be determined exactly. This uncertainty can be expressed by measuring the difference of the time indexes between the points in multiple measurements. When having 2 measurements of the same point, the more those peaks differ in time occurrence, the more uncertain is the position.

The peak width of the resulting accumulated peak is used for the uncertainty, as it can be seen in Figure 3.5.

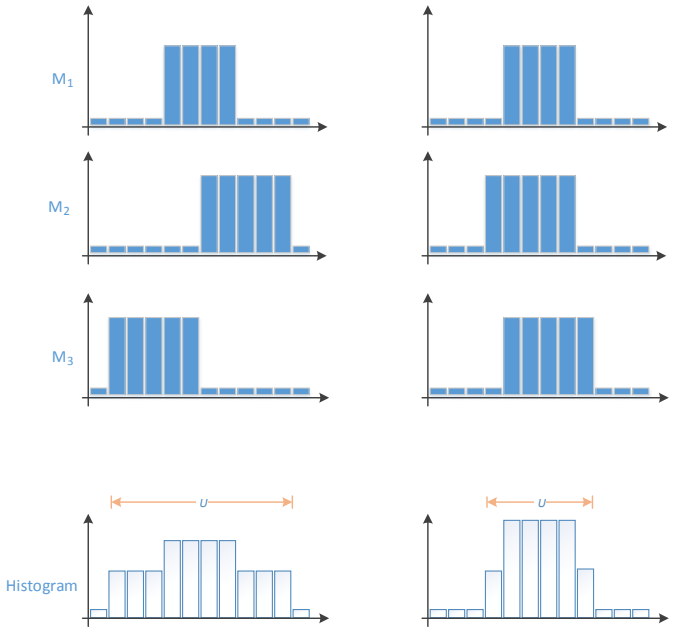


Figure 3.5: Left: Uncertain measurements with high time distribution. Right: Time distribution is narrow, therefore a lower uncertainty factor U.

In Figure 3.5, two histograms of two different points in the scene are illustrated. In the left, 3 measurements with high time distribution are accumulated, which results in a high uncertainty factor U. In the right, time distribution is much narrow, therefore U is lower.

Since the individual measured peaks in the measurements M_x can have different peak widths, these have to be taken into consideration too. The amplitude of a received peak, and therefore also its width, depends on the distance and the reflectivity of the hit material. So, objects further away generate a more narrow pulse.

3.4 Encoding

In this Section, a new approach for the encoding of LiDAR data is proposed and analyzed. First, the principles of the encoding is explained. It will explain the main attributes of backscattered signals. Next, the impact of the encoding on the memory consumption of

the system is calculated. For this part, the results of section 3.3.3 are used and extended.

The Encoding itself was manly designed and implemented in order to reduce the needed memory consumption in exchange of computation time and hardware resources.

3.4.1 Principle

The sampled backscattered signal of an emitted pulse is very sparse. Most of the samples are 0, since the light either has not hit a target or was already fully reflected. Due to the fact that only pulses produced by reflections of objects in the scene are of interest, an encoding exploiting these attributes is proposed.

As described in 3.2.1, the sampling of the reflected light is done by APDs and a comparator. This system does not provide any amplitude information, but has only a resolution of one bit. Reflections that have enough power to exceed the comparator voltage threshold, can then be seen as rectangular signal on the LVDS lines.

The only information which can be extracted from this signal, is the time of the rising edge, indicating the object's distance, and the length of the pulse. These two attributes need to be preserved by the encoding.

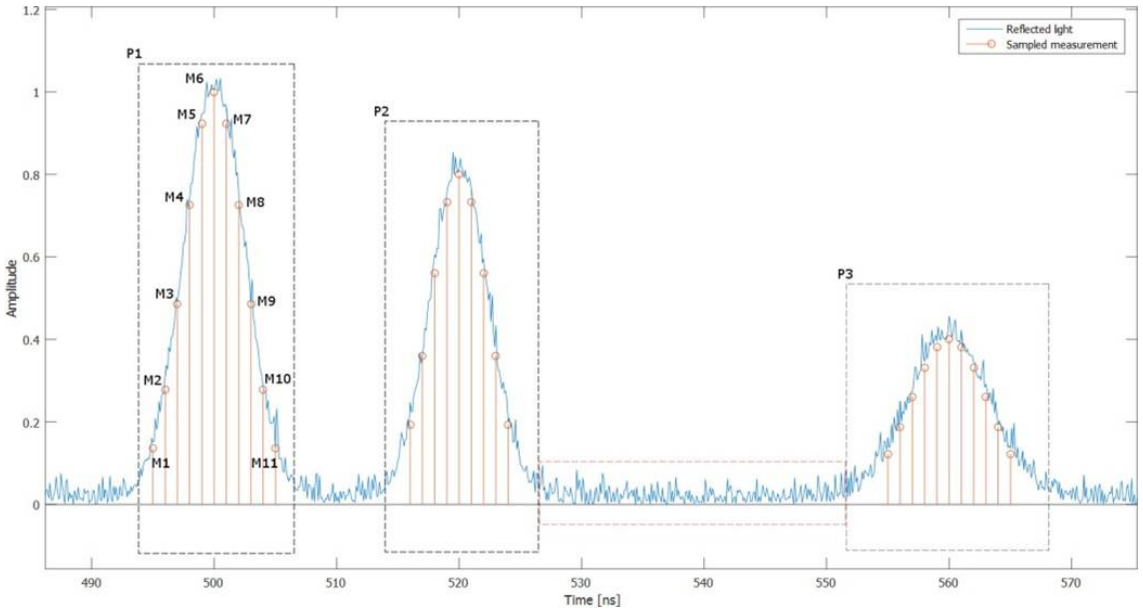


Figure 3.6: A simulated reflection signal with the encoding scheme. In blue, the captured signal is presented. The red stems represent sampled measurements. The rectangles P_1 to P_3 mark valid peaks which informations needs to be preserved. The rectangle between P_2 and P_3 marks the signal part which can be omitted.

In Figure 3.6, a sampled backscattered signal is illustrated. The blue signal is sampled with constant time resolution when it is above an arbitrary threshold. Three reflections are marked with P1, P2 and P3. Between two peaks, P2 and P3, a part of the signal,

which can be omitted, is also marked. By omitting this part of the signal, and other parts which are below threshold, the data can be reduced to a great extent. The idea behind encoding is to reduce the data needed to analyze while still preserving the information of the reflections.

When a measurement starts, a counter is started at the exact same time. Its current value will represent the time passed since the beginning of measurement. Each time, when a rising edge is detected, the current counter is saved as starting index of this pulse. When the next falling edge occurs, the current counter value is used as stop index. Start and stop index can then be further used as pulse definition and also to extract the pulse width.

3.4.2 Memory reduction

As already calculated in Section 3.3.3, the maximum counter index, representing the highest possible distance, is 4171. For each detected pulse, two counter values need to be saved. The equation for computing the memory consumption needs to be extended:

Measurement Index Bit-Width I_{bit}
Since a pulse can theoretically extend over the entire detection distance of 500m, the required bit width is defined for storing the maximum counter index of 4171. A counter with the resolution of the sampling rate (1.25 GHz) is used to determine the time index t_i for each sampled point.

$$I_{bit} = \lfloor log_2(counter_{max}) \rfloor + 1 = \lfloor log_2(4171) \rfloor + 1 = 13bit \tag{3.7}$$

13 bits are needed to represent the maximum index $counter_{max}$. Since this value needs to be saved into memory and also will be transmitted over USB and UDP, a bit-width with power of 2 is chosen. Therefore, the next greater fitting bit-width is 16 bits.

Pulse amount P
Each pulse is saved separately. The number of pulses that can be maximally saved within one measurement, will be limited by P. A suitable number has to be chosen after analyzing real measurements. Also noise has to be considered.

With the encoding and the additional parameters, the needed memory can be computed with Equation 3.8.

$$Memory = I_{bit} \cdot 2 \cdot P \cdot L \cdot A \cdot M_h = 32bit \cdot 2 \cdot P \cdot 128 \cdot 32 \cdot M_h \tag{3.8}$$

In Table 3.2, memory consumption for different settings of M_h and P are calculated. Even with a high number of $M_h = 20$ and $P = 10$, the block ram resources of the Artix 7 FPGA are not fully used.

M_h	P	Memory		
		bit	$kByte$	$MByte$
2	2	1048576	131.07	0.13
2	5	2621440	327.68	0.33
2	10	5242880	655.36	0.66
5	2	2621440	327.68	0.33
5	5	6553	819.20	0.82
5	10	13107	1638.40	1.64
10	2	5242880	655.36	0.66
10	5	13107	1638.40	1.64
10	10	26214	3276.80	3.28
15	2	7864320	983.04	0.98
15	5	19660	2457.60	2.46
15	10	39321	4915.20	4.92
20	2	10485760	1310.72	1.31
20	5	26214	3276.80	3.28
20	10	52428	6553.60	6.55

Table 3.2: Needed memory for different numbers of measurements per histogram M_h and number of pulses P. $I_{bit} = 16bit$, $L = 128$, $A = 32$

3.5 Communication

Communication channels to the FPGA are implemented for configuration and retrieving results from the system. Over Universal Asynchronous Receiver Transmitter (UART), it is possible to configure different parameters for the histogramming and the USB transfer. It is also used for debugging the implemented firmware.

The USB interface, provided by the FTDI chip, is used to transfer raw measured data as well as the encoded detected pulses. The data source can be selected with a command over the UART interface.

The ethernet interface is used for transferring the histogrammed results.

3.5.1 Ethernet

The used development board from Trenz provides a 100MBit ethernet interface. The Physical Layer (PHY) is integrated with a Media Independent Interface (MII). First, a hardware-only control of the MII was considered in order to be able to transfer the data as fast as possible. But since the used MicroBlaze is already fast enough to utilize the whole bandwidth and a software implementation of the IP-stack is more flexible, the open source stack light weight IP (lwIP) is used. This stack is specially made for embedded applications and has a wide set of features.

After the successful computation of the histogram of a line of points, the values are

written into a First In First Out (FIFO) buffer. This buffer is constantly read out by the MicroBlaze. As soon as enough data is available for one full UDP packet, it is sent as broadcast onto the network.

UDP is used because it is a stateless transmission protocol, unlike TCP it does not need to establish a connection with the endpoint beforehand. A drawback is that it can not be guaranteed that every packet is received by the target system, since UDP has no sequence value built in. It is also not possible to request a retransmit of a specific packet.

A broadcast is realized with setting the IP address of the receiver to 255.255.255.255. This way, every system in the local network can receive the LiDAR data and the firmware does not need to resolve a receiver IP to a corresponding MAC address. This saves time and therefore increases the throughput.

Header			Data				
marker	pkt_{cnt}	*line*	$distance_1$	$uncertainty_1$	- - -	$distance_{32}$	$uncertainty_{32}$
4 byte	2 byte	2 byte	2 byte	2 byte		2 byte	2 byte

Table 3.3: Data structure of one measurement packet. This packet contains data from histogrammed points of one line. Multiple measurement packets are part of one UDP packet.

One UDP packet consists out of multiple measurement packets. These packets contain information about the distance measured of a specific line and the uncertainty values for the points. In Table 3.3, the data-structure of one histogrammed measurement can be seen. It consists of 136 bytes of data and the fields can roughly be separated as header and data fields. Their purpose is described in the following list:

marker
: Each measurement packet begins with the fixed marker *0xFFFFFFFF*. This simplifies the process of parsing packets as well as detecting missing data. The marker value is chosen in a way that no other combination of field values can incidentally add up to it. The maximum distance value as well as uncertainty value is *0x104B104B*, since the maximum distance is 4171 (*0x104B*) (see Equation 3.5).

packet
: This value is a counter incremented with each measurement packet. It can be used to analyze and detect missing packets during transmission.

line
: The line in the scene to which the values correspond to. Lines are counted from left to right. With knowledge of the number of lines a scene is split, and the Vertical Field of View (VFoV), a 3D reconstruction is possible.

$distance_x$
: This value gives the ticks passed from beginning of the measurement until the reflection is detected. With knowledge of the sampling frequency f_s and the speed of light, the distance can be calculated. There are 32 distance values in the measurement

packet. The position of this value in the packet corresponds to the APD which has sampled this point.

$uncertainty_x$
: The corresponding uncertainty value to the previous distance value. Distance and corresponding uncertainty are always in a tuple right after each other, starting with the distance.

3.5.2 USB

The USB interface is provided through the FTDI chip FT600 on the controller board. The chip is a USB3.0 to FIFO bridge [FTD15]. The main hardware features are as following:

- Support for USB3.0 SuperSpeed (5Gbps), USB High Speed (480Mbps) and USB 2.0 Full Speed (12Mbps) transfer
- Available with either 16bit/32bit wide parallel FIFO interface
- Supports 2 parallel slave FIFO bus protocols, with data bursting rate up to 400MBps
- Supports multi-channel FIFO interface
- Up to 8 configurable endpoints (pipes)
- Built-in 16kB FIFO data buffer RAM

A 32bit wide parallel FIFO interface with 1 endpoint is used. The control logic is implemented in VHDL and programmed onto the FPGA. It fills up a FIFO buffer with captured data and controls the read cycles from the chip. The USB interface is used for raw data as well as encoded pulse information. Therefore, 2 FIFOs are used, so the source of the data for the FT600 can be selected and switched during runtime.

Raw data

Writing to and reading from the FIFO buffers is processed without any interaction from the MicroBlaze, unlike the UDP transmission process. Since the processing through the firmware with the clock of the MicroBlaze would be too slow this is necessary.

$$f_{PRR} = 40kHz \tag{3.9}$$

$$t_d = \frac{1}{f_{PRR}} = \frac{1}{40000Hz} = 25\mu s \tag{3.10}$$

$$M_b = APDs \cdot S_l = 32 \cdot 4171bit = 133.472kBit \tag{3.11}$$

$$D = \frac{1}{f_{PRR}} \cdot M_b = 5.339\frac{GBit}{sec} \tag{3.12}$$

The time between 2 measurements is defined by the pulse repetition rate (PRR). The frequency for the PRR is $f_{PRR} = 40$ kHz, therefore the time available for transmitting the data is $25\mu s$. For the raw data transmission, the full bit-stream of one measurement needs to be saved into a FIFO buffer and be read out by the FTDI chip. One measurement generates $133.472kBit$ when saving the full bit-stream. Therefore, the data needed to transmit per second can be calculated by multiplying it with the number of measurements per second, resulting in $D = 5.339\frac{GBit}{sec}$. This is not possible with the selected chip.

For reducing the data, the distance recorded is reduced to 200 samples (S_l), representing a distance of $d = \frac{c}{\Delta t_{sample}} \cdot 200 = \frac{299{,}792{,}458\frac{m}{s}}{0.0000000008\frac{m}{s}} \cdot 1000 = 74.95m$. By reducing S_l, the needed throughput is then $D = 256MBit$.

The data structure of the transmitted data packets contains the same header structure as the UDP packets. As it can be seen in Table 3.4, a marker, packet-counter and line is included. Their purpose is described in Section 3.5.1. In the data-section, each bit in one 32bit time field corresponds to a sample of the APD with the same index. For example, the second bit of the third 32bit word corresponds to the third sample of the second APD.

Header			Data				
$marker$	pkt_{cnt}	$line$	T_3	T_2	- - -	T_{19}	T_{20}
4 byte	2 byte	2 byte	32bit	32bit		32bit	32bit

Table 3.4: Data structure of one USB raw measurement packet. Each bit in a 32bit time field represents a corresponding sample of an APD_x.

Encoded data

For debugging purposes and verification of the histogram, also the encoded data needs to be transfered to an external system. Therefore, the encoded data is not only passed to the histogram logic, but is also saved into a FIFO buffer.

The data structure for one encoded packet can be seen in Table 3.5. The header is again the same as in Section 3.5.1, also the data structure is similar. The difference lies in the data field, since the encoder produces multiple points and the length of them is represented in 8bit. So, each APD measurement is represented by 3 points, containing a time index, marking time of occurrence, and the pulse length.

3.5.3 UART

The UART interface is provided by the MicroBlaze processor and the AXI UartLite IP from Xilinx. The Controller board then features a 5 pin connector for a connection to the PC. The UART connection has a baud rate of 115200 and 8 data bits. An interrupt is triggered in the firmware when new data is available on the interface.

This is used to control the system during runtime. It also provides additional debug data in form of printing out its current state. In order to minimize the needed reading

Header			Data													
			APD_1							APD_{32}						
$marker$	pkt_{cnt}	$line$	P_1		P_2		P_3		- - -	P_1		P_2		P_3		
			I	L	I	L	I	L		I	L	I	L	I	L	
4 byte	2 byte	2 byte	16bit	8bit	16	8	16	8		16	8	16	8	16	8	

Table 3.5: Data structure of one USB encoded measurement packet. Each measurement of one APD is represented by 3 points, containing time index and pulse length.

buffer, each command consists of one character only. Following commands are supported:

a, s With this command pair, the threshold of the histogram can be controlled. It basically sets how many pulses need to overlap when accumulating multiple measurements. If the accumulated signals exceed this threshold, it is considered a valid reflection. With a, the threshold is decreased, s increases it. The response should be the current threshold set after the modification.

y, x These commands can alter the number of measurements which will be accumulated to form a histogram. y decreases the number, x increases it.

d, f In order to switch the USB output between raw and encoded transmission, these commands can be used. d enables the encoded output, f the raw one.

3.6 PC Software

In this Section, the developed programs for visualizing, recording and debugging the data of the system are presented. Figure 3.7 illustrates the software, its dependencies and the communication channels. $USBtoPCL$ and $UDPtoPCL$ are used for processing the data directly from the USB and UDP interfaces from the FPGA.

The programs produce point cloud data in form of messages readable by the Robot Operating System (ROS). ROS has a built in Point Cloud Library (PCL) and can work with LiDAR data out of the box. *ROSFilter* can use the produced data and filter out points depending on different settings. All the data is then visualized with ROS Visualization (RVIZ).

3.6.1 ROS / RVIZ

ROS is a flexible cross-platform framework for writing robust robot software. It includes a selection of tools and libraries to simplify the development. First, a so called master node is started. This node handles all the upcoming traffic from other nodes and coordinates routing and messaging. Nodes can then publish messages on so called topics. A topic is defined by a string and the type of message it publishes. Nodes can also subscribe to topics and read messages which are published onto it.

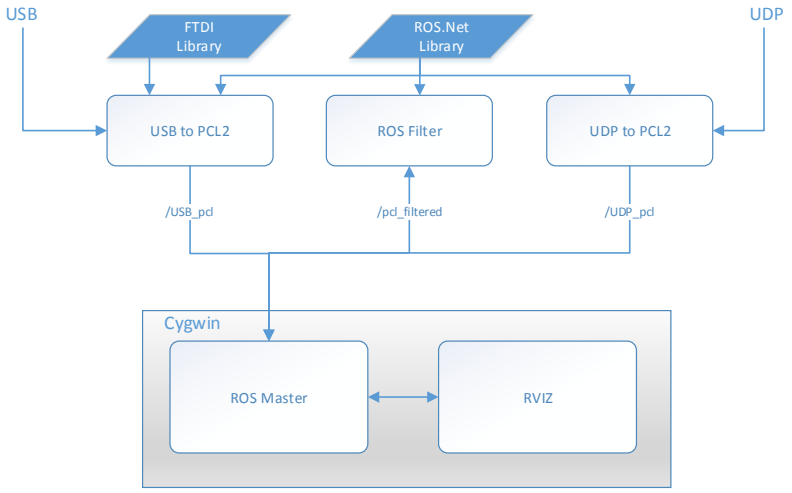


Figure 3.7: Illustration of he developed software components and their interaction through the ROS system.

The messages are transferred via TCP, therefore the master node can be run anywhere on the local network. The self written programs are using the messages from the PCL to describe and transfer the recorded pointcloud.

RVIZ is a stand-alone tool provided by ROS with a Graphical User Interface (GUI). It is configured to read the topics that are published by the programs. It then displays the point-cloud in a 3D representation which can be used to analyze the recorded data. Additionally, multiple point clouds can be displayed at the same time, which makes it possible to directly compare the encoded and histogrammed data.

In order to be able to run ROS on a Windows PC, Cygwin is used. Cygwin is an environment for windows with a library providing PASIX API functionality. Therefore, linux applications compiled in Cygwin can be run in Windows. After several tries to get ROS and RVIZ running in this environment, a precompiled version of ROS within an already provided Cygwin is used [Bri15]. This port of ROS includes binary releases of ROS Jade and Indigo.

Since the messaging mechanism of ROS relies on the network stack, the programs could be developed for Windows and still communicate with the master node in the Cygwin environment. A library is used that enables ROS functions support under Windows is used, the *ROS.Net* library [EM16]. It includes multiple message definitions and also the ability to subscribe and publish onto topics.

3.6.2 USB to PCL2

This program is responsible for processing the data provided by the USB interface. It first detects if the FPGA is plugged and establishes a communication channel. Next, it publishes the ROS topic */USB_pcl* to the master node. If any of the steps fails, it aborts and closes the program.

After establishing all necessary connections, it starts reading the data from the USB interface. The data is then interpreted and converted into the PCL message format. Each time, a full frame is collected, the message is then published onto the topic, so it can be visualized.

In order to read data from USB, the D3XX library is used. This library is provided by FTDI [FTD16] and is able to open a data channel, configure the FTDI chip and perform asynchronous read commands.

For encoded data stream, $USBtoPCL2$ provides the position of one point (x, y and z coordinates) and the pulse width. Raw measurement data can not be represented in this form, therefore it is ignored.

3.6.3 UDP to PCL2

For recording and interpreting histogrammed data from the system, the $UDPtoPCL2$ program is developed. It opens a socket and listens on an arbitrary port for UDP broadcast packets on the network. Concurrently it publishes the topic $/UDP\_pcl$ to the master node. If one of the processes, trying to establish a connection, fails, the program closes itself.

When packets are captured on the UDP interface, the data of the histogrammed measurements are extracted and interpreted. Since one UDP packet contains multiple measurements, the data must be split and processed individually. After collecting one whole frame, the measurements are converted into a PCL message and published onto the previous described topic.

The UDP program publishes PCL messages which contain points with x, y and z-coordinates, but also an additional parameter, called $confidence$.

3.6.4 ROS Filter

The last program developed is used for more detailed analysis of the captured data. It first lists all available topics from the master node. After selecting one topic, usually $/UDP\_pcl$ or $/USB\_pcl$, it is possible to filter out different points from the point-cloud and republish it. Therefore, the program publishes the topic $/pcl\_filtered$ to the master node at startup.

The program provides a GUI in which the different filtering options can be set. It also displays a processed data package counter and the widest point attribute seen. This information can be used to determine the performance and quality of the received data. It is possible to filter points by following attributes:

Point Attribute

With 2 sliders, the maximum and minimum value of a point attribute can be defined. Depending on the source, UDP or USB, the attribute represents the pulse width or the confidence of the point. This can be used to filter out points, where the real distance is uncertain, depending on their confidence. It also can be used to filter out pulses which have not the expected pulse width according to their detected distance.

Distance

Another pair of sliders can be used to define the maximum and minimum distance to display. It can be used to restrict the displayed area.

HFoV

It is also possible to restrict the HFoV. Again, maximum and minimum values can be set, which results in filtering the points out, which are beyond those values.

Chapter 4

Implementation

This Chapter is about the implementation details of this thesis. First the development tools and the work flow is discussed. After this, the encoder, the first module in the signal processing line, is presented. Next, the histogrammer module is explained. Finally, the needed communication channels and their implementation is documented.

4.1 Development

This section describes the tools which are used for implementing the requirements. Afterwards, the used workflow is presented. The development process produced multiple versions of the implemented modules, since bugs and improvements were discovered during testing.

4.1.1 Tools

Multiple different tools are used during development. The most important ones are described here.

Vivado Design Suite

The Vivado Design Suite [Inc16c] produced by Xilinx is a software suite for synthesis and analysis of designs in form of a Hardware Description Language (HDL). It can work with Verilog as well as Very High Speed Integrated Circuit Hardware Description Language (VHDL). Since the used FPGA is also produced by Xilinx, the support for this hardware is already integrated. It provides a programming interface to a connected FPGA as well as debug features for running processes on the device.

Vivado can also be used to create so called Block Designs, in which predefined logic IPs can be instantiated and connected through a GUI. This makes it highly intuitiv and fast to design with already existing IPs from Xilinx.

Vivado itself splits the development process of a design in different main process steps:

Design
In the first step, the system and its modules are designed. Vivado provides many different IPs ready to use from Xilinx. It also features an editor with syntax highlighting and the block diagram mechanism.

Simulation
Vivado supports multiple different HDL simulators. With them, it is possible to simulate the digital designs to pre-validate their functionality. For simulation, a testbench can be created, in which the module under test is instantiated. In the testbench, different signals can be defined and scenarios created to fully evaluate the module-under-test. Waveforms of the output-signals are then generated by the simulator. This step is important since it reduces the development time significant.

Synthesis
After validating the functionality of the modules, a synthesis can be performed. In this step, HDL designs are converted into networks out of the available resources. Therefore, the designs are presented as combination of Flip Flops (FF), Look Up Tables (LUTs) and other resources. After this step, runtime analysis and resource usage can be inspected.

Implementation
With the synthesized modules, the implementation step can be run. In this step, the design is placed onto the layout of the FPGA. New delay times can occur through placement, which Vivado will try to automatically keep under certain limits. The result of this step is a bit-stream which can be used to program the FPGA.

Program/Debug
Vivado also provides a way to automatically program a connected FPGA with the previously produced bit-stream. When it is programmed, Vivado also provides a way to trigger implemented debug cores and read out their results.

Xilinx SDK

The Software Development Kit (SDK) from Xilinx is used for creating the embedded application. It can be used to develop applications for FPGAs already containing a micro processor, like Zynq, but also for MicroBlaze soft-core systems. Multiple device drivers and libraries for the supported micro processors are included, as well as a JTAG debug console. The IDE is based on Eclipse 4.5.0 and CDT 8.8.0.

The Xilinx SDK is used to develop the firmware for the LiDAR system. In Vivado Design Suit instantiated Xilinx IPs can directly be addressed in the C code through the libraries and device drivers.

After compilation, the application is split into definition and binary files. These can then be placed onto a SD card. The LiDAR then boots off of the SD card. Alternatively, it is possible to directly program through the JTAG debugging interface, which reduces the development process.

Visual Studio

The desktop applications implemented in this thesis are developed with Visual Studio 2017 [Mic17]. It is an IDE which supports multiple different programming languages like C#, C++, Python and many others. It is mainly used for Windows applications and also features a debugger for such applications. Since Version 2017 15.4, it is also capable of handling CMake project structures.

For UDP and USB data processing, applications written in C++ are developed with Visual Studio. It is also used to wrap a ROS.NET C# library into a C++ structure. The functionality of retrieving and processing the data provided by the LiDR system is implemented in native programs in order to ensure high performance. C++ is used because it also leaves the option to compile it with a different IDEs for Linux systems.

Wireshark

Another tool used for debugging is Wireshark. In combination with pcap, an open application programming interface, it is possible to intercept, filter and analyze network traffic. Data streams are categorized and individual packets are split up into their different layers. This way, it is possible to identify problems with checksums, byte alignment and many other protocol specific flags. Wireshark can capture data on Ethernet as well as wireless interfaces.

In this work, Wireshark is used to verify and inspect the UDP packets from the LiDAR system. Hence it is not necessary to develop a receiving program just for debugging. It also is used to detect malformed packet structure and identify optimization possibilities.

4.1.2 Work-flow

In this Subsection, the work-flow used in this thesis is presented. The system and its requirements can be split up into two nearly independent parts. Since the communication protocol is already defined, FPGA and firmware development can be done separately from desktop application development.

Hardware/Firmware

Most of the time during development, not all of the components needed for a full working LiDAR were available. Since the MEMS mirror as well as the receiver board were also under development, their input is simulated. An arbitrary simulated scene is generated and saved as binary stream data, simulating input from APDs. With these streams, only a controller board with an attached FPGA is necessary to develop and implement the required signal processing.

The existing system was already able to receive single hit reflections and transmit data over an UDP interface. First, the TDC is replaced with the new designed encoder module. This module is encapsulated into an IP, which simplifies updating and simulation.

The module is designed apart from the whole other system components and verified in an isolated testbench. After verification of the basic functionality, the new model is integrated in the system.

The same steps are performed for the design and implementation of the USB interface and the histogram logic. Since the process of synthesizing and implementing a design is very time consuming, the main focus rests on fully simulating and verifying beforehand.

Additional debug cores are added to an implementation in order to analyze signals of the modules during runtime of a programmed FPGA. With these additional data, the design models are then again modified to consider more scenarios and prevent failures. The whole process is therefore iterated multiple times.

Software

Since the data structure of the expected packets from the system are known, the software can be developed without finished hardware. The first step is to write the USB receiver and evaluate its performance. Since the encoder module needs significant throughput, the main focus in this software is on performance.

First, the USB software only needs to establish a connection channel, read data and store it into a file for manual analysis. For evaluation, a simple hardware module is designed which produces USB data with an incremental counter value. This way, missing data and maximum throughput can be evaluated. After finishing the communication, data interpretation is implemented with a manually created data file, simulating valid measurements.

The UDP receiver first captures data of the existing system. With this data, the receiving and storing functionality can be tested. After verifying that no data is lost, also manually created data is used to implement the interpretation of measurements.

As soon as both programs are able to receive and convert measurement data, the visualization protocol to ROS and RVIZ is implemented.

4.2 Overall System Architecture

Figure 4.1 the system architecture with the designed hardware modules in the FPGA. The interfaces from and to the controller board as well as the connection to the FTDI USB chip and the UDP peripheral are directly connected to the FPGA. This way, these communication lines can be configured and controlled by the FPGA. Most components are directly connected with an AXI bus and controlled by the MicroBlaze. The micro controller can configure the system components through this bus and gather resulting data. The signal processing path consists of the encoder module and the histogram module. The encoder samples the raw data coming from 32 LVDS lines. It encodes detected reflections into multiple points and pulse width indexes.

The encoded data is then stored into the USB output FIFO buffer and also passed on to the histogram module. The USB output FIFO buffer is read out independently,

therefore the throughput is not limited by the AXI bus frequency.

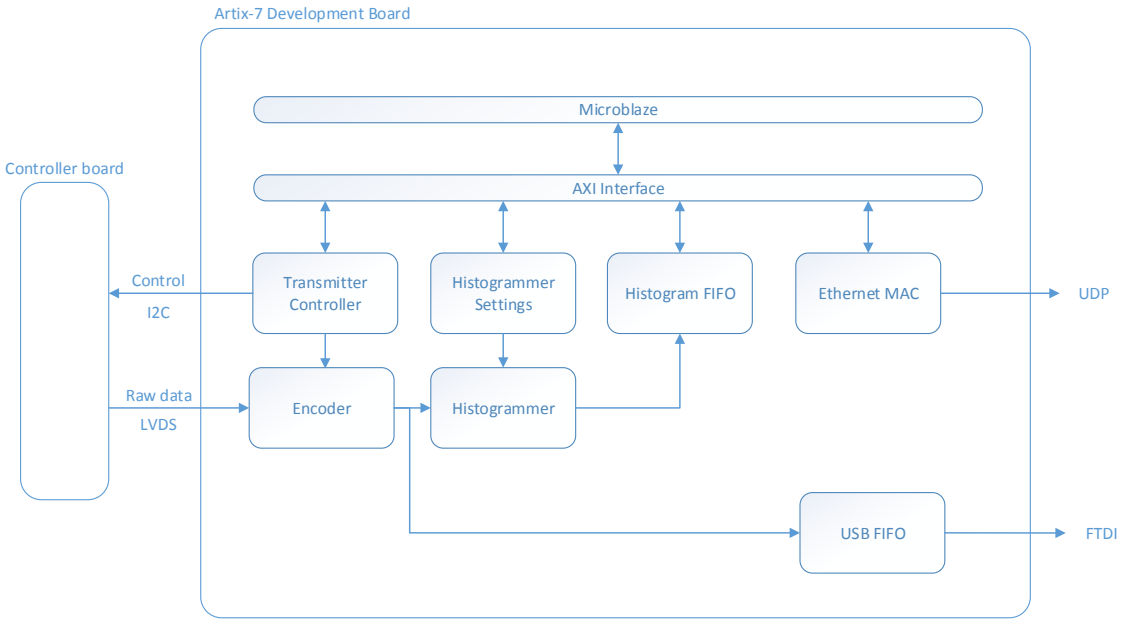


Figure 4.1: System architecture illustrated for the hardware modules in the FPGA. Most components are interconnected via an AXI bus and controlled by the MicroBlaze. The USB communication is independent of the micro controller.

The histogram module takes over the encoded data and saves it into the BRAM. As soon as enough measurements are collected, whereby the number is configurable through the histogrammer settings, the signals are reconstructed, accumulated and a threshold is applied. Detected reflections are then written as point indexes into the histogram FIFO buffer.

The MicroBlaze is constantly reading out the histogram FIFO and building UDP packet buffers. As soon as one packet buffer is filled, it transmits the packet over the ethernet MAC. The whole signal processing pipeline needs to be completed after 25 μs, since the measurement frequency is 40 kHz.

The desktop applications presented in Section 3.6 are started independently from the LiDAR system. As soon as data is available on either the USB or the UDP interface, the data is captured, interpreted and visualized.

4.3 Encoding

The encoder is the first stage in the signal processing line. It is used for data reduction and therefore allows the system to run with much less memory. The signal itself contains mostly no reflection and hence is not needed for later analysis. So it is possible to omit

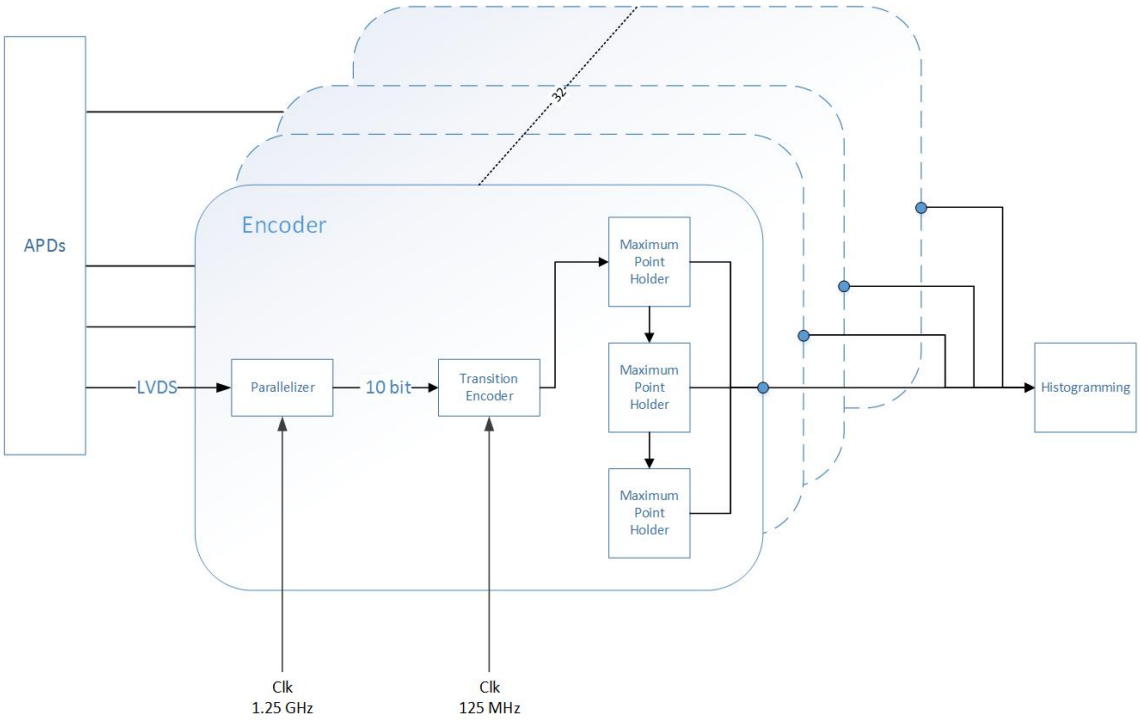


Figure 4.2: Overview of the encoder module.

this obsolete data and only focus on the received reflections.

An overview of the encoder can be seen in Figure 4.2. The raw LVDS from the APD-receiver are used as input. Each instance of an encoder handles one LVDS. So, for the whole system, 32 instances of this encoder are used. Received pulses are detected and stored with their corresponding start- and end-time. This data is then merged with the outcome of other encoder instances and used for histogramming. The merged data is also written into the USB FIFO buffer, used for USB transmission.

The Encoder itself can be split into 3 different submodules, which are discussed in the following subsections.

4.3.1 Parallelizer

As parallelizer, the LogicCore IP SelectIO (v 1.5) from Xilinx [Inc16b] is used. This IP can be configured with the SelectIO Interface Wizard. Following configuration options and settings for this system are used:

Data Rate Dual Data Rate (DDR). This way, the block samples the input lines on rising and falling edge.

I/O Signaling Type Differential LVDS 25. This specification is given by the receiver board.

External Data Width 1. Since each encoder works with data from one APD only, one input line is used.

Serialization Factor 10. With this option, the output is a 10bit bus with $\frac{1}{10}$ of the sampling frequency update rate.

The output signal of this block is called *data_in_to_device* and is, in this configuration, a 10bit wide bus. It also generates an internal clock called *diff_clock*, which is $\frac{1}{10}$ of the input sampling clock. The block is driven with a $625MHz$ clock for sampling and therefore generates a $125MHz$ *diff_clock*.

It is worth to mention that the SelectIO fills up the output bus in reverse. This means, that the Least Significant Bit (LSB) represents the first sampled value. In order to analyze the signal in terms of time of occurrence, this order needs to be considered.

4.3.2 Transition Encoder

The transition encoder converts the raw bit-stream into pulse information. The pulse information is presented as 16bit start index and 8bit pulse width. Input signals into the block and their use is described in following list:

clk_diff
The clock that is used for the input bit-stream. It is fed into the block from the SelectIO block.

enable
When this signal is logic high, the transition encoder starts to read and process the input bus.

dev_data
This bus holds the parallelized sampled data from the SelectIO block.

reset
With the reset signal, all internal counters and signals are reset.

Additional configuration parameters, like $serialization\_factor$ and the time to measure $tof\_measure\_time$, can be set as generics when instantiating this block. The $serialization\_factor$ defines the input bit-width of dev_data, while $tof\_measure\_time$ defines the bit-width of the internal counter as well as the number of clock-cycles that are required by the transition encode for measuring.

As output, two different bus signals are provided: tof_enc_ind and tof_enc_val. In combination, these buses provide the index and width values of the detected peaks. The maximum number of peaks able to be detected in one 10bit measurement is given by an alternating signal. This means, in case of 10bit samples, the signal 1010101010 would hold 5 peaks, each peak with the width of 1. Therefore the bit-width of the output buses can be calculated as follows:

$$tof\_enc\_ind_{bw} = 16 \cdot \left\lceil \frac{serialization\_factor}{2} \right\rceil \tag{4.1}$$

$$tof\_enc\_val_{bw} = 8 \cdot \left\lceil \frac{serialization\_factor}{2} \right\rceil \tag{4.2}$$

When the *enable* signal changes from low to high, an internal counter tof_abs_index is started. With each rising edge of *clk_diff* the counter is incremented by 10 and reflects the current measured distance. Additionally, the input bus *dev_data* is analyzed bit by bit. Therefore, a loop is implemented which tests each bit. Figure 4.3 illustrates the loop

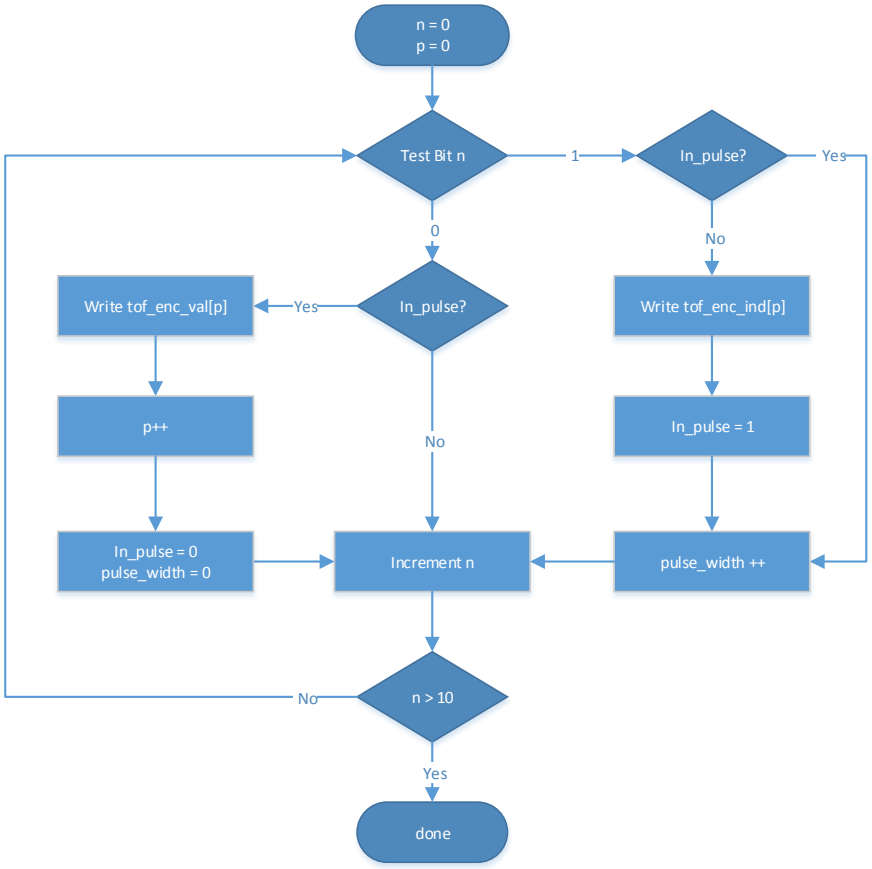


Figure 4.3: Flowchart of the transition encoder algorithm. It analyzes a 10bit input stream and detects pulse start indexes and their width in it.

processing the 10bit samples. It begins with resetting the internal signals to 0. n represents the index of the bit to analyze and p a running counter of detected pulses. Only two signal states are kept between clocks. These are *in_pulse*, which basically represents the state of the previous analyzed bit, and *pulse_width*. If the bit under test is 0 and also *in_pulse* is 0, then no peak is detected, n gets incremented and the loop continues. If the bit is 1, representing a starting pulse ($in\_pulse = 0$), the current index is written to tof_enc_ind at offset p. The index is the sum of n and tof_abs_index. After writing the current index, the signal *pulse_width* is incremented and *in_pulse* is set to 1. In each following iteration, every high sample bit increments *pulse_width* again. The next transition from a 1 to a 0 sample bit then completes the pulse. *pulse_width* is then written to tof_enc_val at offset p, completing the pulse information. p gets incremented, so the next detected pulse will

not overwrite the result. Since *pulse_width* as well as *in_pulse* states will be preserved between 2 clocks, also pulses split over two or more 10bit sample streams can be correctly detected.

The whole processing loop is implemented in VHDL and completes and computes the output within one clock cycle of the $125MHz$ clock. When tof_abs_index reaches the set $tof\_measure\_time$, an output signal *done* is set high for one clock cycle. With these signals, other blocks can detect the end of the measurement.

4.3.3 Maximum Point Holder

The next block in the signal processing line is the Maximum Point Holder (MPH). This block is implemented in order to detect pulses with the highest *pulse_width*. The block takes the buses from the transition encoder block as input. As output, it provides one 16bit measurement index and one 8bit width value. Additionally, it has also output buses with the same width as the input buses, so multiple MPH can be cascaded.

Internally, the block holds one pulse information. Initially, this is set to all 0. When it is enabled and a rising edge occurs on the input clock, the block sorts the pulses in both input buses by the corresponding *pulse_width* in tof_enc_val. If the highest *pulse_width* on the input is higher than the internally stored pulse information, the stored pulse gets replaced. Additionally, the previous stored pulse gets written into the pulse buses, so the next block can compare it with his stored value.

After each clock cycle, the block holds the pulse information with the highest *pulse_width* to which the input buses are set to. Three MPH blocks are cascaded in the implemented system, which adds an addition signal processing delay of 3 clock cycles. One major challenge in the implementation is the tight time constrictions in sorting the incoming pulse information.

4.4 Histogrammer

The histogrammer block is the main component of this thesis. It is implemented as own IP in Vivado. The main purpose of this block is to take the encoded pulse information, store it in Block Memory (BRAM) and accumulate it with previous measurements of the same scene. When the accumulated pulses exceed a given threshold, this pulse is considered a valid reflection and gets written into a FIFO buffer. As already discussed in Section 3.3, the dynamic histogramming approach is implemented. Therefore, each measurement is stored and replaces the oldest one. When the required amount of measurements for one histogram is reached, the signals are accumulated and analyzed. From this point forward, each new measurement triggers a histogram computation.

Figure 4.4 illustrates the instantiation interface as well as its input and output signals. In order to work properly, a FIFO buffer as well as an external BRAM module need to be connected to it. The FIFO buffer is later used for the transmission of the resulting data to the MicroBlaze, the BRAM module is used for storing the measurement data. The main input signals are defined in following list:

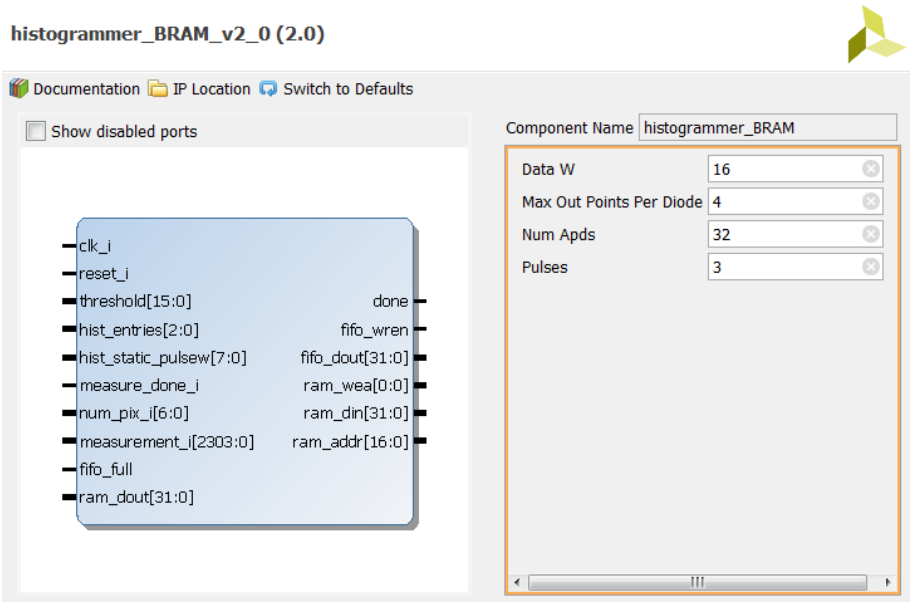


Figure 4.4: Interface of the implemented histogrammer module including input/output signals as well as generic component options.

num_pix_i
This signal gives the line number of the current measurement. Since the LiDAR system illuminates the scene in lines, this signal is a reference to the 3D position of the resulting points.

measure_done_i
When the encoder is finished with extracting pulse information out of the sampled bit-stream, it sets this signal to high. It is used to trigger the histogram computation.

measurement_i
This bus holds all the pulse information from the encoder block. It is split into $pulses \cdot 16 \cdot APDs$ bit index values and $pulses \cdot 8 \cdot APDs$ bit pulse_width information.

In Figure 4.5 an overview of the modules used for the histogram computation is presented. Besides the BRAM and FIFO buffer for data storage, two main modules are used: *Sorter* and *Intersector*. These modules are used for the histogram computation and controlled by the state machine of the histogrammer. Two sorters are used in order to sort the start- and end-index values independently. Also, two intersectors are instantiated in order to accelerate the computation by parallelizing the processing.

In the following Subsections, the internal workings of the histogrammer are explained. First, the implemented computation process is explained which gives more detailed insight into the used blocks. Next, the data structure used in the BRAM is outlined. And in the last subsection, the connection of these components and the state machine of the histogrammer, which makes use of them, is described.

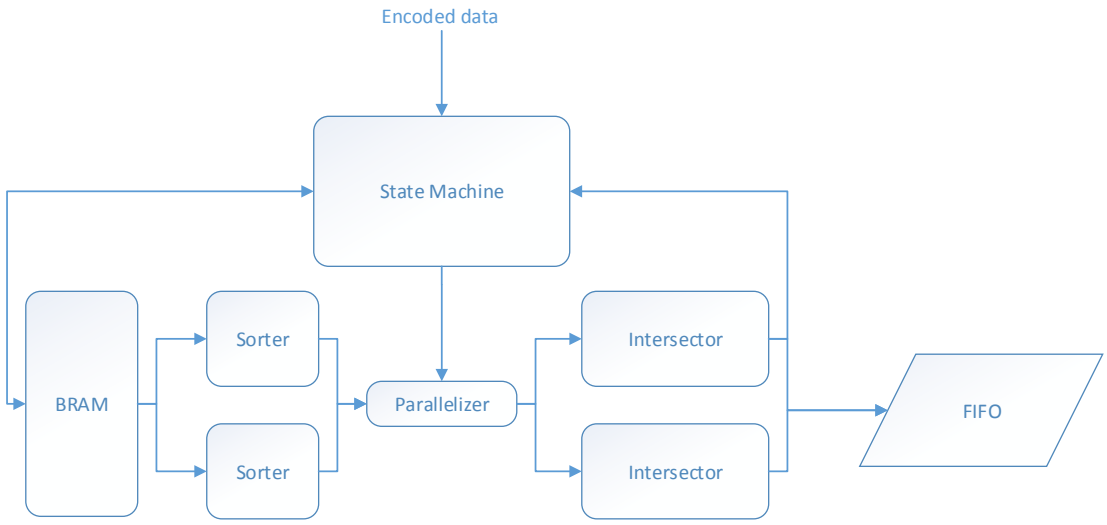


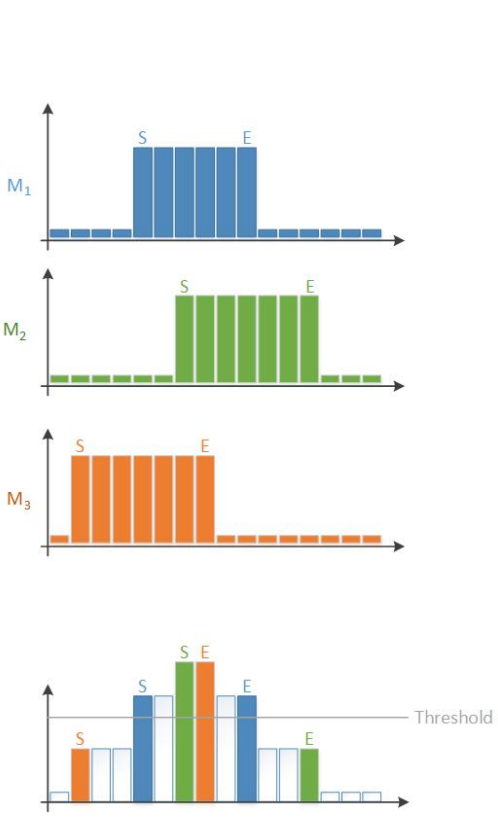
Figure 4.5: Overview of the inner structure of the histogrammer module. With receiving the encoded data the state machine starts and controls the computation of the histogram.

4.4.1 Computation

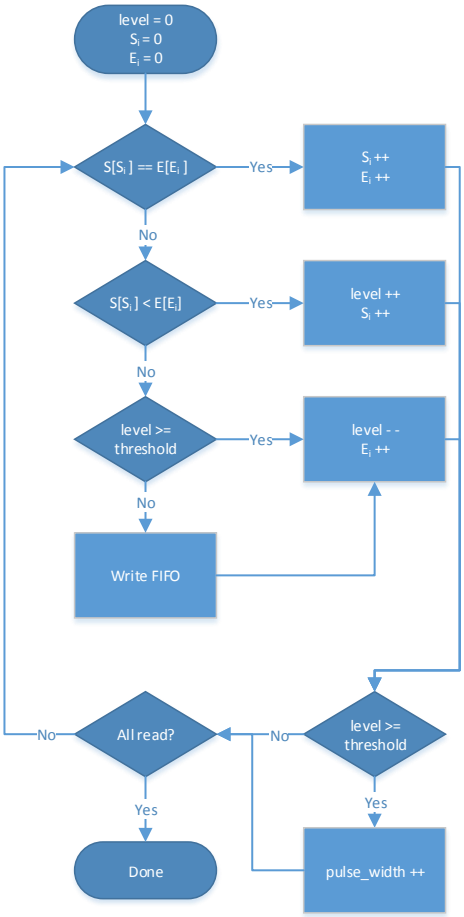
One approach for computing the intersections of the pulses is to iterate through all possible indexes and accumulate the pulses recognized at this distance. But this would take one clock cycle for each index, which would be too slow. Instead, since the start and end index of each recognized pulse is known, this information is used to reduce the time needed.

Figure 4.6a presents sample measurements, their accumulation and a set threshold. The start and end indexes of the pulses in the measurements $M_1 - M_3$ are labeled with S and E. When accumulating, it can be noticed that the level is only modified when a pulse starts or ends. Otherwise it would keep the same value. It does not matter, to which pulse the start or end labeled sample belongs to, as far as the level is concerned.

In Figure 4.6b, the flowchart of the used algorithm is presented. It requires one list of sorted start and one list of sorted end indexes of all pulses. They are sorted by time of occurrence and stored in list S and E. The elements in the list can be accessed by the indexes S_i and E_i. At first, the elements in the list with the current index get compared. Depending on which type of change occurs first, the level gets increased or decreased. Also the index for the corresponding list gets increased. If both, one start and one end, happen at the same time, the level is not modified and the index of both lists gets increased. After adjusting index and threshold, these get compared. If the level exceeds or is equal to the threshold, the parameter $pulse\_width$ gets increased. After this, it is checked if all start and end samples are read, otherwise the loop continues. One special condition is when the detected overlapping area is considered finished. This occurs if an end index gets counted ($E < S$) and the level would get decreased beyond threshold. Therefore, the data of the measured pulse gets written into the output FIFO buffer. The pulse information includes $pulse\_width$ as well as the start index $start = E[E_i] - pulse\_width$.



(a)



(b)

Figure 4.6: (a) Sample measurements with highlighted start and end samples for accumulation. (b) Flowchart of the intersection evaluation algorithm.

With this implementation, the runtime of the intersection detection is drastically reduced. Considering that one iteration of the loop takes one clock cycle, the runtime I_r of this approach can be calculated with:

$$I_r = pulse\_number \cdot 2 \cdot measurments \tag{4.3}$$

4.4.2 Data structure

A data structure with focus on performance and compression is implemented. The Block Memory Generator 8.3 [Inc16a] from Xilinx is used to instantiate the BRAM module. It is a Single Port RAM with 32bit read and write width. By disabling the primitive output register, the read latency is only 1 clock cycle. Figure 4.7 illustrates the layout implemented onto the BRAM memory space. The data for each line shot is placed into a different

Line	Measurements								Metadata	
0x01	M_1					M_2		M_{15}		
	APD_1			APD_2	APD_{32}	APD_1 - APD_{32}	...	APD_1 - APD_{32}	$H_{current}$	H_{active}
	P_1	P_2	P_3	P_1 -P_3	P_1 -P_3					
	32 bit	32 bit	32 bit	96 bit	96 bit	3072 bit		3072 bit	16 bit	16 bit
0x02	M_1					M_2		M_{15}		
	APD_1			APD_2	APD_{32}	APD_1 - APD_{32}	...	APD_1 - APD_{32}	$H_{current}$	H_{active}
	P_1	P_2	P_3	P_1 -P_3	P_1 -P_3					
	32 bit	32 bit	32 bit	96 bit	96 bit	3072 bit		3072 bit	16 bit	16 bit
⋮										
0x80	M_1					M_2		M_{15}		
	APD_1			APD_2	APD_{32}	APD_1 - APD_{32}	...	APD_1 - APD_{32}	$H_{current}$	H_{active}
	P_1	P_2	P_3	P_1 -P_3	P_1 -P_3					
	32 bit	32 bit	32 bit	96 bit	96 bit	3072 bit		3072 bit	16 bit	16 bit

Figure 4.7: Memory layout for storing multiple measurements per shot line.

memory offset. Since the LiDAR system illuminates the scene with 128 lines, the maximum offset is $0x80$. For each line, the measurements M_1 to M_{max}, in the implementation M_{15}, are saved. Additionally, 32bits of metadata can be stored. This metadata is used for storing the last saved measurement offset as well as how many active measurements are already stored for this line.

One measurement M includes all received pulse information for each APD $APD_1 - APD_{32}$. Each APD on the other hand contains the three pulses $P_1 - P_3$ extracted by the encoder block. The pulse information consists out of 16bit start and 16bit end index of the pulse. The pulse width can then be determined by subtracting the start index from the end index.

By storing the APD values of the same measurement successively, writing of the information is kept simple for the histogrammer block. The input buses from the encoder are only needed to be piped into the memory, since they already contain the information for all APDs in one line. Further the memory offset for each line simplifies the addressing of the memory block. The required memory space can be calculated with following equation:

$$Memory = (32bit \cdot P \cdot APDs \cdot M + Metadata) \cdot Lines \tag{4.4}$$

For the configuration of the implemented LiDAR with $P = 3$, $APDs = 32$, $M = 15$ and $Lines = 128$, the resulting memory consumption is 737.792 $kByte$.

4.4.3 State machine

The internal logic of the histogrammer block is implemented as state machine. The steps and transitions are illustrated in Figure 4.8. When the encoding block is finished and valid data is provided on the input bus, the state machine begins with reading out meta data for the currently shot line. The metadata includes the number of already stored measurements for this line, as well as an running index pointing to the latest saved measurement. In the next step, the histogrammer updates the BRAM with the new captured measurement. As soon as enough measurements are stored, the oldest one is automatically replaced, since the pointer to the latest measurement is increased and points to already used memory. Only when enough measurements are already stored, the histogrammer continues with the computation, otherwise it is set back into the *IDLE* state.

The computation process begins with writing out the available information to the FIFO buffer. This contains the marker *0xFFFFFFFF*, a packet counter and the line number (see Section 3.5.1). The next step is to read out the data from the BRAM module. All pulses $P_1 - P_3$ out of all measurements $M_1 - M_{15}$ stored for one APD are read. The start indexes and the end indexes of each pulse are separately stored in a list. These lists are then used to find the intersections of the detected pulses (see Section 4.4.1). When this step is completed, the detected pulses, or *0x00*, are written into the output FIFO buffer and the measurements for the next APD are read and analyzed. This loop is iterated until all 32 APD measurements are processed.

The loop iterating over the APD measurements is the time critical component in the implementation. In order to reduce the needed runtime, the loop is partially parallelized. The sorter and the intersection computation module are implemented in separate blocks. Two instances of the sorter block are generated, where one sorter is used for the starting indexes, the other one for the ending indexes. As sorter, an open implementation [Vas16] is used, which only needed as many clocks as elements are in the list. Next, also two instances of the intersection computation module are generated. Each module needs both lists in order to analyze the measurements, but also require longer time to finish than the sorting module does. Therefore, while intersector 1 is running, the next APD measurements are read and fed into the sorter module. So, while intersector 1 is running, intersector 2 could already start with the next measurements. In the end, this technique is able to nearly halve the needed time for one full histogram.

4.5 Communication

Three different communication channels are implemented in order to transfer the resulting data and for debugging the system. The encoder block directly puts out its encoded pulse

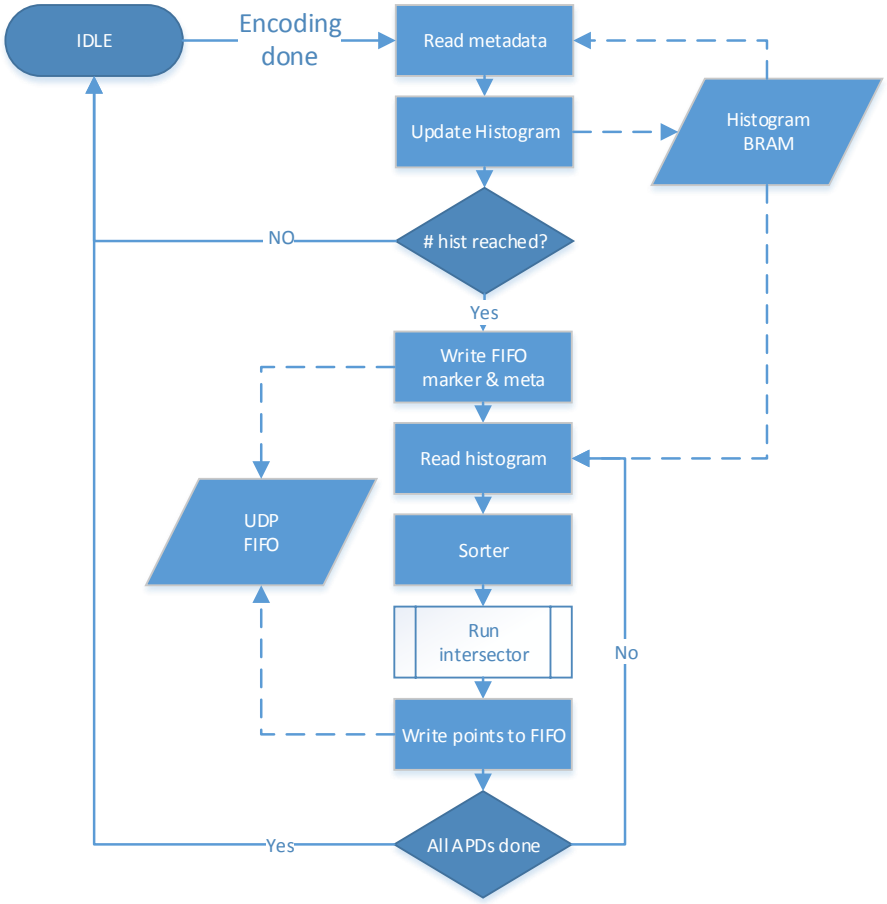


Figure 4.8: Flow diagram for the histogram state machine.

information over a USB interface, connected to a FTDI FT601 [FTD15] chip. The control of this chip is implemented in a separate block. Additionally, an Ethernet interface is used to transfer the histogram results with UDP. Contrary to the USB interface, the data transferred with UDP is provided by the MicroBlaze firmware. Finally, an UART interface is used to control and debug the system during runtime. These interfaces and their implementation are described in the following Sections.

4.5.1 Ethernet

The Ethernet PHY transceiver used on the Trenz Micromodule Artix-7 XC7A200T is a chip manufactured by Texas Instruments TLK106. This chip is controlled by an AXI Ethernet Lite MAC IP by Xilinx. For the communication between EMAC and PHY, an additional IP, MII to RMII v2.0, is used. The PHY transceiver is responsible for converting the control-signals to the corresponding signals on the Ethernet port. The AXI Ethernet Lite MAC enables controlling the Ethernet PHY very easily over the implemented AXI interface. This interface is connected to the internal micro-controller, the MicroBlaze.

In order to correctly assemble a valid UDP packet and control the Ethernet Lite MAC, the LWIP-library is used. Light Weight IP is an open source IP stack and provides a lightweight IP-stack implementation which can be easily used in C. This stack is specially crafted for embedded systems and needs very little memory.

For the data storage, the FIFO Generator (13.1) [Inc17] from Xilinx is used. This FIFO buffer provides the endpoint for the histogrammer module and is read by the MicroBlaze. The settings used for instantiation can be seen in Table 4.1.

Setting	Value
Implementation	Built-in FIFO
Width	32bit
Depth	4096
Empty Threshold	339
Read Clock Frequency	100 MHz
Write Clock Frequency	125 MHz

Table 4.1: FIFO buffer settings for the Ethernet transferring FIFO.

This FIFO has asynchronous write and read clocks, since 2 different parts of the implementation need to access it. The histogrammer module uses the differential clock from the encoder therefore it writes the data with 125 MHz. The MicroBlaze, on the other hand, is clocked with 100 MHz. As implementation, Built-in is chosen, so the FPGA's BRAM resources are not used for the FIFO. The BRAM resources are mostly needed for the histogrammer. The depth of the FIFO buffer is high enough to hold multiple histogram calculation results.

The throughput is one major problem in the default implementation of the LWIP stack. The intended usage in the C driver is to first create an *netif* struct and define the IP address, network mask and gateway IP address. After this, a buffer *payload* is

instantiated and filled with the data to send. As last step, the function *udp_sendto* is called. The parameters for this function include the network interface struct filled before, as well as the *payload* buffer, the IP address of the target and the target port. This function then starts a process which builds up an IP packet. During the process, an ARP request is created in order to get the MAC address of the target, since it is required for the IP packet header. Multiple buffers are created, spliced and concatenated in order to build the packet. Also, checksums of the payload are calculated and written into the header fields. Finally, the resulting IP packet is then transmitted to the Ethernet PHY.

The whole internal process of building up an IP packet needs so much time that, even at constant transmitting *0x0* buffers, the throughput is limited to 30 Mbit. In order to increase this limit, the buffer creation of the different needed headers are implemented beforehand:

UDP Header
The UDP header contains source port, destination port, data length and a checksum. Both ports as well as a packet length can be predefined and filled into a buffer. The checksum would be calculated over the header fields as well as the data, so it cannot be precomputed since the data is not known beforehand. But since the checksum is optional, it is be set to *0x0*.

IP Header
All of the fields in the IP header are known beforehand. Since the checksum in the header is only over the header itself, it can be also calculated beforehand. So, another buffer containing the IP header is created.

Ethernet Header
With changing the destination IP to a broadcast address, also the MAC destination address *FF:FF:FF:FF:FF:FF* is known beforehand. Therefore, no ARP requests are required anymore and the Ethernet header could also be precomputed.

Since the buffers are precomputed, they can be used for every following packet without the need of recalculating any information. After gathering the histogram data out of the FIFO buffer, the IP packet is already complete and can be transmitted with the LWIP mechanism. Therefore, the function *XEmacLite_Send* was used. It takes a pointer to the network interface to be used, a pointer to the data buffer and the length of the buffer as parameters.

With this changes, the throughput is maximized to 100 MBit and only limited by the speed of reading out the histogram FIFO buffer.

4.5.2 USB

For the USB interface, the FTDI chip FT601 was used. The chip is a USB3.0 to FIFO bridge [FTD15] and the control signals are directly connected to the FPGA. The encoding module produces already correct formatted packets in the FIFO buffer. The buffer then only needs to be read by the chip and transferred to the requesting PC. For instantiating

the buffer, the FIFO Generator (13.1) from Xilinx is used. The settings for the generated FIFO buffer can be seen in Table 4.2. Similar to the UDP FIFO instance, it is defined as a Built-in FIFO with asynchronous read and write clocks. The writing encoder module is clocked with the differential clock of the Xilinx SELIO module. The FT601 chip is clocked by an external clock of 100MHz.

Setting	Value
Implementation	Built-in FIFO
Width	32bit
Depth	8192
Empty Threshold	5
Full Threshold	8175
Read Clock Frequency	100 MHz
Write Clock Frequency	125 MHz

Table 4.2: FIFO buffer settings for storing and transferring USB data.

A module is implemented which controls the data transfer to the FT601. When a PC is connected over this interface, it can initiate a data transfer with the provided API calls from FTDI. As soon as a data request is issued, the chip starts to read out of the connected FIFO buffer in the FPGA. In order to read from the connected FIFO, the signal *lidar_ren_o* is driven high. The delay of reading one 32bit word out of the buffer is 1 clock cycle. After this time, the read data is applied onto the data bus *data_i*. The signal *lidar_empty_i* is set to high from the FIFO buffer when the last entry is read and no further reading should be done. Additional to the read signals to the FIFO buffer following control signals for the chip are used in the VHDL implementation:

TXE_N
: This signal is provided by the chip. When it is pulled low, a transaction can be performed. A high state indicates a full internal chip buffer and no write transaction can be performed.

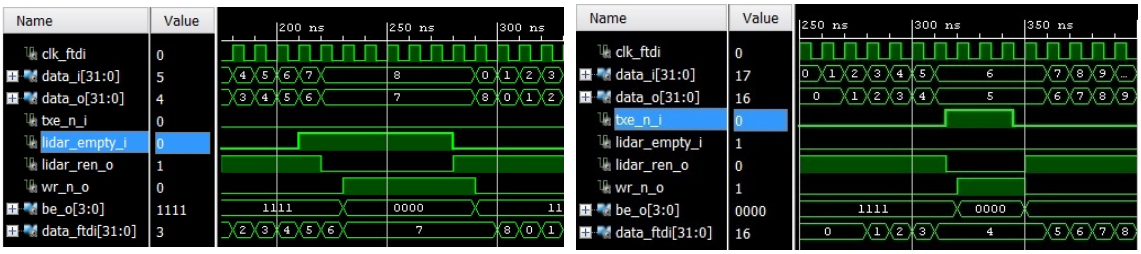
WR_N
: In order to start a write transaction, this signal needs to be pulled to a low state.

BE When a write transaction is performed and data is provided on the data bus, this signal defines which bytes are valid and should be written.

DATA
: The signals on this bus are used in a write transaction.

One major problem while implementing the module is to assure the correct synchronization between the chip and the read FIFO data. Since the read signal to the FIFO buffer and the receiving of the current data has 1 clock cycle offset, it can happen that the chip is already full while the next data is already read. In Figure 4.9 and Figure 4.10, four of the challenging scenarios are simulated and printed. For the simulation, the

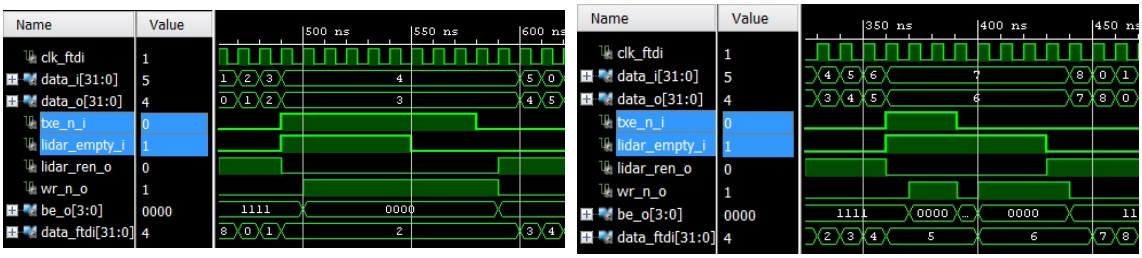
FIFO buffer is filled with an increasing single digit counter, which repeats itself. *data_i* represents the incoming data from the FIFO buffer, *data_o* is the data providing to the FT601 chip. *data_ftdi* represents the successfully written internal buffer of the chip.



(a) (b)

Figure 4.9: (a) Signal waveforms in case of an empty FIFO buffer. (b) Signal waveforms in case of full internal FT601 chip memory.

Figure 4.9a shows the signal waveform for the case that the FIFO is empty. When the *lidar_empty_i* signal is set from the buffer, the read signal is set to 0. As it can be seen on the signal *data_i*, two additional words out of the FIFO buffer are read. This is due to the read delay and the time that is needed to change *lidar_ren_o*. Therefore, the write signal to the chip is hold for 1 clock cycle longer than the FIFO buffer is read. When *lidar_empty_i* is back at 0 again, the write process is continued. Another stop scenario is illustrated in Figure 4.9b. In this case, the internal memory in the FT601 is full and therefore the signal *txe_n_i* is pulled to high. Immediately, the read signal to the FIFO buffer is pulled low, since no data can be written. The already read words need to be cached in the module itself. When *txe_n_i* is pulled back to low, indicating an empty chip buffer, the cached data is instantly transmitted, concurrently to the next read command.



(a) (b)

Figure 4.10: Signal waveforms in case of concurrent FIFO buffer empty and chip memory full scenario. (a) shows the signals when FIFO buffer empty signal is high shorter, (b) when it is high longer then the chip memory full signal.

The more complicated scenario, when both conditions apply at the same time, is simulated and illustrated in Figure 4.10. There is a significant difference when one of the conditions lasts longer than the other one. The case, when the buffer in the FT601 chip is longer occupied than the FIFO reading from is empty, is presented in Figure 4.10a. In this case, the signals need to behave as in Figure 4.9b and the write process needs to start as

soon as the chip buffer is available again. In Figure 4.10b, the FIFO buffer empty signal is longer pulled to low than the transaction signal is pulled high. Here, as soon as the chip buffer is available, the cached data is written to it. After this, the write process is discontinued again until the FIFO buffer is filled with data again.

4.5.3 UART

The UART interface is used for configuring the system during runtime as well as getting debug information. The firmware can configure the histogrammer block with the histogramm_settings block shown in Figure 4.11. It is controlled over the AXI bus and allows to set the number of measurements per histogram, the threshold as well as a static pulse width. The static pulse width is mainly used for debugging purpose and was not used in the resulting system.

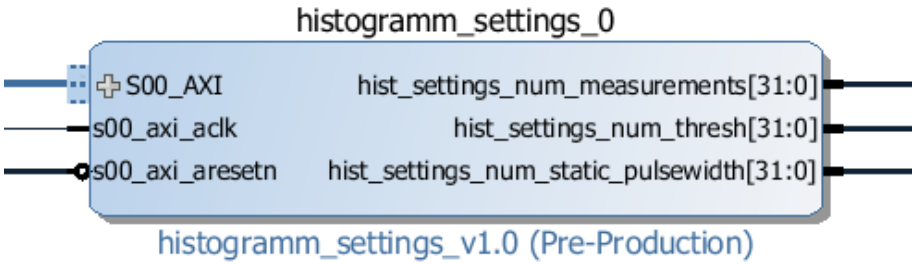


Figure 4.11: Instantiation interface of the histogram settings block.

The block itself provides read and write commands over the AXI bus for these parameters. The number of measurements has a built in maximum value of 15, since the hardware implementation limits it. For receiving the data on the interface, the function *XUartLite_Recv* from the XUartLite library is used. For writing into the settings block, a self written wrapper for AXI component drivers is implemented.

4.6 Software

In order to receive the data of the system, three different programs are implemented. All of these use the *ROS.NET* library in order to communicate to the ROS network. Depending on their functionality, they publish different topics, as can be seen in Figure 3.7. In order to publish point-cloud messages, the message *sensor_msgs/PointCloud2* [ROS15] is used. The definition of the format can be seen in Table 4.3.

With the parameters *width* and *height*, a 2D structure for the points can be defined. Since no priority order of the points is required in this system, *height* is set to 1. *width* is then adjusted to the number of visible points in the specific calculated frame. The next attribute *fields* is an array of field definitions. Each field definition includes a type,

Type	Name
uint32	height
uint32	width
PointField[ ]	fields
uint32	point_step
uint32	row_step
uint8[ ]	data

Table 4.3: Packet format of a PointCloud2 message in ROS.

a name and an offset. For each packet the fields x, y, and z are defined as *float32* type with corresponding name string. The offset defines the order of the points, therefore x has an offset of 0, y of 32 and z of 64. Depending on the data received from the LiDAR system, additional fields like *pulse width* and *confidence* are appended. *point_step* defines the length of a point in bytes, *row_steps* the number of points included in a packet. Finally, the raw point information is written into *data*.

4.6.1 Receiving

In order to maximize the throughput on the receiving end, the software developed uses the ping-pong buffer technique. Therefore, two buffers of the same size, A and B, are created and initialized. When starting receiving, the data is first stored in buffer A. When this buffer is full, it is used to process the data in it. Concurrently, the receiving process is continued filling up buffer B. After finishing processing the data out of Buffer A, it is free to be used again.

With this technique, a continuous receiving can be accomplished. Both implemented programs are using this method. In order to receive the data from the interfaces provided by the LiDAR system, following methods are used:

UDP

For receiving on the Ethernet interface, a socket was used. At the beginning of the program, a socket on the LiDAR port 55056 is instantiated. The socket then listens on the broadcast-address of the current network. When one receiving buffer is full, a new thread begins to analyze the data. Packet-counter as well as UDP counter are extracted in order to determine any packet loss. Additionally, the amount of data received per second is calculated. After processing, all received points are converted into the PointCloud2 message format.

USB

In order to be able to receive the data from the LiDAR USB interface, the library provided by FTDI [FTD16] is used. For detecting the USB device the API call *FT_Create* is used. After successful execution, the command channel to the chip is open. With another FTDI API call, *FT_ReadPipe*, data available in the chip is read. Again, this data is then stored in one of the ping-pong buffers. When processing

the data, the packet counter of the LiDAR encoder module is inspected. Warning information is printed when a packet loss is detected. The extracted information out of the USB data is then converted into PointCloud2 message format and transmitted onto the ROS network.

4.6.2 ROS Filter

The third implemented program is used to filter points out of the point-cloud resulting from either USB or UDP data. The program has a graphical user interface in which the specific filter rules as well as the topic to be filtered can be selected.

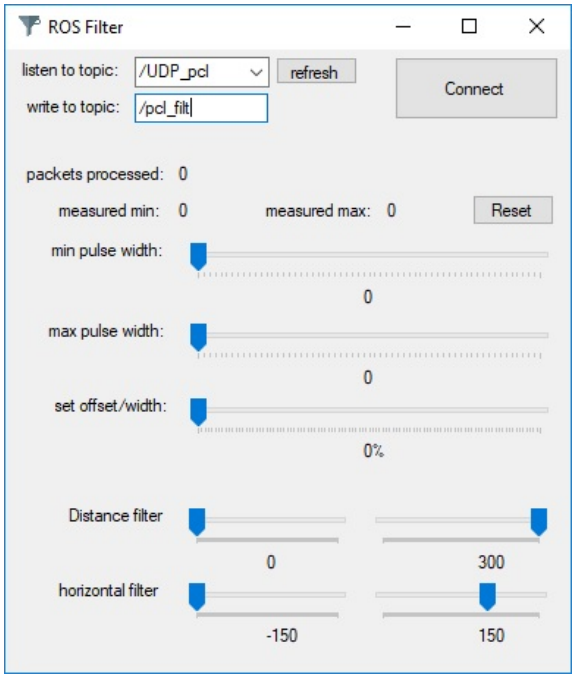


Figure 4.12: Graphical user interface of the ROS_Filter program.

Figure 4.12 shows the graphical user interface of the developed program. In the top, the ROS topic which should be filtered is selected. This list of selectable topics is automatically read from the running ros core. In order to reread the list, the refresh button can be used. Next, the topic on which this program should publish its results needs to be filled out. After completing the information, the process can be started with the *Connect* button. As debug information, the packets processed as well as the minimum and maximum measured peak width are then displayed. These counters can be reset with the included *Reset* button. The sliders and their function are described in following list:

pulse width
: With these two sliders, it is possible to filter points lower or higher than the selected limit.

set offset/width
: This filter defines the object position in respect to its pulse width. If the slider is set to 50%, a pulse with 4 ticks width will be placed on y-axis for 2 ticks further back. This option is included to extend the debug-functionality.

Distance filter
: With these two limits, the maximum and minimum distance of points relating to

the y-axis can be defined. All points out of these limits are discarded.

horizontal filter
Also these two sliders limit the horizontal field of view. This is used for cropping the displayed area.

In the program, first the received data from the selected topic is extracted. All the included points are then filtered by the criteria selected in the user interface. If one point is not within the limits, it is discarded. After filtering, the remaining points are again combined into a PointCloud2 message format and retransmitted again.

Chapter 5

Results

This Chapter presents the results of the proposed real-time signal processing system for one bit LiDAR. First, the implemented design is inspected. The used resources in the FPGA as well as slack in the clock path are described. Next, the performance of the system implemented into the prototype is discussed. The system is placed into one room with a second prototype, which generates light pulses. This increases the noise level and is used to test the histogramming feature. Finally, features extracted out of reflected light pulses are examined. The features are gathered by inspecting different measurements on different materials, but in the same environment. The used materials differ in reflectivity and transparency and are placed at specific distances.

5.1 Implementation

In this Section, the implementation results are discussed. The Xilinx Vivado 2016.2 software tool is used for synthesis and implementation of the hardware design. The target system for creation of the FPGA hardware bit-stream is the Xilinx Artix 7 platform. The bit-stream is then programmed through the software tool Xilinx SDK onto the FPGA platform.

5.1.1 Utilization

The logic designed in VHDL is converted to netlists consisting out of FPGA resources during synthesis. After this step, also slack is calculated. This gives an prediction of signals getting processed too slow to be available on time for the next block. After successful synthesis, the design is implemented on the FPGA. Due to limited resources, routing and placing, the modules again can be composed out of different resources. After implementation and bit-stream generation, the final used resources are presented by Vivado.

In Table 5.1, the used resources of the whole resulting system are listed. With 80.35% the IO connections are the most used resource in the system. This results from the con-

Resource	Utilization	Available	Utilization %
LUT	31544	133800	23.58
LUTRAM	1751	46200	3.79
FF	29967	267600	11.20
BRAM	166.50	365	45.62
DSP	3	740	0.41
IO	229	285	80.35
BUFG	15	32	46.88
MMCM	3	10	30.00
PLL	1	10	10.00

Table 5.1: Used resources of FPGA implementation.

troller board connections, including LVDS lines from the receiver board, SPI connections and other. Second, the global buffer (BUFG) resource is used by 46%. A BUFG is used for sending the clock across in the clock network within the design. Since the design uses multiple different clocks and is spread over most of the FPGA area, the clocks need to be buffered frequently. Next, the block RAM (BRAM) resource is used up by 45 %. One of the major causes is the histogrammer module, which uses up 78 BRAM modules in a configuration with 15 measurements per histogram. Other resources like look up table (LUT), flip flops (FF) and digital signal processing slices (DSP) are used up by the MicroBlaze and other used IPs.

In Figure 5.1, the placement view of the Vivado Design Suite is presented. In this view, blue cells mark occupied FPGA resources, not highlighted spots show available ones. Originated from the connection pins left and right from the FPGA, Vivado begins to place and route the synthesized design. The more area gets occupied, the less placement options for different functionalities are possible and it can happen that the signal lines get too long for a specific clock. Therefore, the so called *slack* occurs, which describes the time a signal needs from one block to the next referring to the sampling clock. For example a negative *slack* of 3 ns states that the required signal transition happens 3ns after the next clock edge. In the implemented system it was possible to avoid any negative slack.

5.1.2 Throughput

The implemented LiDAR system provides an UDP and an USB interface. The LVDS lines from the receiver are sampled by the encoding module with 1.25 *GHz*. The resulting system samples 2500 bits from the input lines. Therefore the whole sampling process needs 2 μs in theory. The measured time needed between rising trigger signal and *encoder_done* signal is 2.024 μs, since the maximum point holder modules also need to process the encoded data. Two different times between the *encoder_done* signal and the *histogram_done* signal are measured. In the first case, the histogrammer module has not enough measurements of the sampled point to calculate a histogram. Therefore, it only describes the time of saving the signal points. This is measured with 860 ns. In the second case, the histogrammer

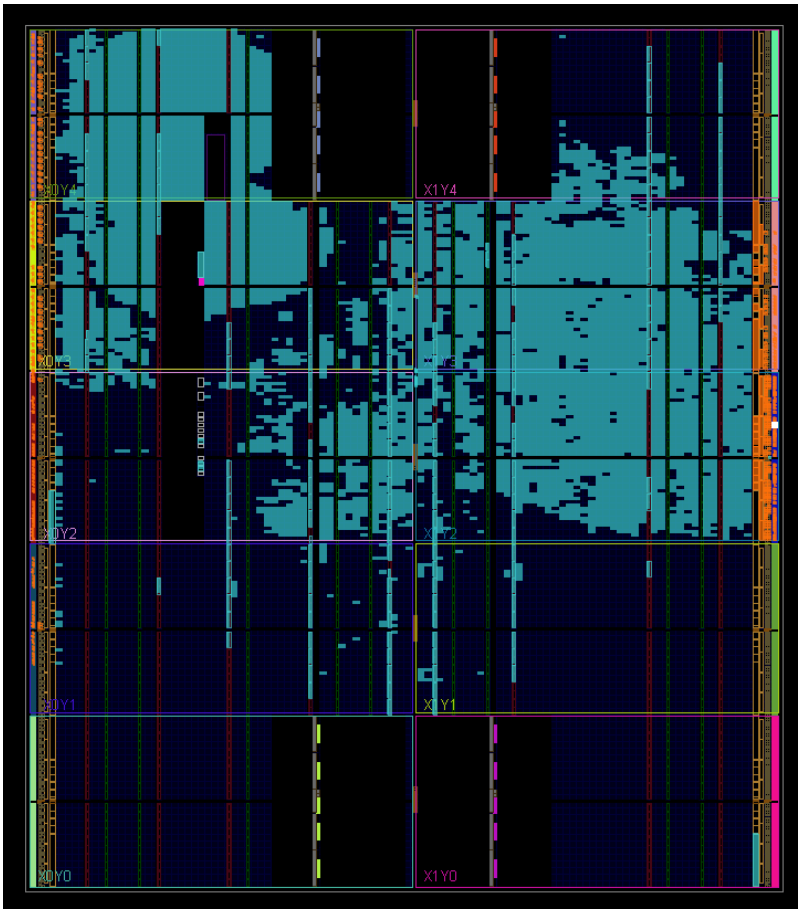


Figure 5.1: Implemented design utilization from Vivado placement view. Blue cells mark used resources like flip flops (FF) or BRAM.

module computes all intersections and writes the found points out to the FIFO buffer. Here, the measured average time needed is 15 μs, the longest measured time with multiple intersections is 17.06 μs. In the scenario, where no reflections are captured at all and every point is 0, the time needed by the histogrammer is 14.004 μs. The highest Pulse Repetition Rate (PRR) achievable by the prototype is 40 *kHz*, so the shortest period of time between two measurements is 25 μs. Therefore, real-time signal processing is given, since the modules finishes latest after 19.084 μs.

For the USB throughput measurement, the dataflow into the developed *USB_to_PCL2* is analyzed. When directly connected to the FT600 chip during measurement, the program starts capturing encoded data into the ping-pong buffer. The throughput measured is in average 94MBit. Due to the packet line numbers, missing packets are also detected. During an one hour measurement, 0.01 % of packets get lost. This may be due to the windows' scheduling process, which sometimes suspends the USB receiving program.

UDP throughput is measured with the windows' internal network monitor. During measurement, the network traffic is around 27 % of the network link's maximum throughput. This is equal to 27MBit on the 100MBit Ethernet link.

5.2 Time-of-Flight Processing Measurements

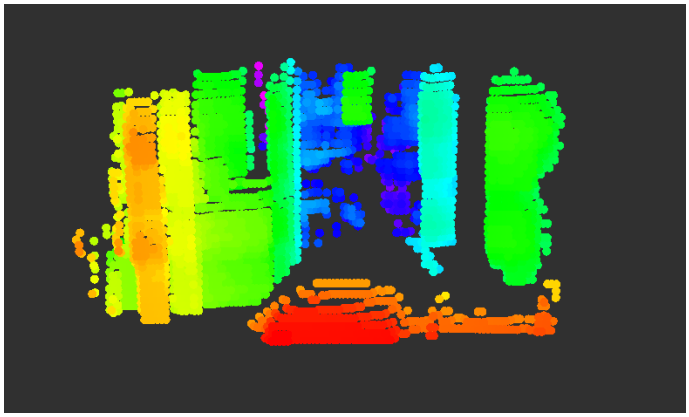
In order to determine the quality of the features extracted from the LiDAR data, different experimental setups and materials are used. For a better visual verification of the histogramming, another LiDAR system is used to produce noise in the same light spectrum. Confidence is evaluated with a target of two different reflectance properties, while the pulse width is analyzed with a variety of objects. The experiments and results are discussed in detail in the following sections.

5.2.1 Testing environment

A company office is used as testing environment. Since the prototype is at the time of testing not eye-save in terms of laser power, the tests are performed inside. In Figure 5.2a, the prototype LiDAR in the test scenery is displayed. The LiDAR system can be seen placed on a table in the bottom half of the picture. It faces away from the camera. On the other side of the room, a reference object is hung from the ceiling. The reference object has the size of a DIN A4 normed paper and is a diffuse white target. It is placed 6.629 meters away from the LiDAR prototype. The camera right in front of the LiDAR is a thermal camera and is used for monitoring the temperature of the system during measurements.

In Figure 5.2b, the point cloud representation of the testing environment is illustrated. The color encoding of the points represents their placement on the y-axis. The color scheme is from *RGB(255,0,0)* to *RGB(0,0,255)*. So, the red dots are near the LiDAR system, blue dots are the furthest away. The red half circle near the LiDAR represents the reflections of the chair which can also be seen in Figure 5.2a. Another prominent object is the white reference target, which can be seen in the middle of the point cloud colored green.

(a)



(b)

Figure 5.2: In (a), a RGB-picture of the environment seen from the LiDAR perspective is shown. (b) shows the point-cloud of the scene.

All the following measurements and experiments are executed in this environment. The background illumination of the room is not exactly determined. It was a cloudy day and the measurements where taken during the afternoon.

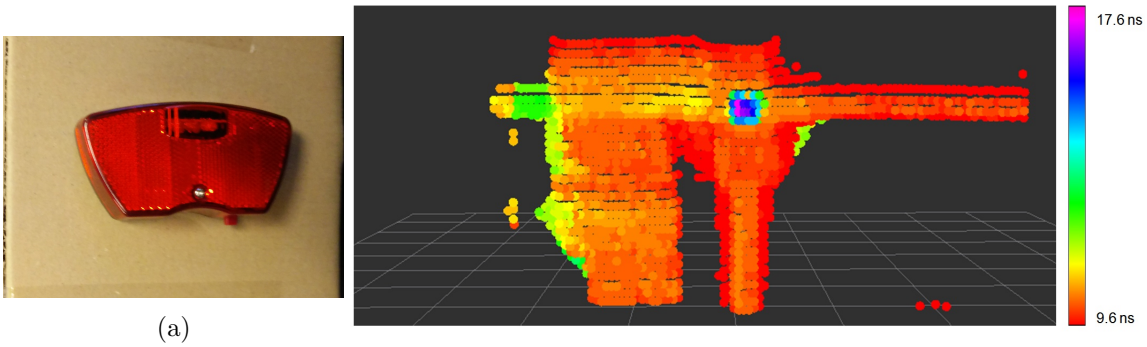
5.2.2 Pulse Width

The pulse width information is extracted with the encoding module. This information is then transmitted over the USB interface and captured by the *USB_to_PCL* software. After converting into PCL format, the data is visualized with RVIZ. Additionally to the point-cloud position, each point has a pulse width attribute. This attribute can further be used to filter or classify objects. In this work, the reflectance of objects is analyzed. Further, the pulse width is used to filter false detections out of the point cloud.

Reflectance

Since the amplitude of the received reflection correlates with the reflectivity of the objects, also the pulse width has a weak correlation with it. In order to proof this assumption, a high-reflective object is measured and the resulting points are color encoded with respect to their pulse width. Figure 5.3a shows the used object, a bike reflector. This is a retro-reflective material, meaning it has very high reflectivity and reflects light back with the same incident angle.

The object is held in the left hand of a pedestrian facing the LiDAR system. The target is held in a distance of 4 meters away from the prototype. In Figure 5.3b, the point cloud representation of this scene is illustrated. The color encodings of the points represent their



(a) (b)

Figure 5.3: Pulse width representation of a bike reflector. (a) shows the object itself, (b) the point-cloud of a pedestrian holding the object.

pulse width. As it can be seen, the bike reflector can be identified in the right side of the point cloud very easily. It produces much wider reflections than its surroundings. It seems that the distance is too small to prevent crosstalk to other receiving APDs, therefore a vertical stripe of reflections can also be observed. Further, a horizontal reflection stripe is visible, this may be due to not enough focus of the laser beam when measuring a different part of the scene. Since retro reflective material reflects the light back to its source, it is also detected as object in line of sight.

Filtering

Since the sampling frequency as well as the emitted pulse width is known, the pulse width of the received reflected light can be used for filtering. Depending on the distance of the light traveled and the reflectivity of the hit target, the amplitude and therefore the pulse width of the reflection decreases. Within a certain close range, these effects can be ignored, as described in [HGF13].

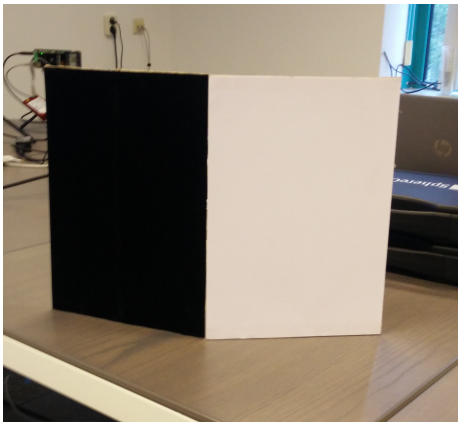
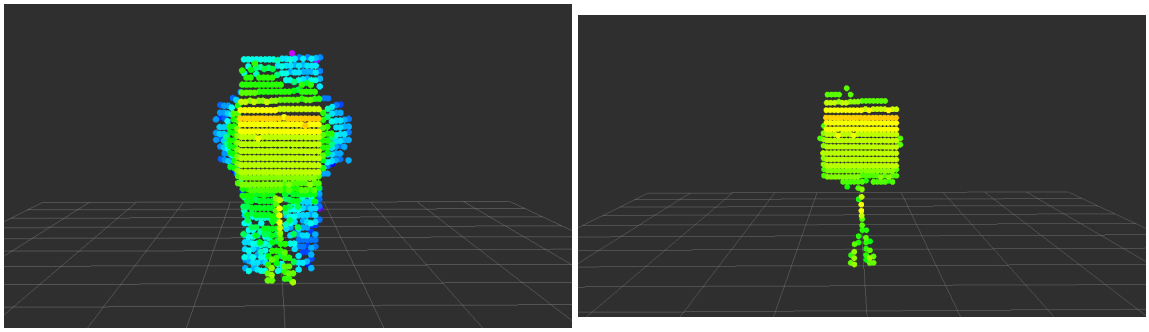


Figure 5.4: Used target consisting of black felt and white paper to analyze confidence feature.

Figure 5.4 shows the used object to verify the extracted pulse width. A cardboard with the dimensions of 42cm x 29.7cm is cut out and covered with two different materials. The left side is cloaked with black felt, the right side with a white sheet of paper. The target is then mounted onto a tripod and placed at 3.75 meters distance to the LiDAR prototype.



(a) (b)

Figure 5.5: Point cloud representation of carboard target on a tripod. (a) without filtering, (b) with filtered pulses below 18 clock ticks.

In Figure 5.5a, the unmodified captured 3D point-cloud is presented. Again, the points are color encoded with respect to their distance to the LiDAR. As it can be seen, especially on the edges of the object, non existing target points are falsely detected. It is assumed that this is due to the light scattering on edges of the object. With the developed filter program *ROS_Filter*, points with lower confidence factor of 18 clock ticks are filtered out. The result can be seen in Figure 5.5b. The LiDAR system transmitter emits light pulses with an average duration of 15 ns. Since the light reflections are sampled with 1.25 GHz, the expected full pulse reflection has a duration of 18.75 clock ticks.

This sort of filtering can be used to improve the quality of the resulting 3D point-cloud representation and exclude false detected points.

5.2.3 Confidence

The confidence feature is examined with the data provided over the UDP interface. The histogram module provides beside the 3D position of the individual points also a confidence factor, as described in Section 3.3.

In Figure 5.6, a measurement of the empty testing environment is illustrated. In the middle of the room, a reference target is hanging from the ceiling. All other obstacles are removed. The color encoding of the points represents the confidence factor, as calculated from the histogramming module. The color palette starts with green and transits into red, while green states a constant confidence. During multiple measurements, only few points are classified as uncertain. These points always occur at edges of objects, independent of their distance to the LiDAR. This factor may be used for edge detection or filtering.

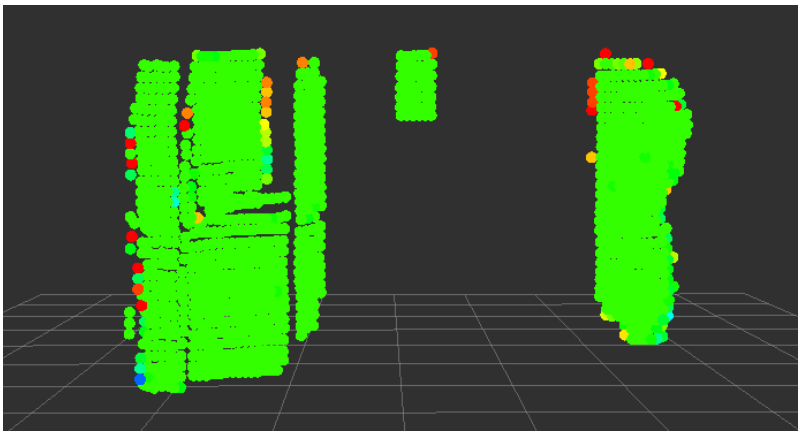
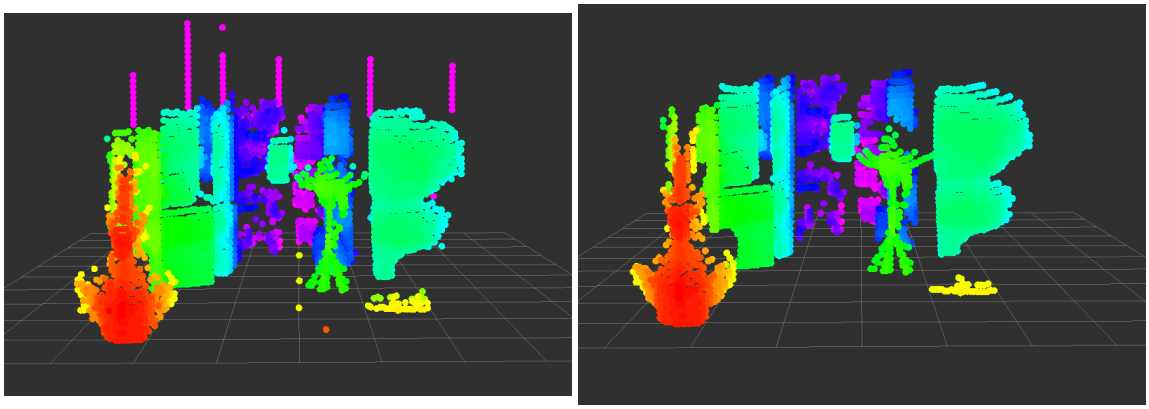


Figure 5.6: Point cloud representing the testing environment color encoded with confidence factor.

5.2.4 Histogram

For validating the histogramming module, both data streams provided over USB and UDP are captured and compared. Since the mechanism reduces noise by inspecting overlapping pulses of multiple measurements, higher noise in the data best represents this feature. After setting up the prototype in the test environment, the difference between both streams is visible but not significant. Therefore, a second LiDAR system with the same specifications is set up. This reference LiDAR faces against the LiDAR system under test. It is mounted onto a tripod and placed 2.94 meters away from the second LiDAR. Since it produces also light pulses in the same spectrum while scanning the scene, those scan lines are also captured by the implemented system. These scan lines occur when the LiDAR scanning patterns happen to be nearly or fully synchronized.



(a) (b)

Figure 5.7: On the left side, the raw output of a scene with a second LiDAR is represented. On the right, a histogram of the same measurement is displayed.

In Figure 5.7a, the encoded data received over USB is displayed. The reference LiDAR

beside the object hanging from the ceiling can be seen in green on the tripod. Also, purple stripes appearing behind the captured scene can be observed. These stripes result from the scanning pattern of the reference LiDAR and can be seen as noise. In Figure 5.7b, the same measurement concurrently captured over UDP is displayed. In this 3D representation, the data is histogrammed by the hardware module implemented. It can be seen that every noise measurement is filtered out. Both data streams are having the same Frames per Second (FPS), while the histrogrammed UDP version has a delay of 15 FPS.

During measurement, different amounts of measurements per histogram are tested. As reference, the noise in the encoded USB data stream is taken. After increasing the number of measurements above 6, no additional effect is observed in the histogrammed data. So it seems that 6 measurements are sufficient for noisy environments.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The Light Detection and Ranging technology has especially attracted interest in the field of Advanced Driver Assistance Systems (ADAS) in recent years. This technology provides the perfect extension to the already used sensors, since it is capable of producing a 3D representation of the observed environment. Further, it can provide additional properties of the object in a scene. Full waveform LiDARs open up the possibility to analyze the returning signals for more features of objects. The systems get therefore more expensive since the sampling rate is increased and the digital converters at higher speeds increase component costs. Since the full waveform analysis has emerged in recent years, the discoveries can be utilized and applied to 1bit LiDAR systems.

In this work, a hardware-accelerated 1bit LiDAR system is proposed, which is able to extract different features out of a sampled reflection bit-stream. Furthermore, it reduces the Signal to Noise Ratio with a signal averaging which additionally provides a new confidence attribute to each detected point. The FPGA-based system is independent of the hardware which is used to provide incoming LiDAR data. It is capable of providing resulting data-streams concurrently to give the opportunity of applying external signal processing techniques. With this system, a framework for capturing and displaying the processed point-cloud is proposed.

To fulfill the requirements, the Xilinx Artix 7 platform is used, which allows complex and resource intensive hardware-accelerated implementations. A MicroBlaze is used as micro processor to control the different components in the VHDL design. The core components can be split into an encoder and a histogrammer module. The encoder module samples the incoming reflections and encodes the information. During encoding, the pulse width as well as the elapsed time until the reflection are extracted. Out of the time-stamp, the distance of the object with respect to the LiDAR can be calculated. The resulting points can then directly be transmitted over an USB interface for further processing. This encoding provides the first feature, pulse width, as well as a significant data reduction, since it omits all remaining data, which does not contain any pulse information.

The next component in the signal processing is the histogramming module. This

takes the encoded data and detects intersections of measured pulses. Therefore, it merges multiple measurements and filters pulses with high deviation. Additionally, it provides every pulse with a confidence value, taken from the degree of intersection. The resulting data is then processed by the implemented MicroBlaze and transmitted over an Ethernet connection.The whole signal processing path is executed in real-time, since encoding and histogramming takes less than 20 μs.

Three different software programs are developed in order to display generated point-clouds and evaluate the implemented system. With different experiments, the basic functionality of the LiDAR as well as the different extracted features are analyzed. It is shown that even one bit resolution LiDAR systems allow basic filtering with pulse width information. Additionally, the histogramming method achieves extensive noise reduction while preserving high performance.

6.1.1 Future Work

The presented work proves the feasibility of a real-time signal processing LiDAR system extracting features out of a 1bit data stream. The design VHDL modules as well as the produced software can be further improved and extended to improve accuracy and performance. Moreover, the evaluation process can be further improved. Future work might involve following topics:

Parallelization of Hardware Components
With additional parallelization of different hardware components, the performance of the system could be increased even further. Especially in the histogramming algorithm, the measurements of the different APDs do not have any dependencies to one another. Therefore, by generating multiple instances of the sorter and intersector, the computations performance can be maximized.

Additional FPGA-Resources
With additional FPGA resources, the limiting parameters of encoding and histogramming can be increased. In the current implementation, the number of pulses detected per measurement is limited to three. Also, the number of measurements per histogram is limited to 15. With more BRAM resources, it is possible to overcome these limitations.

Hardware-only UDP Transmission
In the current implementation, calculated points are stored in a FIFO buffer and later read by the MicroBlaze. The micro processor then creates a UDP packet and transmits it over an Ethernet interface. This process can be implemented with a VHDL design, making the MicroBlaze obsolete, finally resulting in more free FPGA resources.

System Response Calibration
In order to be able to correctly classify reflection pulses, the system response of the LiDAR has to be known. This could be achieved with multiple different measurements and analyzing the analog waveforms of the receiver.

Reflectance Calibration

As stated in [HGF13], the LiDAR system could be calibrated regarding the reflectance of objects. With a calibrated template for different distances, also features like pulse width and confidence could be interpreted more accurately.

Reference LiDAR System

For evaluation of the implemented system, reference data could be gathered with a second full waveform LiDAR. Also, the different features extracted could be compared and validated more efficiently.

Appendix A

Terminology

ADC analog-to-digital converter

APD Avalanche Photo Diode

AXI Advanced eXtensible Interface

CFAR Constant False Alarm Rate

CWT continuous wavelet transformation

DDR Dual Data Rate

FIFO First In First Out

FPGA Field Programmable Gate Array

FPS frames per second

FTDI Future Technology Devices International Ltd.

FWHM Full Width Half Maximum

HFoV Horizontal Field of View

IDE Integrated Development Environment

IP Intellectual Property

LiDAR Light Detection and Ranging

LVDS Low Voltage Differential Signal

lwIP light weight IP

MEMS Micro-electro-mechanical System

MII Media Independent Interface

PHY Physical Layer

ROC Receiver Operation Characteristic

SNR Signal-Noise Ratio

SPI Serial Peripheral Interface

TDC time-to-digital converter

UART Universal Asynchronous Receiver Transmitter

UDP User Datagram Protocol

USB Universal Serial Bus

VFoV Vertical Field of View

VHDL Very High Speed Integrated Circuit Hardware Description Language

WC wavelet coefficient

Bibliography

[AG.17] Audi AG. Audi pre sense 360. https://www.audi-mediacenter.com/de/technik-lexikon-7180/fahrerassistenzsysteme-7184, 2017. [Online; accessed 1.2.2018].

[Bri15] Gregory Brill. Cygwin port of ROS. https://github.com/codenotes/ros_cygwin, 2015. [Online; accessed 6.8.2017].

[CC13] John A. Christian and Scott Cryan. A Survey of LIDAR Technology and its Use in Spacecraft Relative Navigation. In *AIAA Guidance, Navigation, and Control (GNC) Conference*, Reston, Virginia, aug 2013. American Institute of Aeronautics and Astronautics.

[EM16] University of Massachusetts Lowell Robotics Lab Eric McCann. ROS .NET Library. https://github.com/uml-robotics/ROS.NET, 2016. [Online; accessed 6.11.2017].

[FTD15] Future Technology Devices International Ltd FTDI. FT601 Datasheet Version 1.05. http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT600Q-FT601QICDatasheet.pdf, 2015. [Online; accessed 30.09.2017].

[FTD16] Future Technology Devices International Ltd FTDI. D3XX Drect USB Drivers. http://www.ftdichip.com/Drivers/D3XX.htm, 2016. [Online; accessed 12.10.2017].

[HGF13] Preston J Hartzell, Craig L Glennie, and David C Finnegan. Calibration of a Terrestrial Full Waveform Laser Scanner. *ASPRS 2013 Annual Conference Proceedings*, page p. 7, 2013.

[Inc16a] Xilinx Inc. Block Memory Generator. https://www.xilinx.com/products/intellectual-property/block_memory_generator.html, 2016. [Online; accessed 15.6.2017].

[Inc16b] Xilinx Inc. SelectIO Interface Wizard. https://www.xilinx.com/products/intellectual-property/selectio_interface_wizard.html, 2016. [Online; accessed 15.6.2017].

[Inc16c] Xilinx Inc. Vivado Design Suite. https://www.xilinx.com/products/design-tools/vivado.html, 2016. [Online; accessed 15.6.2017].

[Inc17] Xilinx Inc. FIFO Generator v13.1 LogiCORE IP Product Guide (PG057). https://www.xilinx.com/support/documentation/ip_documentation/fifo_generator/v13_1/pg057-fifo-generator.pdf, April 5, 2017. [Online; accessed 22.6.2017].

[Lab] MIT Lincoln Laboratory. Geiger-Mode Avalanche Photodiodes. https://www.ll.mit.edu/mission/electronics/ait/imaging-technology/geiger-mode-photodiodes.html. [Online; accessed 4.12.2017].

[Mic17] Microsoft. Visual Studio. https://www.visualstudio.com/de/, 2017. [Online; accessed 12.10.2017].

[OW16] Takashi Ogawa and Gerd Wanielik. TOF-LIDAR signal processing using the CFAR detector. *Adv. Radio Sci*, 14:161–167, 2016.

[PWKJ07] Thomas J. Papetti, William E. Walker, Charles E. Keffer, and Billy E. Johnson. Coherent backscatter: measurement of the retroreflective BRDF peak exhibited by several surfaces relevant to ladar applications. page 66820E, sep 2007.

[PWWU14] Martin Pfennigbauer, Clifford Wolf, Josef Weindopf, and Andreas Ullrich. Online waveform processing for demanding target situatons. *Proc. SPIE 9080: Laser Radar Technology and Applications XIX and Atmospheric Propagation XI*, 2014.

[ROS15] ROS. Point Cloud Message Format. http://docs.ros.org/api/sensor_msgs/html/msg/PointCloud2.html, 2015. [Online; accessed 16.8.2017].

[SANW16] Jason Stoker, Qassim Abdullah, Amar Nayegandhi, and Jayna Winehouse. Evaluation of Single Photon and Geiger Mode Lidar for the 3D Elevation Program. *Remote Sensing*, 8(11):767, sep 2016.

[SKKK] Hocheol Shin, Dohyun Kim, Yujin Kwon, and Yongdae Kim. Illusion and Dazzle: Adversarial Optical Channel Exploits against Lidars for Automotive Applications.

[UPa] A Ullrich and M Pfennigbauer. Linear LIDAR versus Geiger-mode LIDAR: Impact on data properties and data quality.

[UPb] Andreas Ullrich and Martin Pfennigbauer. Echo Digitization and Waveform Analysis in Airborne and Terrestrial Laser Scanning 53 rd Photogrammetric Week, Stuttgart, September 2011.

[UP11] Andreas Ullrich and Martin Pfennigbauer. Echo digitization and waveform analysis in airborne and terrestrial laser scanning. *Photogrammetric Week*, pages 217–228, 2011.

[Vas16] Joshua Vasquez. Sort faster with FPGAs. https://hackaday.com/2016/01/20/a-linear-time-sorting-algorithm-for-fpgas/, 2016. [Online; accessed 22.6.2017].

[Wan12] C K Wang. Exploring Weak and Overlapped Returns of a Lidar Waveform With a Wavelet-Based Echo Detector. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XXXIX-B7(September):529–534, 2012.