Agradecimentos

Agradecimentos.....
Acknowledgments.....

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

Resumo

Abstract

Contents

List of Figures **ix**

List of Tables **x**

Glossary **xi**

1 Introduction **1**
1.1 Context . . . 1
1.2 Motivation . . . 1
1.3 Main Goal . . . 1

2 Background and State of the Art **2**
2.1 Introduction to LiDAR Technology . . . 2
2.1.1 Basic Concepts . . . 3
2.1.2 Imaging Techniques . . . 5
2.1.3 Interfaces . . . 8
2.2 LiDAR in Automotive . . . 9
2.2.1 Applications . . . 9
2.2.2 Challenges . . . 11
2.3 Related Work . . . 12
2.3.1 Data decoding acceleration . . . 13
2.3.2 Communication protocol acceleration . . . 13
2.3.3 Software driver manipulation . . . 14
2.3.4 Discussion . . . 14

3 Platform, Tools and Sensors **15**
3.1 Advanced LiDAR Framework for Automotive . . . 15
3.2 Zynq UltraScale+ MPSoC ZCU104 . . . 16
3.2.1 Zynq UltraScale+ EV . . . 16
3.2.2 AMBA Advanced eXtensible Interface . . . 16

3.2.3 AXI 1G/2.5G Ethernet Subsystem . . . 18
3.2.4 AXI Direct Access Memory . . . 19
3.2.5 AXI4-Stream Broadcaster . . . 20
3.3 EVAL-CN0506-FMCZ . . . 20
3.4 Robot Operating System . . . 21
3.5 Yocto Project . . . 22
3.6 Hokuyo . . . 22
3.6.1 Sensor's functioning . . . 22
3.6.2 SCIP2.0 . . . 23
3.7 Velodyne Lidar . . . 28
3.7.1 Sensor's functioning . . . 28
3.7.2 Data packet . . . 29
3.7.2.1 Data Packet Structure . . . 31

4 System Specification 33
4.1 Ethernet Interface System . . . 34
4.1.1 MAC and PHY Connection . . . 34
4.1.2 Configuration and Data Movement . . . 35
4.2 Hokuyo . . . 37
4.2.1 Message Controller . . . 38
4.2.2 AXI Lite Interface . . . 39
4.2.3 Queue & Multiplexer . . . 40
4.2.4 Header Analyzer . . . 41
4.2.5 Block Organizer . . . 43
4.2.6 Decoder & Organizer . . . 43
4.3 Velodyne . . . 44
4.4 "Encapsulator" . . . 44
4.5 Software . . . 44
4.5.1 Software Stack . . . 44
4.5.1.1 Drivers . . . 44

5 Implementation 45
5.1 Hardware . . . 45
5.2 Software . . . 45

6 Evaluation and Results 46
6.1 Test Setup . . . 46
6.2 Results . . . 46

7 Conclusion **47**

7.1 Future Work 47

References **48**

List of Figures

2.1 Operation principle 3
2.2 Field of view representation 4
2.3 Beam Divergence illustration 5
2.4 Representation of Velodyne VLP-16 side view [1] 6
2.5 Optical phase array principle. Taken from [2] 7

3.1 Zynq UltraScale+ EV Block Diagram 17
3.2 Memory-mapped interfaces channel overview 18
3.3 AXI Ethernet Subsystem Block Diagram. Adapted from [3] 19
3.4 EVAL-CN0506-FMCZ Block Diagram 21
3.5 Hokuyo's FoV zones 23
3.6 Request Message structure 24
3.7 Response Message structure 24
3.8 2-character encoding example 25
3.9 Check code calculation example 26
3.10 Distance-Intensity Pair structure 26
3.11 Multiecho structure 27
3.12 Block splitting structure 27
3.13 Dual Return functioning illustration 28
3.14 Data Block examples of sensors with more or less than 32 laser channels . 31
3.15 Dual Return packet structure 32

4.1 Proposed System Block Diagram 33
4.2 OSI Model 34
4.3 EVAL-CN0506-FMCZ and MAC Connection Diagram 35
4.4 Ethernet Interface System Block Diagram 37
4.5 Hokuyo Controller System Overview 38
4.6 Control segment structure 38
4.7 Message Controller Interface Overview 39
4.8 Control segment structure 40

4.9 Mode field structure 40
4.10 Queue and MUX interface 41
4.11 Control segment structure 42

List of Tables

2.1 Brief summary of the Imaging techniques. Adapted from [4]. 6
2.2 Interface comparison between different types of sensors 9

3.1 Azimuth Offset for the VLS-128 30
3.2 Factory Bytes meaning 31

4.1 Constraints Used for the FMC Connector 36

Glossary

ADAS Advanced Driver-Assistance System

ALFA Advanced LiDAR Framework for Automotive

AMBA Advanced Microcontroller Bus Architecture

ASCII American Standard Code for Information Interchange

AXI Advanced eXtensible Interface

CMOS Complementary Metal–Oxide–Semiconductor

CORDIC COordinate Rotation DIgital Computer

DMA Direct Memory Access

FMC FPGA Mezzanine Card

FMC-LPC FPGA Mezzanine Card (FMC) Low Pin Count

FoV Field of View

FPGA Field-Programmable Gate Array

GEM Gigabit Ethernet Controller

GPS Global Positioning System

IP Internet Protocol

IP Intellectual Property

LiDAR Light Detection And Ranging

MAC Medium Access Control

MDIO Management Data Input/Output

MII Media Independent Interface

MIO Multiplexed Input/Output

NMEA National Marine Electronics Association

OPA Optical Phased Arrays

OSI Open Systems Interconnection

PHY Physical Layer Device

PL Programmable Logic

PPS Points per second

PS Processing System

RGMII Reduced Gigabit Media Independent Interface

RMII Reduced Media Independent Interface

ROS Robot Operating System

SCIP Sensor Communication Interface Protocol

SPI Serial Peripheral Interface

TCP Transmission Control Protocol

ToF Time of Flight

UDP User Datagram Protocol

1. Introduction

1.1 Context

1.2 Motivation

1.3 Main Goal

2. Background and State of the Art

This chapter addresses the main concepts of Light Detection And Ranging (LiDAR) technology in the section 2.1, where the basic concepts of how these sensors work and the advantages of the different types are covered. Then, in section 2.2, the main trends of LiDAR technology in the automotive field will be aborded alongside some of the challenges of this emerging technology. Finally, in section 2.3, a review and discussion on implementations similar to the one presented in this dissertation can be analyzed.

2.1 Introduction to LiDAR Technology

The use of techniques that measure the travel-time and intensity of light beams date back to before the laser's invention [5]. In 1930, Synge proposed a method to measure the density of the upper atmosphere [6]. It consisted of projecting a sufficiently strong beam of light to allow the light to scatter and be detected and measured by a photo-electric device. In 1935, the use of pulse width modulation was suggested, marking a significant improvement in the field [7]. This technique allowed the infusion of identification to the scattered light, making it easier to detect the desired signal and filter out background noise. In 1953, Middleton and Spilhaus [8] introduced the LIght Detection and Ranging (LiDAR) acronym for this technique.

With the invention of the laser (1960) [9], LiDAR technology started to develop quickly. Still, in 1960, Hughes Aircraft Company, which mainly specialized in defense electronics, created an internal competition to design a rangefinder using lasers for military use, demonstrating early the capabilities of this type of technology [10]. About a decade later, all of the basic LiDAR techniques had been suggested and tested [5]. Since 1976, with the publishing of the first textbook on LiDAR [11], improvement has been tied to optical and electronic progress, especially laser technology. This connection is attributed to the high requirements some LiDAR techniques have. These requirements range from laser technology, e.g., laser power, beam shape, to computer systems that can process large amounts of data [5], raising the need for custom-tailored hardware/software solutions.

2.1.1 Basic Concepts

A LiDAR system is composed of two components, the transmitter, and the receiver. The transmitter is the unit responsible for emitting the light pulses with wavelengths that range between 250 to 1600 nanometers (nm). The used wavelengths are application-dependent. For example, the solar spectrum has noticeable dips due to atmospheric absorption, and one of these dips occurs at 940nm, which can bring better SNR due to reduced solar background noise [12]. On the other hand, the receiver is responsible for receiving the reflected light, organizing the received data, and computing the information.

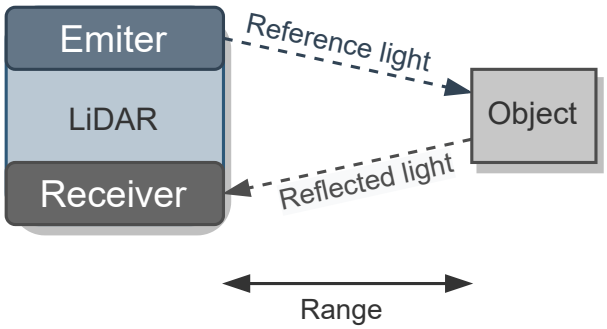


Figure 2.1: Operation principle

The working principle of LiDAR technology is based on the round-trip travel time of a light signal, also known as Time of Flight (ToF) [13]. As depicted in Figure 2.1, the transmitter emits a pulse that, when reflected on an object, is captured by the receiver. From here, distance to a target (R) can be calculated using equation 2.1. In equation 2.1, τ represents the time difference between the emission of the pulse and the reception of the corresponding reflection. Finally, c represents the speed of light in the medium.

$$R = \frac{1}{2}c\tau \tag{2.1}$$

The value of τ is achieved by modulating the emitted signal (intensity, phase, frequency) and measuring the time required for the same pattern to be received [2]. Despite, the existence of several techniques to calculate the ToF, LiDAR sensors are commonly characterized by standard metrics like range, Field of View (FoV), angular resolution, and frame rate are commonly used to characterize a LiDAR sensor [14].

Range

The most important values when it comes to the range are the maximum and minimum values. These values are specified in controlled environments using targets with 80% of

diffuse reflectivity, and they determine the maximum and minimum distance in which an object is visible to the sensor. In real-world measurements, these values can change due to light interferences and target reflectivity, which alongside transmitter power levels and receiver sensitivity, are the parameters that mainly define the sensor's effective ranges [15].

Field of View and Angular resolution

FoV represents the extent of the observable surroundings that can be sensed by the sensor at any given moment. In optical systems, FoV is specified with two horizontal and vertical angles for 3D LiDAR (Figure 2.2) and only horizontal for 2D LiDAR. Multiple applications require 360º degree coverage, which is already available by some LiDAR sensors or by combining multiple LiDAR with a smaller FoV [2, 15]. Another important aspect is the lateral or angular resolution, which measures the ability to distinguish two adjacent points in the FoV. Both of these metrics are affected by the imaging technique used by the sensor [12].

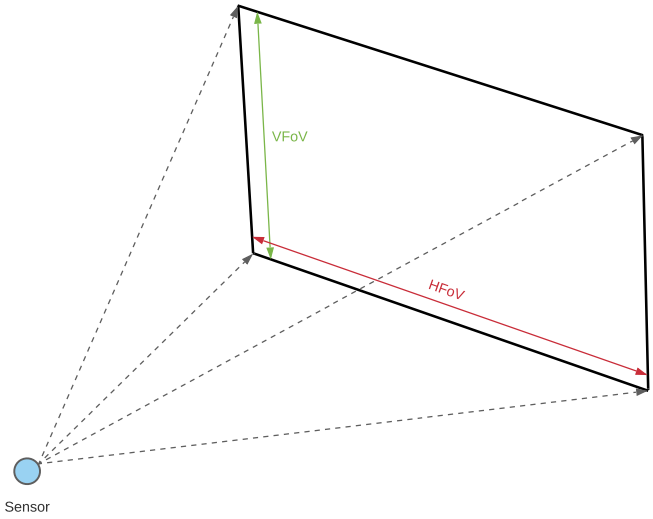


Figure 2.2: Field of view representation

Frame rate

The frame rate of a LiDAR sensor is the frequency at which it can measure a data point. This value is dependent on the type of LiDAR sensor and can reach a few hundred Hz. Frame-rate is an essential metric, not only because it allows faster object detection, needed in some real-time applications, but also because lower frame-rate values can introduce motion blur in moving targets [4]. The imaging technique used in a sensor is the biggest frame rate limiter, being the ones without moving parts the ones that can deliver higher speeds.

Beam Divergence

In all laser systems, the beam emitted by the laser increases in diameter as the distance to the target increases. The beam divergence phenomenon forms a thin cone instead of a cylinder shape beam [16]. In Figure 2.3 an illustration of this effect can be observed.

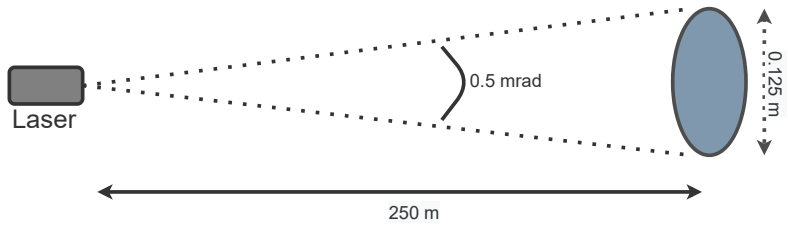


Figure 2.3: Beam Divergence illustration

Although this can be seen as an undesirable effect, as the same amount of energy is dispersed over a larger area, this effect is only noticeable at very high ranges. Beam divergence also allows the sensor to receive "multiple returns" if the pulse grows large enough, effectively doubling or tripling (depending on the sensor) the Points per second (PPS) the sensor is capable of capturing.

2.1.2 Imaging Techniques

LiDAR images are often constructed using multiple points to build an accurate representation of the surroundings. So, to capture all of these points in an FoV, different imaging techniques can be used. Currently, there are two main categories, scanners, and detector arrays, also known as Flash LiDAR [15]. Scanning-based LiDAR sensors use beam steering components to sweep various angular positions of the FoV, and can be divided into mechanical rotated and solid-state. The main difference between them is that in solid-state LiDAR, there are no rotating mechanical parts. Because of this, solid-state LiDAR sensors offer a more limited FoV than their mechanical counterpart but are cheaper to produce and offer higher frame rates. On the other hand, flash LiDARs illuminate the entire FoV and use an array of receiving elements, each one capturing a different angular section, to create a point cloud of the surroundings[17]. In Table 2.1, a summary of the various techniques' characteristics can be observed.

Scanners

LiDAR systems based on this technique usually have a rotating mirror or prism, which gives this type of sensor a 360º view with relative ease. The mechanical actuator in this type of sensor rotates around a single axis to obtain one dimension cover. The

Table 2.1: Brief summary of the Imaging techniques. Adapted from [4].

	Mechanical Scanners	**MEMS**	**OPAs**	**Flash**
Working principle	Rotating lasers, prisms or mirrors	MEMS micromirrors	Phased array of emitters	Flood illumination of the FoV
Main advantage	Long range, 360º of HFoV	Compact, lightweight, low power consumption	High frame-rates, On-chip solutions possible	High frame-rates, no moving parts
Main limitation	Moving elements, bulky, expensive	Power management, limited range	Long range not commercially available	Limited range

second dimension is usually obtained by adding sources and detectors with some angular difference between each other [18]. As Figure 2.4 exemplifies, Velodyne VLP-16 creates a 30 degrees vertical FoV by powering 16 individual lasers/receivers at different vertical angles. Those pairs, also known as channels are rotated horizontally, creating the typical 360 HFoV.

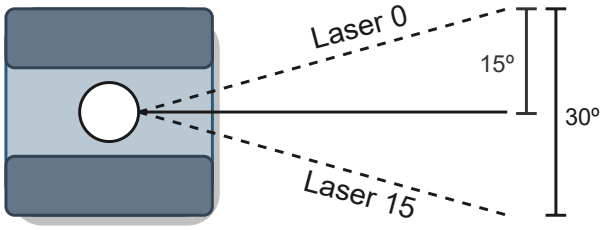


Figure 2.4: Representation of Velodyne VLP-16 side view [1]

An advantage of this type of sensor is that the optical arrangement is reasonably simple and efficient, which allows the sensor to collect faint diffuse light, thus achieving longer ranges. These sensors work with a high-frequency pulsing laser, and the laser's angular width and scan rate determine the angular resolution of the sensor [4]. Although they are very efficient in long-range applications, the setup presents numerous disadvantages such as limited scanning speeds, poor resistance to shock and vibration, large power consumption, bulky design, and price. Despite this, it is the most mature imaging technique, making this type of sensor the primary choice for autonomous research and development [14].

Microelectromechanical Scanners

Microelectromechanical scanners (MEMS) are an emerging solution. MEMS sensors have no rotating mechanical parts, instead, tiny mirrors with a few mm in diameter

are tilted at various angles when a stimulus is applied [4]. This stimulus depends on the actuation technologies used, and the required performance usually determines which technology is used [19]. As mentioned earlier, solid-state scanners offer smaller FoV than their mechanical counterparts, however, there are already several designs that fuse the data of multiple MEMS scanners to achieve higher FoV [20, 21].

This type of sensor presents poor resistance to shock, and the power limitations that arise to avoid mirror damage limit the maximum range. Despite this, due to the lightweight, compactness, and low power consumption, MEMS-based scanners are receiving increasing interest from the automotive world [19], being considered a suitable replacement for mechanical rotating scanners.

Optical Phased Arrays Scanners

Another emerging solid-state technology is Optical Phased Arrays (OPA) based scanners. These scanners employ a very different technique from the two mentioned earlier. While the former scanners are based on beam steering techniques, OPA-based scanners use beam shaping to achieve the same effects. These consist of multiple optical emitters, and by controlling the phase and amplitude of each one, the electromagnetic field close to the emitters can be fully controlled [22], enabling the control of the shape and orientation of the wave-front (Figure 2.5). As there are no moving parts, they are very resilient to shock and vibration and capable of notably high scanning speeds. Another advantage of this technique is that it can be deployed on on-chip solutions. An on-chip LiDAR fabricated using combined Complementary Metal–Oxide–Semiconductor (CMOS) and photonics manufacturing, if industrialized, has tremendous potential regarding reliability and reduced cost [4], which in the last years sparked the interest of various manufacturers. However, one major disadvantage is the insertion loss of the laser power in the OPA, which makes currently available solutions only fitted for mid to close range applications [23].

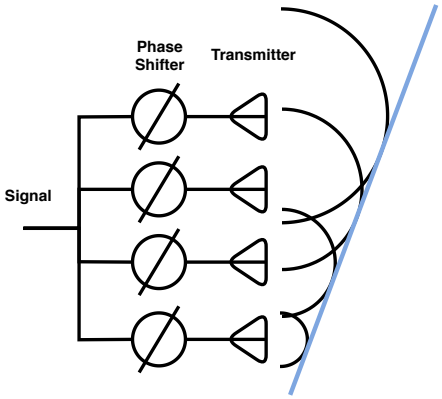


Figure 2.5: Optical phase array principle. Taken from [2]

Flash LiDAR

Flash LiDAR, are considered Full solid-state systems as they have no moving parts, and their operation is comparable to standard cameras. Unlike its counterparts, Flash LiDAR use a laser or array of lasers to illuminate the entire FoV and a detector array to capture the reflected signals [4]. As a consequence, the angular resolution of the system is highly influenced by the array's density, which makes this solution expensive for a large FoV [24]. The use of a flashing technique makes the emitter need higher power lasers to illuminate the entire FoV which also increases the power requirements [12]. Consequently, eye-safety regulations limit the power levels decreasing the maximum detection range (50-100 meters). Despite this, as the emitter illuminates the entire FoV simultaneously, these sensors can achieve very high capture rates, making them more resilient to moving artifacts. Also, as there are no moving parts, there exists the potential to create miniaturized systems, which can be a significant advantage against some of its counterparts [17, 25]. So, as these sensors are very straightforward in terms of mechanical complexity, they present a good solution for low to medium-range applications.

2.1.3 Interfaces

High-quality point-cloud representations of the surrounding environments are only possible using high-resolution sensors with a wide FoV, large maximum range and high frame-rate. The problem is that satisfying these metrics results in sensors with a high data-rate output, meaning that buffering and receiving all of this data becomes a great challenge to any system that uses a LiDAR sensor. Furthermore, to connect the LiDAR to the processing system, a high-bandwidth interface is also needed, raising the constraints of these types of systems even more.

CAN

CAN is a field bus level communication network for exchanging short real-time messages [26]. Currently is used in most automotive applications and is supported by several manufacturers of integrated circuits and subsystems [27]. It features built-in error detection, robustness, and extensive flexibility, and its best use case is for the exchange of small control messages [28]. Its low bandwidth makes it only suitable for sensors with very limited FoV or with a low angular resolution, limiting its use in the automotive field [29]. Its extensive use in automotive and industrial settings still makes it one of the main interfaces present in LiDAR sensors.

Ethernet

Ethernet is a family of wired networking technologies commonly used in local area networks (LAN), metropolitan area networks (MAN), and wide area networks (WAN) [30]. Since its creation, the Ethernet interface has been improved to support higher bit rates (+100Gbps), more nodes, and longer link distances [31]. The large bandwidth and high versatility make the Ethernet interface the main interface of current LiDAR solutions in the market. Although the Ethernet interface is becoming the standard in high data-rate outputs for LiDAR sensors, the protocol that supports communication is still very much manufacturer-based (table 2.2). Despite this, most manufacturers are converging on a solution that uses HTTP for sensor control and UDP-based data transfer.

Table 2.2: Interface comparison between different types of sensors

Sensor	Type	Interface	Control Protocol	Data transfer protocol
Hokuyo (UST and UTM)	Mechanical rotor	100 Mbps Ethernet	Proprietary (Based on TCP/IP)	Proprietary (Based on TCP/IP)
Velodyne VLP-16	Mechanical rotor	100 Mbps Ethernet	Proprietary (Based on HTTP)	Proprietary (UDP Based)
Velodyne VLS-128	Mechanical rotor	Gigabit Ethernet	Proprietary (Based on HTTP)	Proprietary (UDP Based)
RS-LIDAR-M1	MEMS Solid-state	Gigabit Ethernet	Proprietary (Based on HTTP)	Proprietary (UDP Based)
LeddarTech Pixel	3D Flash	100 Mbps Ethernet	Proprietary (TCP/IP Based)	Proprietary (TCP/IP Based)

2.2 LiDAR in Automotive

The use of LiDAR technology in the automotive field started around 2004 with DARPA Autonomous Vehicle Grand Challenges. These challenges consisted in creating a fully autonomous vehicle to navigate rural and urban settings. The necessity of LiDAR quickly became obvious, and in the third edition of the challenge, five of the six finishers had Velodyne LiDAR sensors [32]. The winner that year was Carnegie Mellon University, with their vehicle containing two video cameras, five radars, and 13 LiDAR [33]. These challenges represented an enormous leap in autonomous driving, attracting powerful agents to this area. Today various solutions in the market are based on the use of LiDAR sensors. One example of this is Google's (and Waymo's) approach to these types of systems, which is mainly based on LiDAR [34]. So, autonomous vehicles are quickly becoming a reality, and the use of LiDAR sensors in this type of system is rising.

2.2.1 Applications

Currently, autonomous driving capabilities are only possible using multi-sensor data fusion since different variables need to be monitored simultaneously [35]. The three principal sensors used for these systems are cameras, RADAR, and LiDAR. The arrangement and selection of these sensors are some of the most important aspects of a perception

system. As a consequence, car manufacturers keep changing approaches to fit specific requirements and needs. Nonetheless, there are three main areas of study when developing algorithms that push us closer to a fully autonomous system. These areas include traffic sign detection, perception of the environment, and obstacle detection [36].

Traffic Sign Detection

Traffic sign detection is an essential aspect of an autonomous system as these signs transmit information about the traffic regulations on the area or even possible warnings. LiDAR sensors are helpful in these tasks as traffic signs have high reflectivity values, making their detection fairly easy. In [37, 38], this property is explored as the point cloud is filtered by intensity values and then clustered to perform the detection of traffic signs. However, the most promissory designs for automotive are the ones that use sensor fusion, like [39, 40], which use algorithms that fuse camera data with LiDAR data to achieve high accuracy both in detection and recognition.

Perception of the Environment

Being able to perceive the environment around the car is essential. Moreover, understanding the road geometry (curves, intersections) and road marks (stopping lines, road limits) is a must for an autonomous system. LiDAR's main applications concentrate on detecting the road boundaries and ground plane [41]. Road marks, like traffic signs, present high reflectivity values. Recent works take advantage of this characteristic to detect the road marks successfully [42, 43]. However, weak road maintenance and weather conditions that obstruct the road surface, e.g., covering the road with water, may cause the system to yield weak predictions. To solve this, in [44] a system that fuses LiDAR and camera data was proposed. This solution presented higher stability, a higher level of redundancy, and better results when faced with adverse conditions.

Obstacle Detection

An autonomous system must detect and recognize objects that may present a risk to it or its passengers. In addition to this, a fully autonomous system must detect the speed, motion direction, and position of objects around the car. As this is an information-demanding task, systems with a single sensor struggle to give feasible results [45]. In [46], an algorithm that fuses Radar and LiDAR data is proposed. This method of fusion increases detection speeds and precision when compared to a system that only uses LiDAR. However, a very recent work achieved high success rates and real-time constraints fusing

camera data with LiDAR information [47]. It uses LiDAR for quick obstacle detection/-tracking and a CNN-based detection algorithm specialized for image processing for object classification.

2.2.2 Challenges

Despite the widespread consensus that LiDAR sensors can be a major component of Advanced Driver-Assistance System (ADAS), several challenges are still being tackled by industry and academic research. In this section, some of these challenges will be addressed as they introduce some necessary concepts to this dissertation's concept.

Compression

Storage and transmission of data generated by LiDAR sensors are some of the most challenging aspects of their deployment [48]. Both of these problems can be resolved using compression methods. When it comes to data storage, compression algorithms are mainly used to reduce the point cloud size, as they can achieve several gigabytes of data. In this category, several lossless and lossy techniques can be found [49]. However, these techniques are mainly indicated for point-cloud storage and streaming as they require a complete frame to be received in the ECU, making their use advantageous for, for example, airborne geographical mapping and not for mobile real-time applications [50].

To address the problem of data transmission, two types of solutions can be introduced: (1) the design of faster interfaces to handle the data, like the system proposed in this dissertation; (2) the use of compression algorithms to reduce the strain on the communication interfaces. In [48] a method to reduce the size of RAW data for storage and transmission is suggested. The proposed technique uses the difference between the small resolution the transferred data has in relation to the actual resolution of the sensor, allowing the effective bit-masking of n least significant bits, making all the values lower than the sensor's resolution equal to 0. The advantages of this technique are the efficiency, as only an AND operation is needed for the masking, and that any compression algorithm of choice can be used without losing the accuracy rating claimed by the vendor.

Despite presenting good results in reducing transmission strain, performing compression tasks may heavily penalize real-time applications that rely on the sensor output [2]. Another problem that may arise is that for the best performance possible, the compression needs to occur as close to the LiDAR as possible, ideally inside of it. Current LiDAR solutions don't present this feature natively, which makes the validation of these systems harder.

Interfaces

As mentioned earlier, most of the current LiDAR COTS use Ethernet as the main output interface, not only because of the high bandwidth requirements these systems have but also because most industries that use them have the Ethernet interface as one of the main standards [51]. Ethernet has become a recognized network technology for in-car communications, for adoption in specific domains, like multimedia/infotainment and ADASs [52] as current automotive network typologies cannot provide enough bandwidth for these applications. Another point in favor of Ethernet is that a common network technology would reduce the communication complexity introduced by the various network technologies that support the different automotive domains, e.g., powertrain, chassis [53, 54]. In addition, the large market availability, because of the widespread use over several domains, make this transition easier.

Despite these advantages, some problems arise when considering the extreme conditions automotive networks have to endure. As these networks have to operate in extreme temperatures and high electromagnetic radiation, standard Ethernet networking solutions have to be adapted [55]. Another problem is the real-time constraints that automotive systems require. A protocol like CAN offers low maximum latency times, which makes all the difference on the road. While standard Ethernet networking solutions do not offer real-time support, several standards that address this issue have already been created and are under development [52].

Currently, multiple IEEE standards define the Automotive Ethernet, which supports various speeds as well as being able to perform under the harsh conditions of a car. These are the 10GBASE-T1, 5GBASE-T1, 2.5GBASE-T1, 1000BASE-T1, 100BASE-T1, 1000BASE-T, 100BASE-TX, and 10BASE-T standards. The 2.5G/5G/10GBase-T1 are defined in the IEEE 802.3ch standard, and the 100/1000Base-T1 are defined in IEEE 802.3bw and IEEE 802.3bp, respectively.

When it comes to the rest of the networking stack, the Internet Protocol (IP) is seen as the possible network layer for use in Automotive Ethernet networks [56]. The main advantages of the IP protocol are the enormous support already existent, which helps to reduce complexity and provides the flexibility of using established transport layer protocols [57]. Nonetheless, Automotive Ethernet has sparked a lot of interest in major automotive companies which will surely drive innovation forward.

2.3 Related Work

This dissertation aims to study and provide an FPGA-based approach for driving automotive LiDAR sensors that are Ethernet interface enabled. It must support both

2D sensors as well as 3D and be encapsulated inside a configurable IP Core. Finally, it will also feature software drivers that are ROS compatible. In the following section, implementations that accelerate the decoding and/or the receiving of LiDAR data will be discussed.

2.3.1 Data decoding acceleration

In [58, 59] a design for a system-on-chip that can decode LiDAR data was suggested. The suggested system can achieve real-time performance by creating a point cloud directly from data packets sent over an Ethernet interface. It consists of multiple cores that divide the necessary data, e.g., azimuth, distance, and feed it into a COordinate Rotation DIgital Computer (CORDIC) system that calculates the points for the point cloud representation. This implementation differs from the one presented in this dissertation because it is not customizable and can only support one sensor, the VLP-16 from Velodyne [1]. It also can't decode the data for all possible functioning modes of the sensor, and the sensor must be locked at 10Hz, half of the maximum available by the sensor.

2.3.2 Communication protocol acceleration

The Ethernet interface is composed of various layers. In the lower layers, the physical aspect of the network, e.g., power levels, timings, is controlled by specialized hardware, while on the top layers, data flow is mainly handled by a software stack. This software stack, when deployed on an operating system, can introduce unwanted delays in the transfer of data, reducing the bandwidth of the interface. Several implementations of these stacks that range from the use of specialized chips [60] to custom Intellectual Property (IP) Cores [61, 62] exist and could be a reliable solution to eliminate the need for the stack to be present in an operating system. In [41], the use of a specialized chip that implements a Transmission Control Protocol (TCP)/IP stack to accelerate the reading of LiDAR data was suggested. The used chip is the W5500, which supports various transport layer protocols and the 10BaseT/100BaseTX Ethernet standards. The communication with the Field-Programmable Gate Array (FPGA) is done via Serial Peripheral Interface (SPI), for both the configuration and transfer of data. The supported sensor is the SICK LMS111, however as this implementation does not implement any decoding of data, any sensor can be integrated. Nevertheless, as no actual decoding and processing of data are done to the LiDAR data, extra engineering effort would have to be done for the use of a sensor only in hardware.

2.3.3 Software driver manipulation

Another way of accelerating a process is using software acceleration. Although usually little benefit can be obtained by applying software acceleration methods to general-purpose applications, software acceleration can be very profitable in special-purpose applications [63]. In [64] an FPGA-based deep learning platform for real-time point cloud processing is suggested. Although the main focus of this implementation is on a hardware deep-learning platform, the software driver for the sensor was slightly modified to better fit into the application. The modifications consist of changing the coordinate system the drivers convert the data into and upload it directly into DDR memory. Time is saved as there is no need to wait for the driver's output data, as it converts the LiDAR data into an unwanted coordinate system, and as the data is available to the hardware as soon as it is pre-processed by the altered software driver.

2.3.4 Discussion

The system proposed in this dissertation implements a system that supports both 3D and 2D sensors in their multiple modes. The implementation proposed in [58, 59] presents and highlights the advantages of decoding and processing LiDAR data in hardware. Despite this, the system is limited in sensor support, only featuring one sensor in a specific mode of operation. In [41], specialized hardware to accelerate the network stack was used with great success to accelerate the communication with a LiDAR sensor. Although the implementation has potential for use with various sensors, it only shows the advantages of not relying entirely on software to handle the complete network stack of an Ethernet interface, as no actual decoding and pre-processing of LiDAR data is done in hardware.

Finally, in [64], the need for hardware native solutions for the decoding and pre-processing of LiDAR data was demonstrated. The Velodyne driver was altered to modify the coordinate system and to inject data directly into DDR to be used by hardware. Although the solution proposed in this dissertation features a polar coordinates system natively, modules to expand this functionality can be easily added. Another advantage is that data never leaves the hardware layer so, data transfer delays between software and hardware are non-existent. In the end, the presented implementations demonstrate and support the necessity for a hardware approach to decode LiDAR data.

3. Platform, Tools and Sensors

In this chapter, the used platforms, tools, and sensors necessary for the concretization of this dissertation will be addressed. First, the used boards will be described in section 3.1, followed by the platforms and tools ***. Finally, in sections 3.6 and 3.7 the supported sensors will be explained.

3.1 Advanced LiDAR Framework for Automotive

Advanced LiDAR Framework for Automotive (ALFA) is an open-source Framework for Automotive that aims to offer a multitude of features helpful for the validation and development of automotive LiDAR systems. These features are:

- Generic and multi-sensor interface;
- Pre-processing algorithms for data compression, noise filtering, ground segmentation, along with others;
- Configurable output for High-level applications;
- Reconfigurable point-cloud representation architecture;

ALFA can be divided into several Cores. These Cores are independently configurable, removable, and expandable, boosting ALFA's functionality and flexibility. The individual cores offer basic communication interfaces and resources. From here, an algorithm can be built and added to its specific Core, eliminating the hassle of developing an entire system to validate an algorithm or part of it. As can be seen in Figure **??**, ALFA is also divided into hardware and software capabilities. The ALFA Core accelerators are the main components of the system, providing a Control Unit, a Memory Unit, point cloud pre-processing, and a LiDAR interface, being the latter where this dissertation is inserted. Although the system proposed in this dissertation is easily integrable into other platforms, the main purpose is for the suggested implementation to be inserted and used in ALFA. As can be seen, the ALFA Core is inserted in the hardware layer, making the system suggested in this dissertation even more helpful, as was discussed in section 2.3.4.

3.2 Zynq UltraScale+ MPSoC ZCU104

The ZCU104 is an evaluation kit tailored for embedded vision applications such as ADAS, machine vision, and others. It is equipped with a ZU7EV device that features a quad-core ARM Cortex-A53 APU, a dual-core Cortex-R5 real-time processor, a Mali-400 MP2 GPU, and 16nm FinFET+ Programmable Logic (PL). Alongside this, the board features multiple IO Ports and Expansion Connectors that give a lot of flexibility to the board. To expedite the system design and deployment on the board, the Xilinx's Vivado Design Suite as well as the Vitis Unified Software Platform provide full compatibility with this board.

3.2.1 Zynq UltraScale+ EV

As it can be seen in Figure 3.1, the Zynq UltraScale+ EV family integrates two distinct systems: the Processing System (PS) and the PL. Their compositions vary greatly as their realms of action are different. The PS is composed of resources that are tailored for software: a quad-core ARM Cortex-A53, a dual-core Cortex-R5, on-chip memory, external memory interfaces, and directly connected I/O peripherals and PL interconnects. On the other hand, the PL contains resources that make the deployment of hardware solutions easier: configurable logic blocks, direct access to memory-oriented blocks (BRAM, URAM), access to high-speed I/O solutions, and an integrated video codec.

This combination of features extends the functionality of hardware-oriented solutions as well as software-oriented solutions, making the software/hardware co-design process easier. However, to make this co-design possible high-speed interfaces between the two systems are needed. For this communication, multiple interfaces based on the Advanced eXtensible Interface (AXI) protocol are provided.

3.2.2 AMBA Advanced eXtensible Interface

The Advanced Microcontroller Bus Architecture (AMBA) is an open-standard, on-chip interconnect specification for the connection and management of functional blocks in system-on-a-chip (SoC) designs. It defines various communication protocols that can be used to make functional blocks communicate with each other [65]. One of the AMBA standards is the AXI protocol. The AXI protocol features independent read and write channels, no strict timing relationship between address and data operations, support for unaligned data transfers, out-of-order transactions, and burst transactions [66]. As of the time of writing, the latest AXI version is the AXI5 specification. However, the IPs used

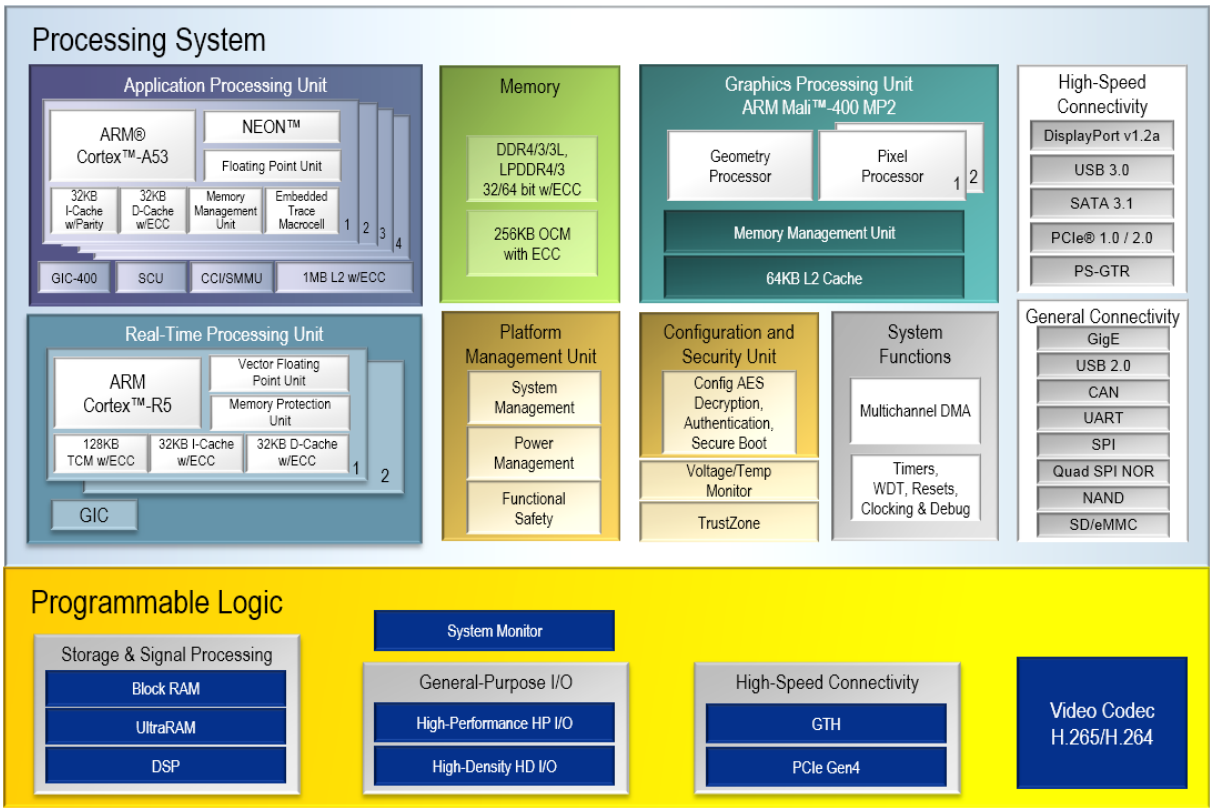


Figure 3.1: Zynq UltraScale+ EV Block Diagram

with the board are only compliant with the AXI4 specification, so that is the version aborded from now on.

There are three different types of interface in the AXI protocol: AXI-Full [67], AXI-Lite [68] and AXI-Stream [69]. The AXI-Full and AXI-Lite interfaces are considered Memory-mapped interfaces, where the write and read operations are issued on a specific address. On the other hand, the AXI-Stream interface is considered as a point-to-point interface, where the objective is to take data from a component to another with no specific address constraints. Despite their differences, all three interfaces work on the principle of master/slave communication, where the master is the one that controls which type of transfer (read/write) occurs and when they occur. Another characteristic in common is a handshake process for each channel based on two signals: the VALID and READY signals. A transaction only takes place when both the VALID (set by the master) and READY (set by the slave) signals are set.

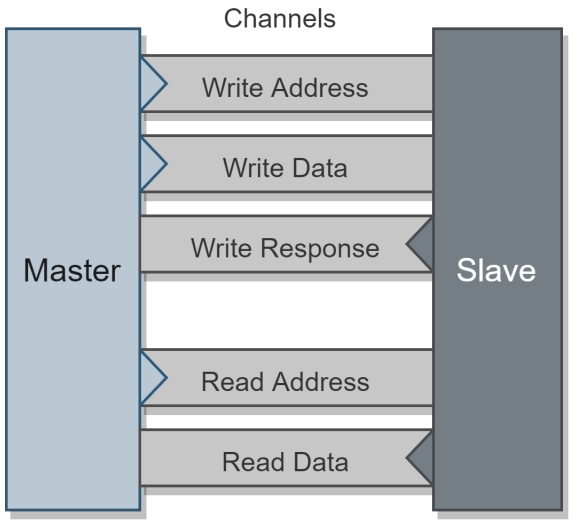


Figure 3.2: Memory-mapped interfaces channel overview

Both Memory-mapped interfaces have individual channels for address and data transfer, each with independent control signals. They also support read and write operations between master and slave, something that the AXI-Stream interface does not support. In Figure 3.2, the channel composition of both Memory-mapped interfaces can be observed. Despite their similar structure, the AXI-Lite interface is more limited than the AXI-Full interface, not allowing bursts bigger than one data length, and all the accesses use the full width of the data bus. The AXI-Stream interface only has one channel for writing data. Also, as mentioned, the AXI-Stream interface does not use addressing to specify where the data is going, making it ideal for streaming of large data buffers without the need to implement an address controlling system.

3.2.3 AXI 1G/2.5G Ethernet Subsystem

The AXI 1G/2.5G Ethernet Subsystem is an IP core developed by Xilinx that implements a tri-mode (10/100/1000 Mb/s) Ethernet MAC or a 10/100 Mb/s Ethernet MAC [3]. It supports Media Independent Interface (MII), Reduced Media Independent Interface (RMII) and Reduced Gigabit Media Independent Interface (RGMII) Medium Access Control (MAC) interface modes that allows it to connect to a Physical Layer Device (PHY) chip. It also features an Management Data Input/Output (MDIO) interface for access to the PHY management registers, useful to configure the PHY to the different functioning modes and speeds.

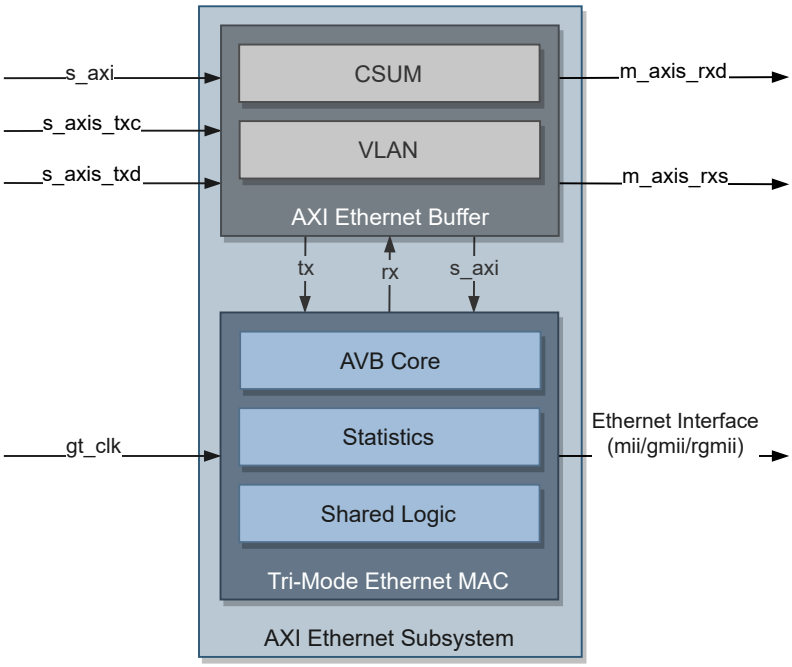


Figure 3.3: AXI Ethernet Subsystem Block Diagram. Adapted from [3]

As can be observed in Figure 3.3, the subsystem is composed of two main blocks: the MAC and a AXI Ethernet Buffer that handles data buffering and MAC control. An AXI Lite interface is provided for simple access to the IP Core configuration registers. On the other hand, 32-bit AXI-Stream buses are provided for moving both the transmited and received Ethernet data. When data is received by the Ethernet interface the data is provided on the *m_axis_rxd* port, meaning that any custom IP Core can access the data by implementing the AXI-Stream protocol. The same is true for sending data to the Ethernet interface, however in this case the port that must be used is the *s_axis_txd*.

3.2.4 AXI Direct Access Memory

The AXI Direct Memory Access (DMA) is an IP core developed by Xilinx that provides high-bandwidth memory access between memory and custom peripherals using the AXI Stream protocol. The AXI DMA provides two operation modes that differ mainly on their features and possible configurations: the Register Mode and the Scatter/Gather mode [70].

The typical use case of the AXI DMA is to read and write data from the DDR memory to and from the PL. The DDR Controller provides several AXI Full slave interfaces used by the AXI DMA master interfaces to access the DDR (*S2MM* and *MM2S*). On the other hand, to move data to and from the PL, one AXI Stream slave is used to receive data from the PL, and one AXI Stream master is used to send data to a PL peripheral. Finally, an AXI Lite interface is used to configure the IP Core's parameters.

3.2.5 AXI4-Stream Broadcaster

The AXI4-Stream Broadcaster provides support for replicating a single input AXI4-Stream interface into multiple output AXI4-Stream interfaces. This AXI4-Stream Broadcaster is an IP core developed by Xilinx and offers up to 16 outbound AXI4-Stream interfaces. As mentioned earlier, in the AXI Stream protocol, data flows from a single master to one slave. However, some applications may require the broadcast of data to multiple slave peripherals. So, instead of creating a custom solution to handle this problem, the AXI4-Stream Broadcaster IP Core can be used.

Alongside this feature, it also supports the use of a *TUSER* signal, which allows the mapping of data to a specific slave interface instead of broadcasting all of the data to all the slave interfaces, allowing the creation of a single master instead of multiple masters to multiples slaves.

3.3 EVAL-CN0506-FMCZ

The EVAL-CN0506-FMCZ is a dual-channel, low latency, low power Ethernet PHY card that supports 10 Mbps, 100 Mbps, and 1000 Mbps speeds for industrial Ethernet applications using line and ring network topologies. The dual Ethernet PHYs are the ADIN1300 that feature a MII, RMII, RGMII MAC interface modes. The board also features two RJ45 ports, input and output clock buffering, management interface, and subsystem registers. The design is powered through the FMC Low Pin Count (FMC-LPC) connector by the host board that, in this case, will be the ZCU104. A block diagram of this board can be further analyzed in Figure 3.4.

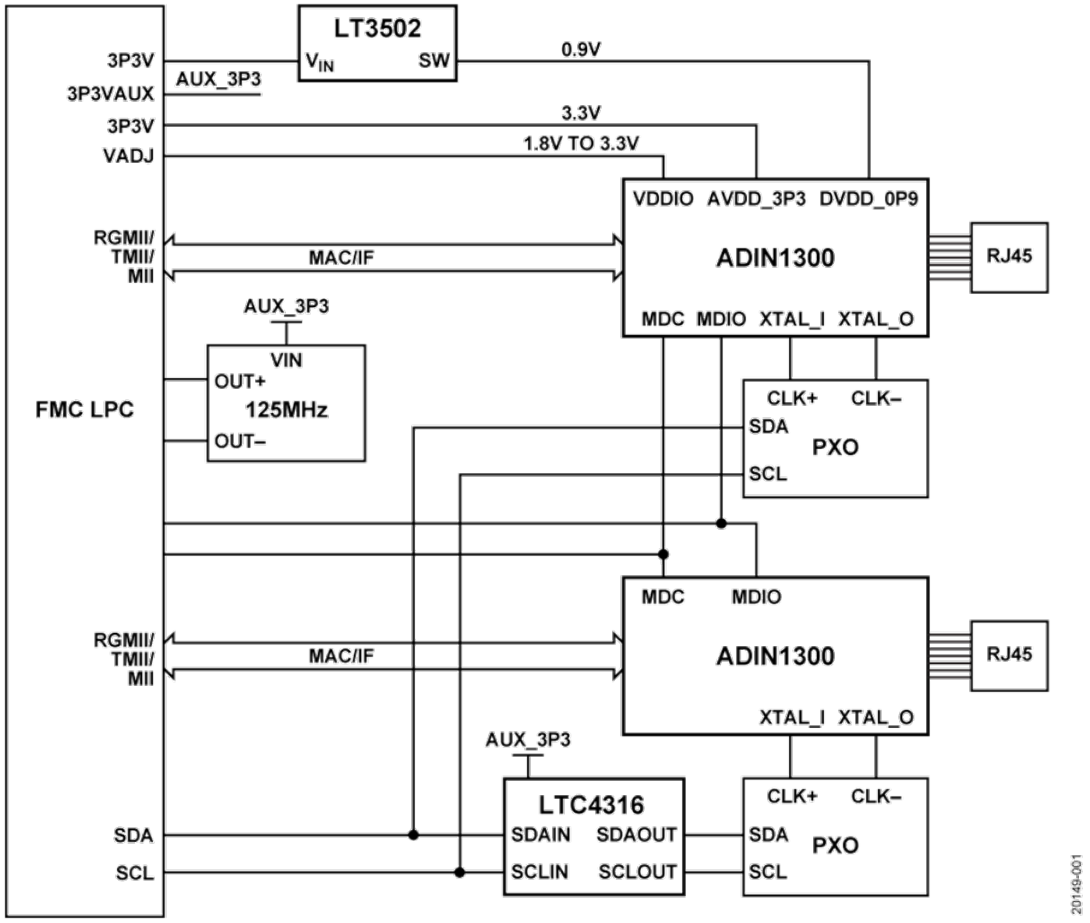


Figure 3.4: EVAL-CN0506-FMCZ Block Diagram

3.4 Robot Operating System

The Robot Operating System (ROS) is a flexible framework that features a collection of tools, libraries, and conventions that simplify the creation of complex and robust robot systems. As ROS is open source, it has a large and engaging community of developers and companies. Its main use is the design and implementation of heterogeneous computer clusters, with message and service capabilities between the users inside the same network.

ROS is based on nodes, which are processes executing a task. These nodes can be connected using the message and service infrastructure that ROS provides. The message infrastructure is based on a publish/subscribe topology controlled by a central node called Master. In ROS1, the Master is an essential part of the network, and all the nodes and topics("bus" in which the nodes can send and receive messages) must be registered in the Master to be considered part of the network. A node can also have services. Services are indicated to activate a specific task or change configurations within a node, on contrary to topics, which are better for continuous data exchange.

ROS was also chosen because both platforms of sensors in use feature a ROS driver. So, using ROS within the developed system is an advantage, not only because it will make the transition easier and be a good communication base for the system, but also because it is a well-known platform with many applications inside the automotive world.

3.5 Yocto Project

The Yocto Project is an open-source framework that provides templates, tools, and methods intending to simplify the software development process for Linux distributions. The project is supported and governed by high-tech industry leaders that have invested resources and offer platform support.

The Yocto Project is built around the OpenEmbedded framework which, offers a build automation framework and a cross-compile environment. The OpenEmbedded framework is based on the concept of layers and recipes. A recipe lists the steps and dependencies the system must perform and have to build a specific package. Alongside this, different layers offer different packages and recipes. For example, to include ROS capabilities into a Linux distribution, the ROS layer must be included. From there, the recipes for ROS packages are included in the project and can be incorporated into the final Linux distribution. The layer/recipe based functioning makes it easy to make custom-tailored Linux distributions with only the wanted packages and functionalities.

3.6 Hokuyo

Hokuyo is a Japanese company founded in 1946 that provides various products from photoelectric sensors to factory automation systems. Most relevant for this dissertation is the 2D LiDARs that Hokuyo produces. The use of a 2D LiDAR for this dissertation was deemed necessary as various uses for a sensor with this characteristic can be found useful in the automotive world. This, allied with the worldwide reference that Hokuyo represents, were the reasons for this choice.

3.6.1 Sensor's functioning

Hokuyo sensors feature a wide range of functioning modes allied with a relatively high horizontal FoV and a high refresh rate. All of the high-speed Hokuyo sensors feature an Ethernet interface with a communication protocol based on the TCP. The sensors are divided into "steps" in which each step represents a division (angular resolution) of the entire sensor's FoV. For example, if a sensor has an FoV of 250º and 1000 steps, each step represents $\frac{250}{1000} = 0.25º$. Also, some sensors have a dead zone where they will not sense

anything, despite having steps in those zones. A representation of a sensor FoV can be observed in Figure 3.5.

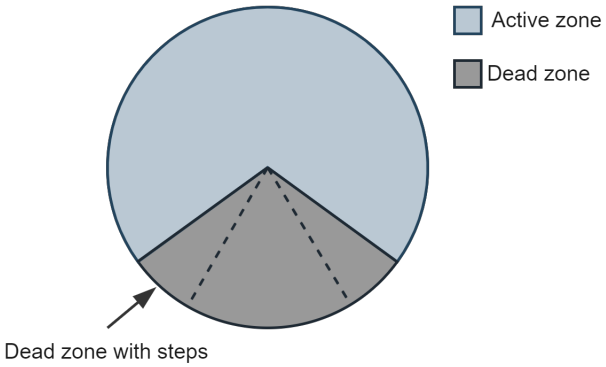


Figure 3.5: Hokuyo's FoV zones

3.6.2 SCIP2.0

The Hokuyo sensors use the *SCIP2.0* protocol, which is a communication protocol compliant with the Sensor Communication Interface Protocol (SCIP). This protocol does not depend on a particular type of sensor and it is based on "command-response" type of control, where a request is sent by the host unit (processing system), and a response to the command is sent by the sensor. Several measurement commands control the type of information the sensor provides, and non-measurement commands provide sensor information and basic control.

Message Types

As was mentioned earlier, this protocol is based on a command-response behaviour. For this, three types of messages have been defined by the protocol. These types are Request Message, Response Message, and Scan Response Message. The fields inside each of these message types can be different depending on the command or asked parameters and sensor configuration. Command codes are expressed by two alphabet characters, and they define the sensor's behavior depending on the requested parameters. The command parameters, may vary depending on the command, however they are all defined by integers in which the number of digits is fixed. For example, if the parameter has three fixed digits and the wanted value is four, the value to be sent is 004.

Request Message: To start a scan or query sensor's information, first, a message must be sent from the host system to the sensor, denominated Request Message. These types of messages are composed of a command code, command parameters, a user-defined

string as well as a request terminator (Figure 3.6). The user-defined string is an optional element inside a request message that is used for message identification. Finally, the request terminator can either be the carriage return (CR) character, the line feed (LF) character, or both, serving as a way to more easily parse the message.

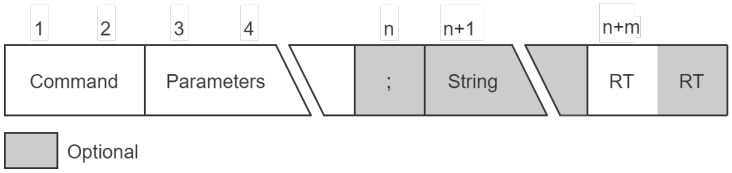


Figure 3.6: Request Message structure

Response Message: This type of message defines the message sent by the sensor if the command represents a single scan command or an information request command. A single scan command only queries information about a single scan by the sensor, which in the case of the Hokuyo sensors is a full rotation of the FoV. On the other hand, an information request command represents a command that requests status information about the sensor. So, a response message is composed of an echo of the sent Request Message, status information, and scanning data (Figure 3.7). After the echo of the Request Message, the message is composed of two bytes containing status information, followed by a checksum byte and a response terminator. Depending on the command type (mentioned earlier), there may or may not exist scanning information in the data field. Finally, the last byte is also a response terminator, signaling that the message has ended.

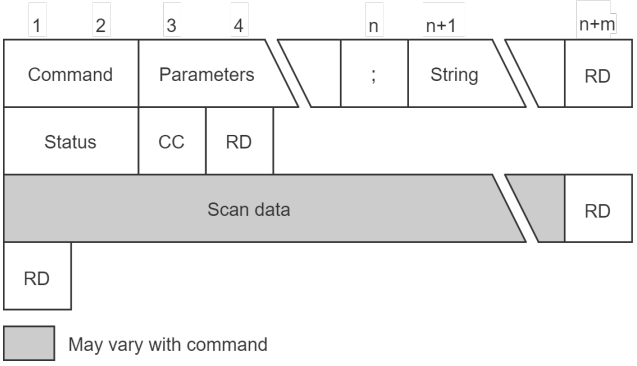


Figure 3.7: Response Message structure

Scan Response Message: When a continuous scanning command is sent by the host system, the type of message the sensor sends back is the Scan Response Message. A Scan Response Message has the same structure as a Response Message. However, the field corresponding to the retransmission of the Request Message is partially different. All of the continuous scanning commands have a parameter that defines the number of scans the host system wants. In a Scan Response Message, the sensor alters this parameter

to signal which scan the message represents. Also, when a continuous scanning mode is requested, the sensor sends a Response Message without any data before starting the transmission of the Scan Response Messages.

Encoding

In SCIP, the numerical representation of the scanning data passes through encoding to reduce the strain on the communication interface. The name of the performed encoding is character encoding, and it consists of dividing the numbers into 6-bit groups and transforming them into 6-bit encoded characters. After the groups' creation, the number 0x30 is added to each 6-bit group creating an American Standard Code for Information Interchange (ASCII) representation of the integer value. This value is then ordered in big-endian and sent over the interface to the host. If, after encoding, the value has two characters, it is called 2-character encoding and so on and so forward. In Figure 3.8 an example of the character encoding with the number 2607 can be observed.

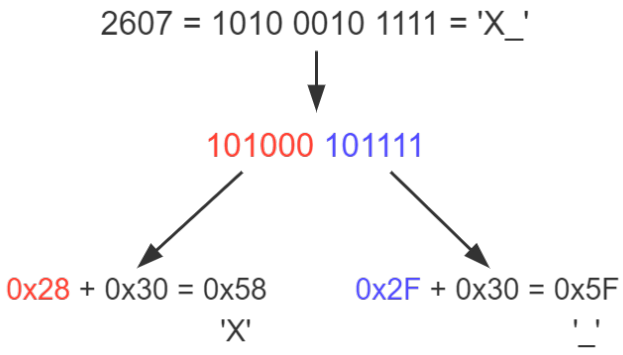


Figure 3.8: 2-character encoding example

Check Code

Check codes are used in a variety of settings to help in error detection that may have occurred during transmission or storage. SCIP implements a type of check code to verify if a block of data has suffered any corruption during the transmission process. The algorithm that calculates the check code starts by adding all of the data inside a determined block. After adding all of the data, the last six bits of the value are taken. Finally, the number 0x30 (character encoding) is added to the 6-bit value obtained in the previous step. An example of the check code calculation with the string 'ABC' can be observed in Figure 3.9. The final check code in this example is 0x36.

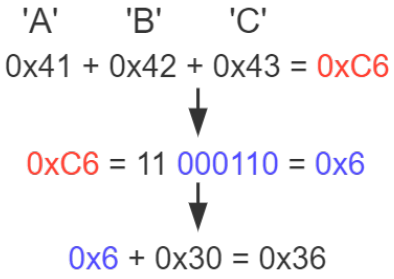


Figure 3.9: Check code calculation example

Measurement Data

The Hokuyo sensors provide four types of scanning data. These are distance, distance-intensity pair, multiecho distance, and multiecho distance-intensity pair. Which type the sensor provides at any specific moment is command-dependent.

- *Distance:* The sensor is capable of sending distance values of up to 262143 mm. However, for this to happen, the sensor has to be functioning in a mode that uses 3-character encoding. This is the case because to represent such a high distance value, 18-bits of data are necessary. If the sensor is functioning with a mode that uses 2-character encoding, the maximum distance value is 4095 mm using 12-bit data.

- *Distance-Intensity Pair:* Alongside distance, the sensor is capable of collecting and providing the intensity values for the reflected signals. Both distance and intensity come in 18-bit data so, 3-character encoding is used. When this mode is active, the pair comes for each step the sensor takes. The first three bytes correspond to the distance values, and the remaining three correspond to the intensity values (Figure 3.10). Note that the intensity value is directly proportional to the energy the receiver detects.

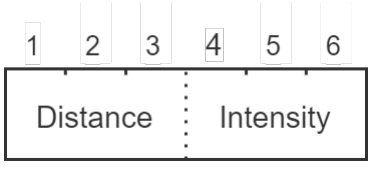
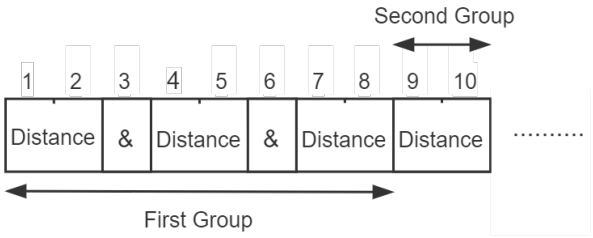
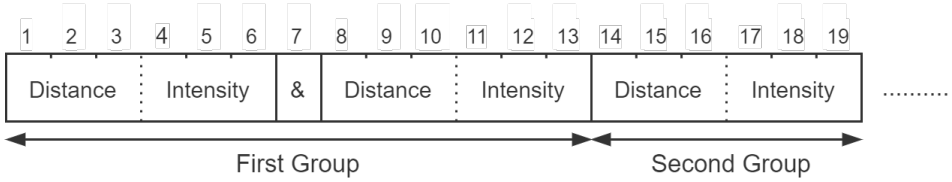


Figure 3.10: Distance-Intensity Pair structure

- *Multiecho:* Several LiDAR sensors can provide the results for several returns. In SCIP2.0, this is managed by using the multiecho structure. The basic data structure in this mode is equal to the correspondent structures mentioned earlier. The main difference is that, for each step, data for one or several returns comes separated by a '&' character. An example for this structure can be consulted in Figure 3.11



(a) 2-Character encoding Distance only example



(b) Distance-Intensity Pair example

Figure 3.11: Multiecho structure

Message and Block Splitting

The Hokuyo sensors send the entire FoV scan in a single message. Depending on the sensors' FoV and angular resolution, this can represent that a long message is sent all at once. Another problem that arises is ~~that~~ when the sensor is in multiecho mode, the messages can vary in size. TCP has a feature that controls the window size that clients and servers use to communicate. This, in turn, also limits the packet size that the sensor can send to the host. So, depending on the total size of a scan message, the message can be divided into several packets and sent to the host system. Because of this, if there is no organizing mechanism, the data becomes very difficult to comprehend. As an organizing mechanism, SCIP implements a block splitting method, organizing data into blocks of 64 characters with individual check codes and response limiters. However, as the size of the messages is not always a multiple of 64, the last block may not have the complete length of a standard block. (Figure 3.12)

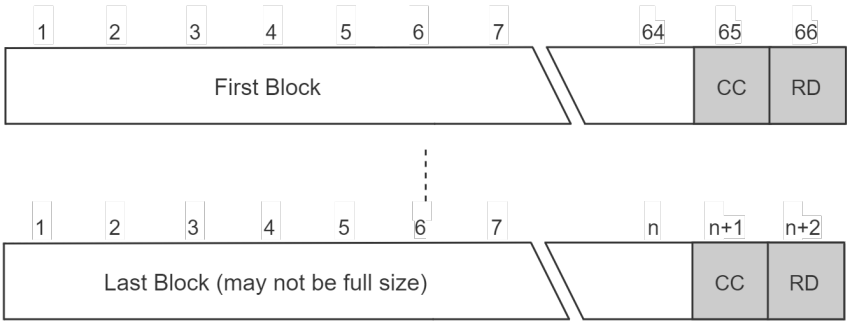


Figure 3.12: Block splitting structure

3.7 Velodyne Lidar

Velodyne Lidar provides powerful LiDAR solutions for autonomous vehicles, driver assistance systems, and various other fields. Velodyne Lidar has a vast list of products as well as hundreds of active customers, making their products a worldwide reference. Alongside this, there are multiple implementations of autonomous vehicles [71, 72] that use Velodyne sensors with great success. For these reasons, and for a greater impact, it was decided that the ALFA-Pi system would support Velodyne sensors.

3.7.1 Sensor's functioning

Velodyne's line-up of Mechanical Scanners are based on the ToF principle. All of the sensors are capable of producing 3D point clouds with a 360º horizontal FoV [73] featuring three return modes: Strongest Return, Last Return and Dual Return. The sensors also feature various configuration possibilities, accessible via a web-page hosted by the sensor. The first two modes only provide one return value for each emitter/receiver pair. However, the Dual Return mode provides the information on two returns that the emitter/receiver pair received. When in dual return the sensor returns both the strongest and latest return. The strongest return represents the return where the biggest amount of the pulse returned to the receiver and the last return represents the furthest point the pulse had to travel. An illustration of this operation can be observed in Figure 3.13

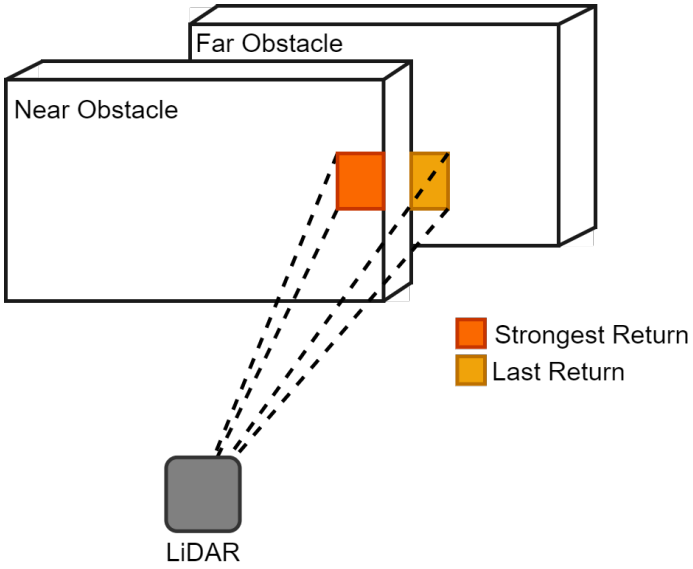


Figure 3.13: Dual Return functioning illustration

3.7.2 Data packet

Velodyne sensors use a proprietary protocol that is based on User Datagram Protocol (UDP). The sensors generate two types of packets: Position packets and Data packets. Position packets provide a copy of the last National Marine Electronics Association (NMEA) message received by the sensor from an external Global Positioning System (GPS) source. On the other hand, data packets contain the 3D data measured by the sensor, calibrated surface reflectivity, a set of azimuth values, time-stamping information, and sensor information. As the position packets are of no interest to this dissertation they will not be further discussed in this section.

The Data packets are divided into various constructs that help in the decoding and organization of data. The sensor reports distances relative to itself in spherical coordinates, consisting of radius (distance), elevation (vertical angle), and azimuth (horizontal angle).

- *Firing Sequence:* A firing sequence occurs every time all the lasers in a sensor are fired. Because of this, the firing sequence is highly dependent on the line and model of the sensor. For example, in the VLP16, a firing sequence corresponds to when all the 16 lasers are fired.

- *Laser Channel:* A laser channel represents a single emitter and detector pair. Each of these pairs is fixed at a particular vertical angle relative to the horizontal plane. Each laser channel has an ID. The ID of a data point can be inferred by the location of its data inside a data packet. Because of this, the elevation angle of a particular channel does not appear inside the data packets.

  In some sensors, the laser channels are not vertically aligned with each other. Instead, they have a specific and fixed azimuth offset to reduce possible cross-talk that may happen between channels. As an example, the first eight offsets for the VLS-128 sensor can be observed bellow (the missing IDs repeat the offset values in groups of eight).

- *Data Point:* A data point represents the information referent to a laser channel. A data point is represented by three bytes being the first two bytes distance information, and the last byte the calibrated reflectivity of the surface the emitted signal bounced off. The distance is an unsigned integer, and its granularity depends on the sensor. For example, the VLP-32C sensor, has a granularity of four millimeters, meaning that a value of 2500 represents a distance of 10000 millimeters.

Table 3.1: Azimuth Offset for the VLS-128

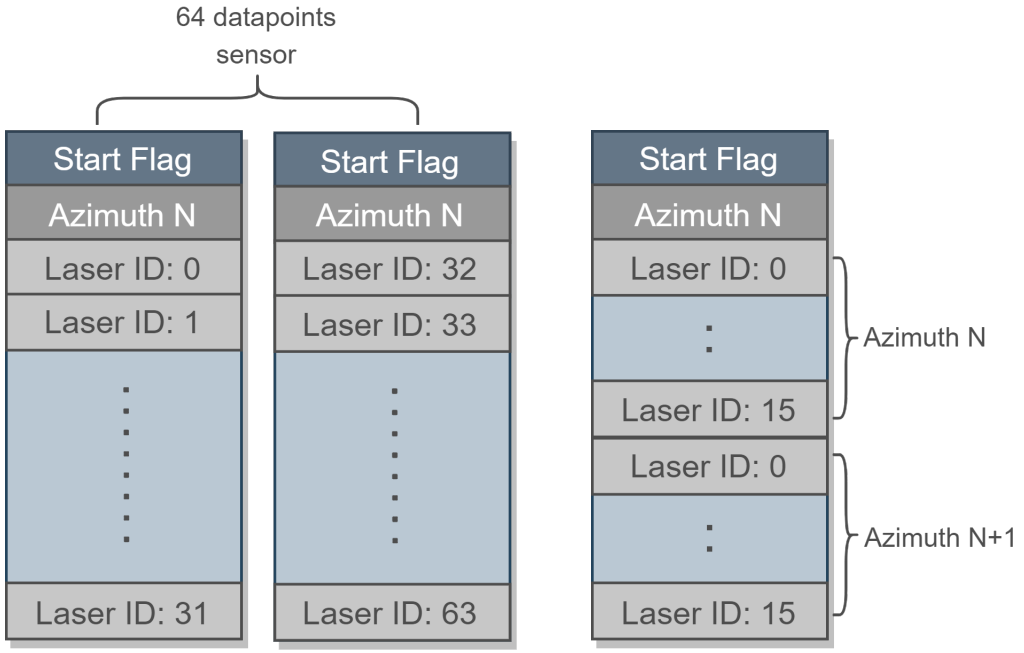
ID	Azimuth Offset (º)
0	-6.354
1	-4.548
2	-2.732
3	-0.911
4	0.911
5	2.732
6	4.548
7	6.354

The calibrated reflectivity is represented on a scale from 0 to 255. Values from 0-110 represent diffuse reflectors, and values from 111 to 255 describe a retroflector, where 255 is an ideal reflection.

- *Azimuth:* The azimuth is an unsigned integer represented by two bytes at the beginning of each data block. The value represents an angle in hundredths of a degree, allowing a high angular resolution without the need for floating points. Valid values range from 0 to 35999, and only one azimuth is reported per data block.

- *Data Block:* A data block is comprised of one hundred bytes of binary data and each data packet has twelve data blocks. The first two bytes of a data block are a flag, that helps understand when a new block starts followed by an azimuth value, and thirty two data points. The information inside this blocks comes in a little endian configuration.

  When in single return mode and with a sensor with more than 32 laser channels, consecutive data blocks hold the same azimuth value as the consecutive data blocks have the information on the missing data channels. In sensors with less than 32 laser channels, multiple azimuths can be inferred on the same data block. For example, on the Puck LITE (a sensor with 16 laser channels), the last 16 data points in a data block have an azimuth value equal to the base azimuth plus the angular resolution of the sensor. An example for both of this cases can be consulted in Figure 3.14.



(a) Two consecutive data blocks with a sensor of 64 channels

(b) Data block structure with a sensor of 16 channels

Figure 3.14: Data Block examples of sensors with more or less than 32 laser channels

- *Factory Bytes:* At the end of each packet a pair of bytes called the Factory Bytes can be found. The first byte has information on the Return Mode and the second indicates the model of the sensor that is sending the message. In table 3.2 the codes for the different sensors and modes can be observed.

Table 3.2: Factory Bytes meaning

Return Mode		**Product ID**	
Mode	**Value**	**Product Model**	**Value**
Strongest	0x37 (55)	HDL-32E	0x21 (33)
Last Return	0x38 (56)	VLP-16	0x22 (34)
Dual Return	0x39 (57)	Puck LITE	0x22 (34)
–	–	Puck Hi-Res	0x24 (36)
–	–	VLP-32C	0x28 (40)
–	–	Velarray	0x31 (49)
–	–	VLS-128	0xA1 (161)

3.7.2.1 Data Packet Structure

A data packet is composed of 1248 bytes and is sent via UDP on the configured port and address. Without the UDP header, the packet is divided into twelve data blocks, a timestamp, and two factory bytes. There are two possible data packet formats dependent

of the sensor's mode. There is a format for both the Single Return modes and one for the Dual Return mode. The main difference is that when in dual return mode, the sensor sends a pair of data blocks for each azimuth, where both data blocks contain information on the same laser channels unlike the cases specified in Figure 3.14, in which the laser channels were different in consecutive data blocks with the same azimuth. The odd-numbered blocks contain the strongest return, and the even-numbered blocks hold the last return (Figure 3.15). On the other hand, when the sensor is working in single return mode, the data blocks come ordered by their azimuth with no difference between even or odd-numbered data blocks.

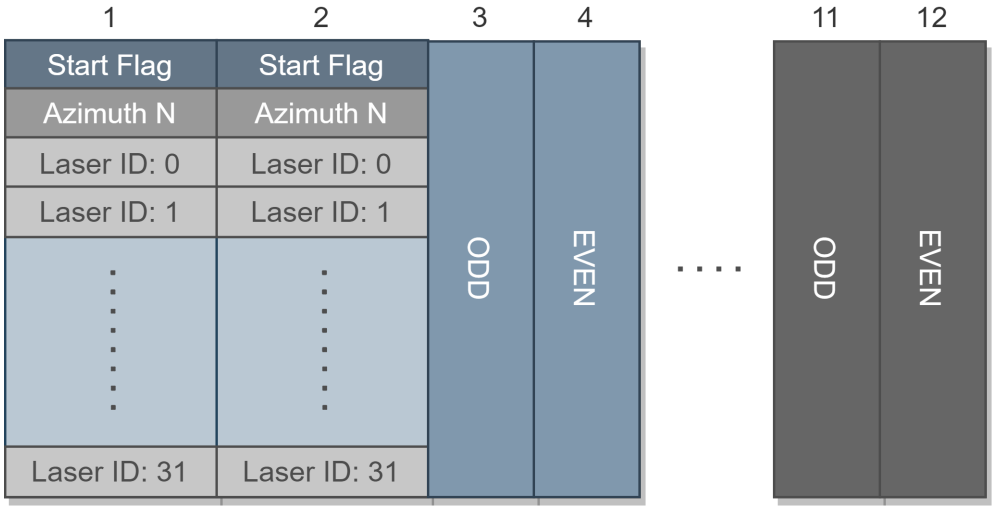


Figure 3.15: Dual Return packet structure

4. System Specification

This dissertation's main goal is to study and provide an FPGA-based approach for driving automotive LiDAR sensors that are Ethernet interface enabled. Figure 4.1 depicts the proposed system architecture, ALFA-Pi, which is responsible for the high-speed decoding and reading of 2D and 3D LiDAR data. To accomplish the proposed goal and provide an easily maintainable platform, the system was divided into several blocks, each responsible for performing a set of tasks.

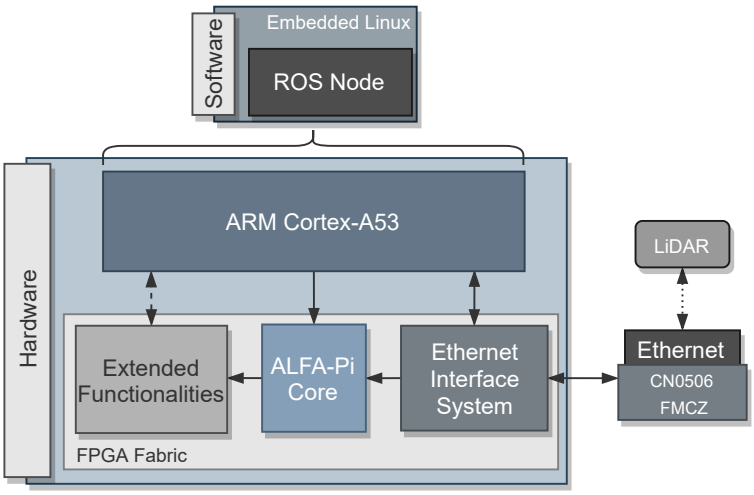


Figure 4.1: Proposed System Block Diagram

In the software domain, a ROS Node controls and configures the hardware Core according to the sensors parameters. Currently, ALFA does not have a complete software interface to control the hardware cores, however, the ROS Node presented can be later incorporated into the system with relative ease.

In the Hardware domain, the system can be divided into two main blocks: the ALFA-Pi Core and the Ethernet Interface System. The ALFA-Pi Core is responsible for receiving Ethernet data, filtering it, determining the sensor that sent data, decoding the received data, and outputting the decoded data in a general format. The Ethernet Interface System encapsulates all the necessary IP cores to have a fully functional MAC on the hardware

layer. Finally, some Extended Functionalities were implemented to show the versatility of the general output format that abstracts the IP Cores downstream from the sensor.

4.1 Ethernet Interface System

The first step in decoding LiDAR data is to receive the Ethernet data and to have it available at a hardware level. The Ethernet Interface System is the combination of multiple IP Cores that together accomplish this task. In following subsections, the design decisions to accomplish this task are explained.

4.1.1 MAC and PHY Connection

The first task in the Ethernet Interface System is to intercept Ethernet data as early as possible in the network stack. Figure 4.2 describes the Open Systems Interconnection (OSI) model, which characterizes and standardizes the communication functions of a telecommunication system, whatever its underlying internal structure and technology are. To have the biggest possible flexibility, the top layers of the network stack (3 and up) are deployed in software using Linux. For this reason and to bypass delays introduced by the operating system, it is necessary to intercept the Ethernet data between layers two and three of the OSI model, allowing the abstraction of the Ethernet framing protocols and error detection of these frames.

Figure 4.2: OSI Model

The used ZCU104 board uses the TIDP83867IRPAP Ethernet RGMII PHY for Ethernet communications and a Bel Fuse L829-1J1T-43 RJ-45 connector with built-in magnetics and LED indicators [**?**]. The PHY is connected to a Gigabit Ethernet Controller (GEM) available in the PS through a Multiplexed Input/Output (MIO) Interface meaning, that to access the Ethernet Data passed through this port on the PL it is necessary to implement a communication system between the PS and the PL. Not only does this introduce unwanted delays, but it is also very inefficient as data has to enter the PS before entering the PL. To solve this, the EVAL-CN0506-FMCZ is used.

The EVAL-CN0506-FMCZ provides two PHY that can be directly accessed through the use of an FMC connector. To control these PHY, handle Ethernet framing protocols and error detection, the AXI 1G/2.5G Ethernet Subsystem is used. Both of these components capabilities and uses were explored in sections 3.3 and 3.2.3, respectively.

In the next section, the MAC present inside the AXI 1G/2.5G Ethernet Subsystem IP Core, is referred simply as MAC, to simplify the reading. Figure 4.3 depicts how both systems are connected. Firstly, the RGMII interface was chosen as it is the only one capable of supporting data rates of 1Gb/s, necessary for connecting one of the supported sensors. When using the RGMII interface for 1Gb/s speeds, the MAC must provide a transmit clock signal of 125MHz. So, instead of creating a clock in the FPGA the clock provided by the EVAL-CN0506-FMCZ board was used to feed the MAC, consequently feeding the transmit clock signal. Finally, and as it is the default, the MDIO interface of the MAC is connected to the MDIO interface of the PHY.

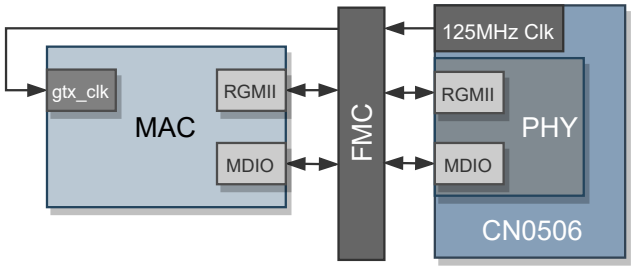


Figure 4.3: EVAL-CN0506-FMCZ and MAC Connection Diagram

Finally, to connect the MAC ports to the correct FMC physical pins on the ZCU104 board, a constraint file was necessary. In Vivado Design Suite, a constraint file determines to which pins the IP Cores ports connect and how they connect (voltage, I/O Standards). The EVAL-CN0506-FMCZ can use 1.8V, 2.5V, and 3.3V of logical levels, however, the maximum voltage level of the ZCU104 is 1.8V, so this was the chosen voltage level. In Table 4.1 the used constraints can be observed.

4.1.2 Configuration and Data Movement

After making the necessary MAC and PHY connections, it was necessary to design a solution to make Ethernet data arrive at the actual decoding system. As mentioned earlier, letting the software layer have access and handle the received Ethernet data is an advantage as it provides easier debug of sensor outputs or any other kind of data the board might receive. Also, both line-ups of supported sensors (Hokuyo and Velodyne Lidar) support configuration over the network. Not only are these configurations easier to implement in software, but also higher flexibility should be achievable. Finally, Xilinx provides device drivers to have the AXI 1G/2.5G Ethernet Subsystem serve as MAC, easing the implementation process into a Linux distribution. However, this device driver is dependent on the use of a DMA engine to receive and send Ethernet data via the AXI 1G/2.5G Ethernet Subsystem. So, two solutions can be devised: (1) have a custom IP between the AXI 1G/2.5G Ethernet Subsystem and the DMA and handle the decoding

Table 4.1: Constraints Used for the FMC Connector

FMC Pin	XCZU7EV U1	Function	I/O Standard
H17	A12	MDC	LVCMOS18
H16	A13	MDIO_IO	LVCMOS18
H19	D16	RESET	LVCMOS18
H14	J15	rx_ctl	LVCMOS18
G6	F17	rx_clk	LVCMOS18
H13	J16	tx_ctl	LVCMOS18
H11	L16	tx_clk	LVCMOS18
G10	K18	rgmii_rx_data[3]	LVCMOS18
G9	K19	rgmii_rx_data[2]	LVCMOS18
H8	K20	rgmii_rx_data[1]	LVCMOS18
H7	L20	rgmii_rx_data[0]	LVCMOS18
C11	G19	rgmii_tx_data[3]	LVCMOS18
C10	H19	rgmii_tx_data[2]	LVCMOS18
D15	G16	rgmii_tx_data[1]	LVCMOS18
D14	H16	rgmii_tx_data[0]	LVCMOS18
H4	E15	125_clk_p	LVDS
H5	E14	125_clk_n	LVDS

there; (2) broadcast the received data to the ALFA-Pi Core and the DMA at the same time.

The second solution was the one implemented and used in the final system. On the first solution, the custom IP would need to regulate the DMA transfers and provide some data buffering. Not only is this much more resource-intensive, but it also introduces bigger delays than the second solution. In Figure 4.4 can be observed that the output data of the AXI 1G/2.5G Ethernet Subsystem IP Core is fed to an AXI4-Stream Broadcaster IP Core. As mentioned in section 3.2.5, this IP enables the connection of one AXI Stream Master to multiple AXI Stream Slaves. This way, the data that previously went directly to the DMA and consequently to the DDR4 memory is ”duplicated” and also sent to the ALFA-Pi Core. This design assures that the actual controller of data transfer, to and from the DDR4 memory, continues to be the AXI 1G/2.5G Ethernet Subsystem IP Core and the DMA. The only downside of this solution is the one or two clock cycle delay introduced by the AXI4-Stream Broadcaster IP Core. Finally, as only the received data is necessary by the ALFA-Pi Core, the transmission of data through the Ethernet Interface is handled exclusively by the PS through the DMA connected to the data input port of the AXI 1G/2.5G Ethernet Subsystem IP Core, with no interference.

To configure both the AXI DMA and the AXI 1G/2.5G Ethernet Subsystem IP Cores, an AXI Lite interface is used. Each of these cores has a slave interface connected to an AXI High-Performance Master in the Full power domain. The communication is mediated

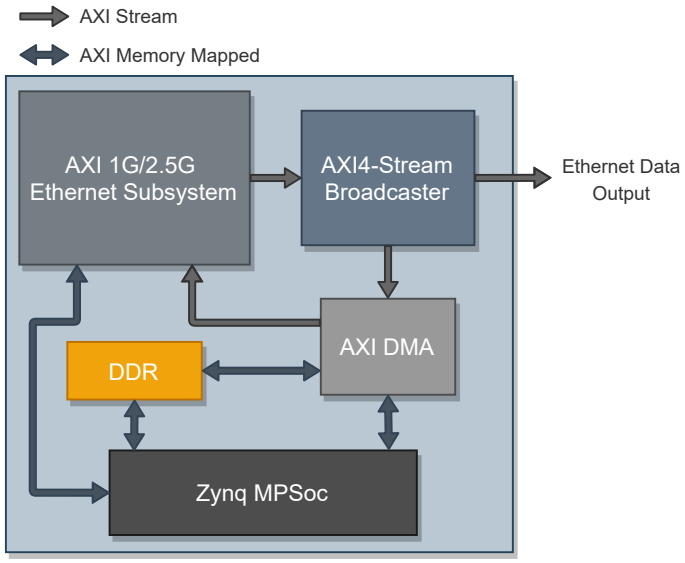


Figure 4.4: Ethernet Interface System Block Diagram

by an AXI Interconnect that handles the routing and buffering of data between all these interfaces. So, the configurations are performed by the PS using Xilinx device drivers, which will be approached later.

4.2 Hokuyo

In this section, the hardware design decisions for the Hokuyo controller will be explained. This controller has the task to transform the Ethernet data received from Hokuyo sensors into azimuth, distance, and intensity information. For simpler implementation and upgradability, the controller was divided into several components that perform specific tasks. In Figure 4.5, a big picture overview of the controller can be observed.

As can be observed in the previous Figure, the controller module has several buffers inside of it. Multiple buffers are needed because a scan message from an Hokuyo sensor can be distributed through various TCP/IP packets. So, before decoding a message from the sensor, the multiple packets must first be received and stored together. Storing all the packets for a specific sensor together inside the same buffer can mean more memory usage, but that's a trade-off for much easier control and faster operation. The controller also has a built-in register bank to store information about each Hokuyo sensor.

To manage this, each sensor has a controller ID associated with it that indicates the buffer where the packets are stored and where in the register bank its information is stored. The ID information from the sensor comes from the Ethernet Filter. To link a packet directly to its sensor ID information a control sequence is placed by the Ethernet Filter at the beginning of each transmission. The first sixteen bits of each packet contains

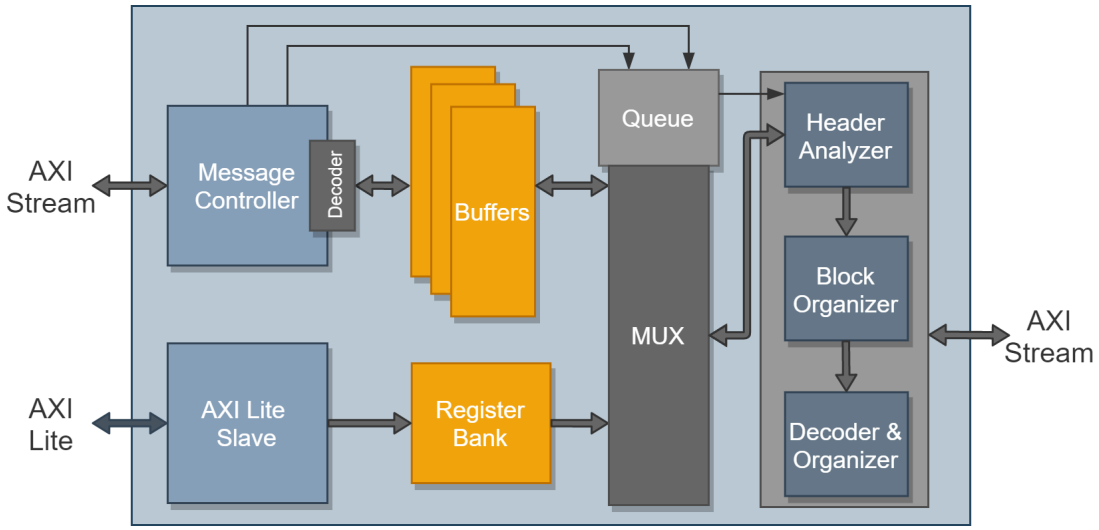


Figure 4.5: Hokuyo Controller System Overview

control flags to indicate that the next two bytes represent ID information, followed by the controller ID and sensor ID. In Figure 4.6, a diagram of the packet structure can be seen.

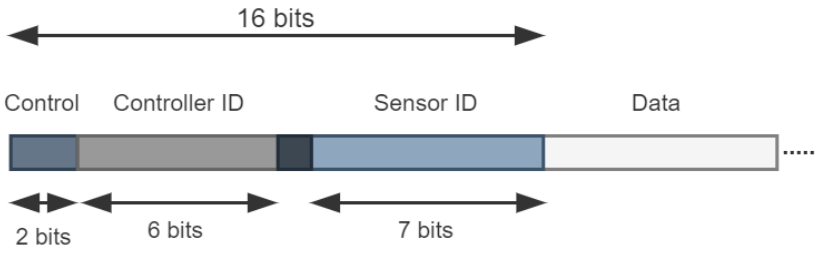


Figure 4.6: Control segment structure

4.2.1 Message Controller

This module is responsible for deciding which buffer receives the current packet and when an entire message has been received. In Figure 4.7, an overview of the module's interfaces can be observed. These interfaces are an AXI Stream Slave as input and an AXI Stream Master, and two data signals as outputs.

To receive data, the Message Controller puts its *tready* flag at one, signaling that it is ready to receive data. When it detects a control sequence (explained above), the module extracts the controller ID and feeds it into a decoder, as the select signal, via the *controller_id* output. This decoder will then redirect data to the buffer that corresponds to the controller ID indicated. The module then connects the buffer's *tready* signal (present in the Master interface of the Message Controller) to the AXI Stream slave interface *tready*

signal, making the Message Controller act as a passthrough. For the correct setting of the decoder registers, the entire AXI Stream transmission is delayed one clock cycle.

The Message Controller also has the task of detecting when an entire message is received. In SCIP2.0, a message is considered finished when two RD characters are received consecutively. So, when the module detects this, it sends a pulse in the *end_of_frame* output.

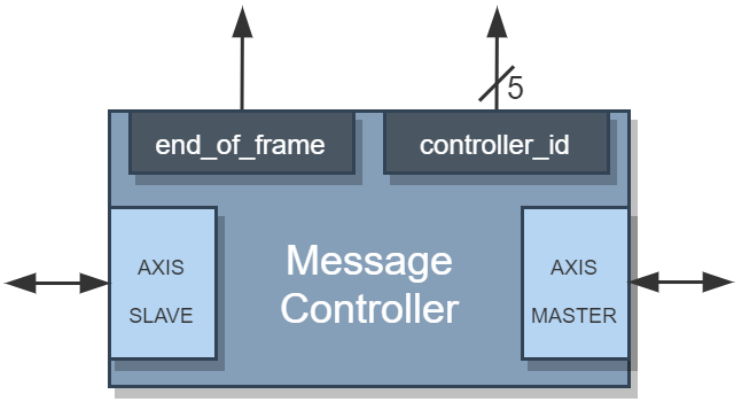


Figure 4.7: Message Controller Interface Overview

4.2.2 AXI Lite Interface

Hokuyo sensors can function in various modes that affect the amount of data received, which data is received and how it is received. To enable the use of various modes of the same sensor without any hardware change, a control and customization interface was needed. So, to enable real time control an AXI Lite Interface was implemented. This will allow the use of software in the upper layers to customize the mode in which the Hokuyo module decodes data, as well as the introduction of some error detection mechanisms.

The AXI Lite interface is based on the use of registers. Each position on this interface corresponds to a register. So, instead of having multiple positions referent to the same sensor, an outside register bank is used. This way, the AXI Lite Interface only needs to use two registers instead of two registers per sensor, abstracting the software from the memory layout used in the hardware. Each position (register) of the AXI Lite Interface has thirty two bits of space. Using two registers for this transmission allows the hardware to receive the first four characters of the used command (used for error detection), the start step and the mode used. In Figure 4.8, the layout for the message can be seen. After receiving a message, this module uses the controller ID to load the values into the correct position inside the register bank.

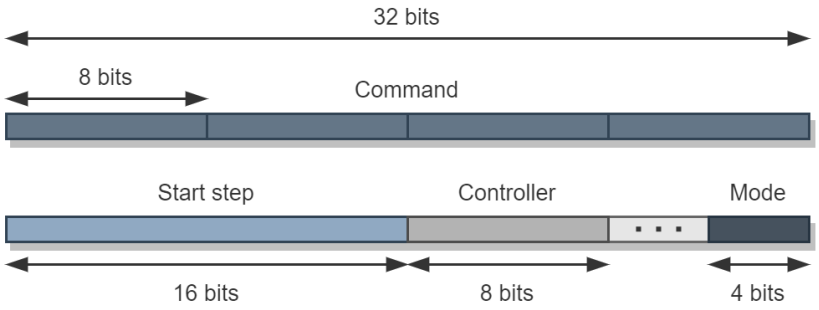


Figure 4.8: Control segment structure

In Figure 4.9, the mode field structure can be observed. This field is composed of three sections. This three sections have all the information the hardware needs to decode the messages correctly.

- *Pair:* The first bit and field, has information about which data comes inside the payload. The Hokuyo, can send distance information or distance-intensity information. When this bit has the value one, it means the the sensor is sending distance and intensity, otherwise only distance is being sent.
- *ME:* This second field indicates if the sensor is in Multi-echo mode, as this affects how data comes distributed inside the payload.
- *Enconding:* The last field has the information about the type of encoding the sensor is using. Depending on the command, the sensor can be using two character encoding and three character encoding.

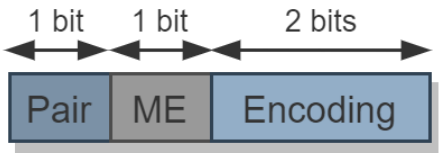


Figure 4.9: Mode field structure

4.2.3 Queue & Multiplexer

The names of these modules indicate their functioning. An overview of the interfaces on these modules can be seen in Figure 4.10. The queue stores information about which buffers have received an entire message and are waiting for the processing data. As multiple sensors can be added to the system, there was the possibility of missing a signal from the Message Controller, indicating that an entire message arrived. This could happen

either because the outputs were stalled (data not being consumed) or because receiving a packet takes less time than processing an entire message. When the Message Controller sends a pulse by the *end_of_frame*, the Queue module adds the controller ID present at the *controller_id* input at the end of the queue. In turn, the Queue module connects to the Header Analyzer and notifies it when it is not empty, setting the *not_empty* signal. Finally, the module outputs via the *o_controller_id* the value that is present on the first position of the queue, selecting from one of the multiplexer inputs.

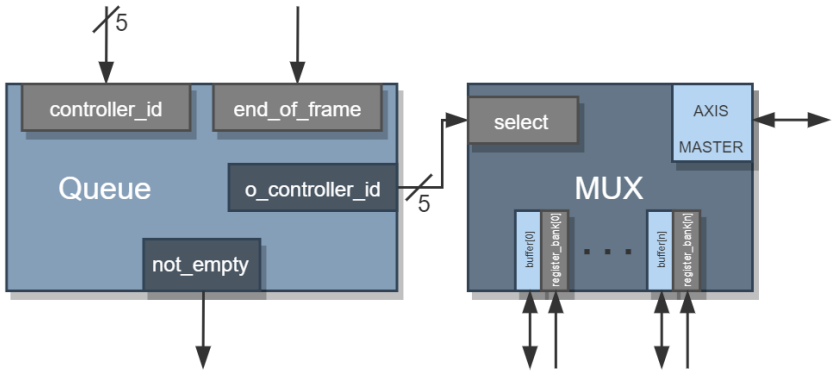


Figure 4.10: Queue and MUX interface

4.2.4 Header Analyzer

The Header Analyzer is the first component of the module that does the decoding and organization of data. It is responsible for analyzing the header of the scan messages, detecting if the command received is correct, detecting response messages, and detecting errors in the middle of the scan message. In Figure 4.11, the state machine that drives this module can be observed.

- *IDLE:* The IDLE state is where the state machine stays until it is triggered. It does nothing apart from the cleaning of some variables that are needed in later states. The Header Analyzer only exits this state when the value of the start signal becomes one, and this occurs when the queue mentioned earlier is not empty, meaning that there is data to be analyzed.

- *TYPE DETECTOR:* This state is responsible for detecting what type of message was received and filtering out the header. To identify the header, the last four bytes that the module received are compared to the command characters received by the AXI Lite Interface. When a match is detected, the flag *header_detected* is set to one, and the TYPE DETECTOR starts its regular operation. This detection makes

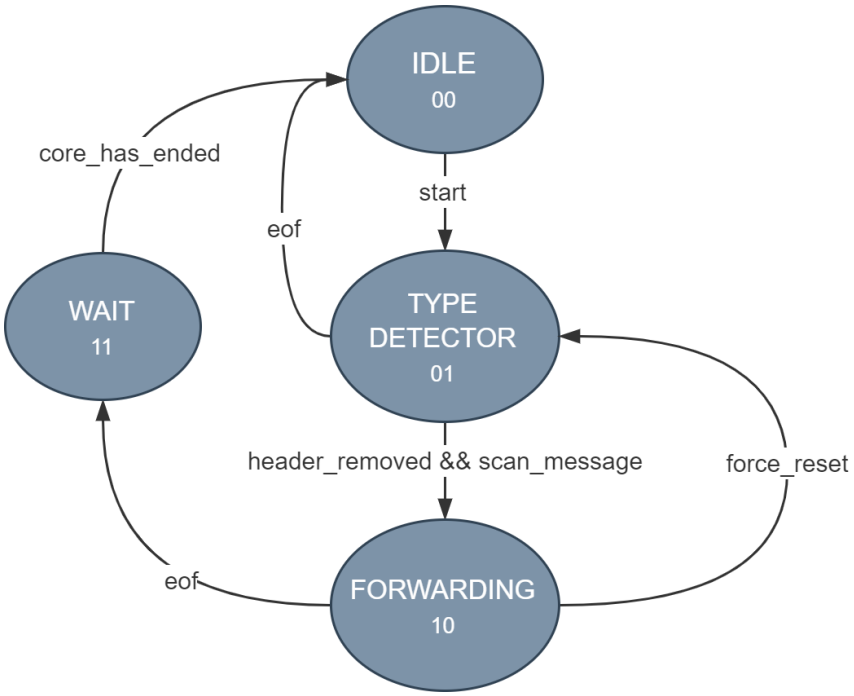


Figure 4.11: Control segment structure

sure that if the sensor has a different mode than the one in the hardware, the sensor messages are ignored, and thus, no errors occur further ahead.

In some commands, the sensor sends a response message before sending the scan response messages. As these messages do not contain any valuable data for the controller, they can be discarded. So, after detecting the start of a header, the TYPE DETECTOR waits until it receives two RD characters. If after the second character, the module detects a consecutive RD character, the *eof* flag is set, and the state machine returns to IDLE. Receiving two consecutive RD characters indicates that the message has ended, meaning that a response message was received.

If the module does not identify another RD character, it means the message received is a scan response message, and the flag *scan_message* is set. Despite the beginning of a message being different in every command, after the second RD character, the sensor always sends six bytes referent to time stamping information. When these six bytes have been counted, the *header_removed* flag is set to one, and the module changes state.

- *FORWARDING:* In this state the Header Analyzer simply receives data from the buffers and forwards it to the next module. However the last four bytes are continuously being monitored and compared to the command characters. If it detects another header, it immediately forces a reset to all modules that come upstream and goes back to the TYPE DETECTOR state. This can occur if, for example,

the transmission was interrupted mid way and restarted, which means that data from the failed transmission would still be inside the buffers and thus needs to be discarded.

- *WAIT:* This state acts as a transitioning state. The queue only advances when a message has been processed, and as there is a delay between the Header Analyzer module and the Decoder & Organizer, there was a need for a state that is just waiting for the other modules to end.

4.2.5 Block Organizer

This module is responsible for verifying the checksum codes on each block of data. As mentioned before, data comes organized in blocks of sixty six bytes divided by a checksum byte and an RD character. The checksum is used to detect problems that may have occurred during data transmission. The mechanism to calculate the checksum was already explained in the section 3.6.2.

4.2.6 Decoder & Organizer

After verifying the correctness of the data and filtering non-important one, the last module is responsible for decoding and organizing data into the final structure. As mentioned earlier, which data comes and how it comes depends on the command the sensor is running on. There are two main ways data comes arranged, and two secondary ways. This is considered like this because the two secondary ways come encapsulated inside the main ones. The main structures data can come organized in are multi-echo mode or not. So, the module is separated into two data flows. Despite this, both flows use the same decoder sub-module. This sub-module is only responsible for receiving data and outputting the decoded form of it. As data is received byte by byte for easier control, the decoding type decides when the decoder sub-module is activated. So, a counter that counts how much data has been received is used. When this counter equals the amount of data needed by the decoder, the sub-module is activated. When the decoder reports it is done, data is arranged in the correct places of the output buffer. The exact position of data depends on the secondary structure the sensor is using. If the sensor is in distance-intensity pair mode, this operation will have to be done twice for each data point, as there is the need to extract both the distance and intensity, otherwise this will only be performed once per data point.

Finally, if the sensor is in non-multi-echo mode the operation mentioned earlier will only happen once, otherwise it will happen twice. When the sensor is in multi-echo mode, it can report multiple returns for the same horizontal angle. The data structure used for

the output only supports two returns per angle so only the first two returns are considered, and the rest is ignored. When the module detects it is at the last output word, it activates the *end_frame* portion of the output structure.

4.3 Velodyne

4.4 ”Encapsulator”

4.5 Software

4.5.1 Software Stack

4.5.1.1 Drivers

5. Implementation

5.1 Hardware

5.2 Software

6. Evaluation and Results

6.1 Test Setup

6.2 Results

7. Conclusion

7.1 Future Work

References

[1] "VLP-16 User Manual 63-9243 Rev. D," 2018.

[2] R. Roriz, J. Cabral, and T. Gomes, "Automotive LiDAR Technology: A Survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. PP, no. 2, pp. 1–16, 2021.

[3] "Axi 1g/2.5g ethernet subsystem v7.2: Product guide." `https://www.xilinx.com/content/dam/xilinx/support/documentation/ip_documentation/axi_ethernet/v7_2/pg138-axi-ethernet.pdf`.

[4] S. Royo and M. Ballesta-Garcia, "An overview of lidar imaging systems for autonomous vehicles," *Applied Sciences (Switzerland)*, vol. 9, no. 19, 2019.

[5] C. Weitkamp, *Lidar : range-resolved optical remote sensing of the atmosphere*. Springer, 2005.

[6] E. Synge, " XCI. A method of investigating the higher atmosphere ," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 9, no. 60, pp. 1014–1020, 1930.

[7] M. A. Tuve, E. A. Johnson, and O. R. Wulf, "A new experimental method for study of the upper atmosphere," *Terrestrial Magnetism and Atmospheric Electricity*, vol. 40, pp. 452–454, 12 1935.

[8] W. E. K. Middleton and A. F. Spilhaus, "Meteorological Instruments," *Quarterly Journal of the Royal Meteorological Society*, vol. 80, p. 484, 7 1954.

[9] T. Maiman, "Stimulated Optical Radiation in Ruby," *Nature*, vol. 187, no. 1959, pp. 493–494, 1960.

[10] G. F. Smith, "The Early Laser Years at Hughes Aircraft Company," *IEEE Journal of Quantum Electronics*, vol. 20, no. 6, pp. 577–584, 1984.

[11] E. D. Hinkley, *Laser Monitoring of the Atmosphere*. Springer-Verlag Berlin Heidelberg, 1 ed., 1976.

[12] M. E. Warren, "Automotive LIDAR Technology," *2019 Symposium on VLSI Circuits*, vol. 1, pp. 254–255, 2019.

[13] S. Patil, B. Singh, D. Livezey, S. Ahmad, and M. Margala, "Functional Safety of a Lidar Sensor System," *Midwest Symposium on Circuits and Systems*, vol. 2020-August, no. 2, pp. 45–48, 2020.

[14] J. Lambert, A. Carballo, A. M. Cano, P. Narksri, D. Wong, E. Takeuchi, and K. Takeda, "Performance Analysis of 10 Models of 3D LiDARs for Automated Driving," *IEEE Access*, vol. 8, pp. 131699–131722, 2020.

[15] B. Behroozpour, P. A. Sandborn, M. C. Wu, and B. E. Boser, "Lidar System Architectures and Circuits," *IEEE Communications Magazine*, vol. 55, no. 10, pp. 135–142, 2017.

[16] D. Gatziolis and H. E. Andersen, "A guide to LIDAR data acquisition and processing for the forests of the pacific northwest," *USDA Forest Service - General Technical Report PNW-GTR*, no. 768, pp. 1–32, 2008.

[17] P. Mcmanamon, P. Banks, J. Beck, D. Fried, A. Huntington, and E. Watson, "Comparison of flash lidar detector options," *Optical Engineering*, vol. 56, p. 31223, 3 2017.

[18] N. Muhammad and S. Lacroix, "Calibration of a rotating multi-beam Lidar," *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*, pp. 5648–5653, 2010.

[19] H. W. Yoo, N. Druml, D. Brunner, C. Schwarzl, T. Thurner, M. Hennecke, and G. Schitter, "MEMS-based lidar for autonomous driving," *Elektrotechnik und Informationstechnik*, vol. 135, no. 6, pp. 408–415, 2018.

[20] B. L. Stann, J. F. Dammann, and M. M. Giza, "Progress on MEMS-scanned ladar," in *Proc.SPIE*, vol. 9832, 5 2016.

[21] G. Kim, J. Eom, and Y. Park, "Design and implementation of 3D LIDAR based on pixel-by-pixel scanning and DS-OCDMA," in *Proc.SPIE*, vol. 10107, 2 2017.

[22] M. J. Heck, "Highly integrated optical phased arrays: Photonic integrated circuits for optical beam shaping and beam steering," *Nanophotonics*, vol. 6, no. 1, pp. 93–107, 2017.

[23] K. Van Acoleyen, W. Bogaerts, J. Jágerská, N. Le Thomas, R. Houdré, and R. Baets, "Off-chip beam steering with a one-dimensional optical phased array on silicon-on-insulator," *Optics Letters*, vol. 34, p. 1477, may 2009.

[24] C. H. Jang, C. S. Kim, K. C. Jo, and M. Sunwoo, "Design factor optimization of 3D flash lidar sensor based on geometrical model for automated vehicle and advanced driver assistance system applications," *International Journal of Automotive Technology*, vol. 18, pp. 147–156, feb 2017.

[25] A. Gelbart, B. Redman, R. Light, C. Schwartzlow, and A. Griffis, "Flash lidar based on multiple-slit streak tube imaging lidar," *Proc SPIE*, vol. 4723, 7 2002.

[26] M. Kumar, A. K. Verma, and A. Srividya, "Response-time modeling of controller area network (CAN)," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5408 LNCS, pp. 163–174, Springer, Berlin, Heidelberg, 2009.

[27] M. Sauerwald, "CAN bus, Ethernet, or FPD-Link: Which is best for automotive communications?," *Analog Applications Journal*, pp. 20–22, 2014.

[28] J. Hu and C. Xiong, "Study on the embedded CAN bus control system in the vehicle," in *Proceedings - 2012 International Conference on Computer Science and Electronics Engineering, ICCSEE 2012*, vol. 2, pp. 440–442, 2012.

[29] "Leddar is16." `https://leddartech.com/solutions/is16-industrial-sensor`. Accessed: 2021-03-26.

[30] R. Santitoro, "Metro Ethernet Services – A Technical Overview What is an Ethernet Service ?," *Metro*, pp. 1–19, 2003.

[31] J. Sommer, S. Gunreben, F. Feller, M. Köhn, A. Mifdaoui, D. Saß, and J. Scharf, "Ethernet - A survey on its fields of application," *IEEE Communications Surveys and Tutorials*, vol. 12, no. 2, pp. 263–284, 2010.

[32] Buehler, Martin, K. Iagnemma, and S. Singh, *The DARPA Urban Challenge*, vol. 56. 2009.

[33] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. N. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. M. Howard, S. Kolski, A. Kelly, M. Likhachev, M. McNaughton, N. Miller, K. Peterson, B. Pilnick, R. Rajkumar, P. Rybski, B. Salesky, Y. W. Seo, S. Singh, J. Snider, A. Stentz, W. Whittaker, Z. Wolkowicki, J. Ziglar, H. Bae, T. Brown, D. Demitrish, B. Litkouhi, J. Nickolaou, V. Sadekar, W. Zhang, J. Struble, M. Taylor, M. Darms, and D. Ferguson, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.

[34] L. Chappell, "The Big Bang of autonomous driving," 2016.

[35] J. Guerrero-Ibáñez, S. Zeadally, and J. Contreras-Castillo, "Sensor technologies for intelligent transportation systems," apr 2018.

[36] E. Martí, M. Á. De Miguel, F. García, and J. Pérez, "A Review of Sensor Technologies for Perception in Automated Driving," *IEEE Intelligent Transportation Systems Magazine*, vol. 11, pp. 94–108, 12 2019.

[37] S. Weng, J. Li, Y. Chen, and C. Wang, "Road traffic sign detection and classification

from mobile LiDAR point clouds," *2nd ISPRS International Conference on Computer Vision in Remote Sensing (CVRS 2015)*, vol. 9901, no. Cvrs 2015, p. 99010A, 2016.

[38] S. Gargoum, K. El-Basyouny, J. Sabbagh, and K. Froese, "Automated highway sign extraction using lidar data," *Transportation Research Record*, vol. 2643, pp. 1–8, 2017.

[39] L. Zhou and Z. Deng, "LIDAR and vision-based real-time traffic sign detection and recognition algorithm for intelligent vehicle," *2014 17th IEEE International Conference on Intelligent Transportation Systems, ITSC 2014*, pp. 578–583, 2014.

[40] H. Guan, W. Yan, Y. Yu, L. Zhong, and D. Li, "Robust Traffic-Sign Detection and Classification Using Mobile LiDAR Data with Digital Images," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 11, no. 5, pp. 1715–1724, 2018.

[41] P. Sun, X. Zhao, Z. Xu, R. Wang, and H. Min, "A 3D LiDAR Data-Based Dedicated Road Boundary Detection Algorithm for Autonomous Vehicles," *IEEE Access*, vol. 7, pp. 29623–29638, 2019.

[42] B. Yang, Z. Wei, Q. Li, and J. Li, "Automated extraction of street-scene objects from mobile lidar point clouds," *International Journal of Remote Sensing*, vol. 33, no. 18, pp. 5839–5861, 2012.

[43] T. Li and D. Zhidong, "A new 3D LIDAR-based lane markings recognition approach," *2013 IEEE International Conference on Robotics and Biomimetics, ROBIO 2013*, no. December, pp. 2197–2202, 2013.

[44] H. Lee, S. Kim, S. Park, Y. Jeong, H. Lee, and K. Yi, "AVM/LiDAR Sensor based Lane Marking Detection Method for Automated Driving on Complex Urban Roads," *2017 IEEE Intelligent Vehicles Symposium (IV)*, no. Iv, pp. 1434–1439, 2017.

[45] B. Li, T. Zhang, and T. Xia, "Vehicle detection from 3D lidar using fully convolutional network," in *Robotics: Science and Systems*, vol. 12, 2016.

[46] D. Gohring, M. Wang, M. Schnurmacher, and T. Ganjineh, "Radar/Lidar sensor fusion for car-following on highways," *ICARA 2011 - Proceedings of the 5th International Conference on Automation, Robotics and Applications*, pp. 407–412, 2011.

[47] G. H. Lee, J. D. Choi, J. H. Lee, and M. Y. Kim, "Object Detection Using Vision and LiDAR Sensor Fusion for Multi-channel V2X System," *2020 International Conference on Artificial Intelligence in Information and Communication, ICAIIC 2020*, pp. 1–5, 2020.

[48] P. Caillet and Y. Dupuis, "Efficient LiDAR data compression for embedded V2I or V2V data handling," apr 2019.

[49] M. M. Abdelwahab, W. S. El-Deeb, and A. A. Youssif, "LIDAR data compression challenges and difficulties," *2019 5th International Conference on Frontiers of Signal Processing, ICFSP 2019*, pp. 111–116, 2019.

[50] I. Maksymova, C. Steger, and N. Druml, "Review of LiDAR Sensor Data Acquisition and Compression for Automotive Applications," *Proceedings*, vol. 2, p. 852, dec 2018.

[51] Y. Takayanagi and M. Ichikawa, "Up-to-date trend of real-time ethernet for industrial automation systems," *ICCAS-SICE 2009 - ICROS-SICE International Joint Conference 2009, Proceedings*, pp. 4595–4598, 2009.

[52] L. L. Bello, "Novel trends in automotive networks: A perspective on Ethernet and the IEEE Audio Video Bridging," *19th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2014*, 2014.

[53] T. Nolte, H. Hansson, and L. Lo Bello, "Automotive communications - Past, current and future," in *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, vol. 1 2 VOLS, pp. 985–992, 2005.

[54] A. Kern, T. Streichert, and J. Teich, "An automated data structure migration concept - From CAN to Ethernet/IP in automotive embedded systems (CANoverIP)," in *Proceedings -Design, Automation and Test in Europe, DATE*, pp. 112–117, 2011.

[55] L. L. Bello, "The case for ethernet in automotive communications," *ACM SIGBED Review*, vol. 8, no. 4, pp. 7–15, 2011.

[56] BMW Group, "Security in embedded ip-based systems," 2009.

[57] Rohde and Schwarz, "Automotive ethernet: The future for in-vehicle networks." URL: `https://cdn.rohde-schwarz.com/fr/general_37/local_webpages/2019__automotive_tech_day_3_0/13_RS_Automotive_Ethernet_The_Future_for_In_Vehicle_Networks.pdf`, 5 2019.

[58] L. J. Zheng and Y. C. Fan, "Data packet decoder design for LiDAR system," *2017 IEEE International Conference on Consumer Electronics - Taiwan, ICCE-TW 2017*, no. 2, pp. 35–36, 2017.

[59] Y. C. Fan, L. J. Zheng, and Y. C. Liu, "3D Environment Measurement and Reconstruction Based on LiDAR," *I2MTC 2018 - 2018 IEEE International Instrumentation and Measurement Technology Conference: Discovering New Horizons in Instrumentation and Measurement, Proceedings*, pp. 1–4, 2018.

[60] A. Choudhary, D. Porwal, and A. Parmar, "FPGA Based Solution for Ethernet Controller As Alternative for TCP/UDP Software Stack," in *2018 6th Edition of International Conference on Wireless Networks and Embedded Systems, WECON 2018 - Proceedings*, pp. 63–66, Institute of Electrical and Electronics Engineers Inc.,

nov 2018.

[61] T. Uchida, "Hardware-based TCP processor for gigabit ethernet," in *IEEE Transactions on Nuclear Science*, vol. 55, pp. 1631–1637, jun 2008.

[62] M. R. Mahmoodi, S. M. Sayedi, and B. Mahmoodi, "Reconfigurable hardware implementation of gigabit UDP/IP stack based on spartan-6 FPGA," in *Proceedings - 2014 6th International Conference on Information Technology and Electrical Engineering: Leveraging Research and Technology Through University-Industry Collaboration, ICITEE 2014*, Institute of Electrical and Electronics Engineers Inc., jan 2014.

[63] M. Edwards, "Software acceleration using coprocessors: is it worth the effort?," in *Proceedings of 5th International Workshop on Hardware/Software Co Design. Codes/CASHE '97*, pp. 135–139, 1997.

[64] L. Bai, Y. Lyu, X. Xu, and X. Huang, "PointNet on FPGA for Real-Time LiDAR Point Cloud Processing," *arXiv*, 2020.

[65] "Amba specifications." `https://developer.arm.com/architectures/system-architectures/amba/specifications`. Accessed: 2021-10-30.

[66] "Amba axi and ace protocol specification axi3, axi4, and axi4-lite ace and ace-lite." `https://developer.arm.com/documentation/ihi0022/e?_ga=2.67820049.1631882347.1556009271-151447318.1544783517`. Accessed: 2021-10-30.

[67] "Amba axi3 and axi4 protocol specification." `https://developer.arm.com/documentation/ihi0022/e/AMBA-AXI3-and-AXI4-Protocol-Specification?lang=en`. Accessed: 2021-10-30.

[68] "Amba axi4-lite interface specification." `https://developer.arm.com/documentation/ihi0022/e/AMBA-AXI4-Lite-Interface-Specification?lang=en`. Accessed: 2021-10-30.

[69] "Amba axi4-stream protocol specification." `https://developer.arm.com/documentation/ihi0051/a?lang=en`. Accessed: 2021-10-30.

[70] "Axi dma v7.1 logicore ip: Product guide." `https://www.xilinx.com/support/documentation/ip_documentation/axi_dma/v7_1/pg021_axi_dma.pdf`.

[71] "Local motors: Sustainable, accessible transportation solution for all." `https://velodynelidar.com/case-studies/local-motors/`. Accessed: 2021-10-27.

[72] "Velodyne lidar announces autonomous driving collaboration with ford otosan." `https://velodynelidar.com/press-release/autonomous-driving-collaboration-with-ford-otosan/`. Accessed: 2021-10-27.

[73] "Velodyne product lineup." `https://velodynelidar.com/products/`. Accessed: 2021-10-18.