UMinho | 2022

Luís Cunha

ALFA-Pi: Generic LiDAR Ethernet Interface

Universidade do Minho
Escola de Engenharia

Luís Carlos da Costa e Cunha

ALFA-Pi: Generic LiDAR Ethernet Interface

maio de 2022

Universidade do Minho
Escola de Engenharia

Luís Carlos da Costa e Cunha

ALFA-Pi: Generic LiDAR Ethernet Interface

Dissertação de Mestrado
Engenharia Eletrónica Industrial e Computadores
Sistemas Embebidos e Computadores

Trabalho efetuado sob a orientação de
Professor Doutor João Monteiro
Professor Doutor Tiago Gomes

maio de 2022

Agradecimentos

Esta dissertação é a representação do fim de toda uma jornada académica e do início de um novo capítulo. Como tal, gostaria de agradecer às pessoas que sempre me apoiaram e prestaram um pouco do seu tempo, experiência e paciência, tornando isto possível.

Aos meus orientadores, Doutor Tiago Gomes e Mestre Ricardo Roriz, um sincero obrigado por todo o conhecimento, disponibilidade e conselhos transmitidos, mas acima de tudo pela confiança depositada em mim para a realização deste trabalho.

Queria também agradecer aos meus amigos de curso, com um achega especial para as pessoas com quem convivo e partilho experiências diariamente – André Campos, João Sousa, Pedro Sousa, Samuel Pereira e Mestre Diogo Costa. Em geral, um obrigado a todos que fizeram parte desta jornada muito importante para mim.

Por fim, um profundo obrigado aos meus pais, irmão e a um dos meus pilares, Cidália Pereira. Vocês fizeram com que nesta jornada nunca me faltasse apoio, motivação e amor. Estiveram lá a comemorar nos melhores momentos e a servir de suporte nos piores. Obrigado pela companhia e carinho que deram a este ”Sr. Engenheiro”.

STATEMENT OF INTEGRITY

Abstract

ALFA-Pi: Generic LiDAR Ethernet Interface

In recent years, the great growth of the automotive industry combined with rapid technological advances in areas such as electronics, has enabled the development of more robust and safer vehicles. This growth also allowed for a paradigm shift towards autonomous driving. An autonomous driving system typically refers to a system capable of moving with little or no human intervention. These systems require highly reliable perception features to navigate the environment, being Light Detection And Ranging (LiDAR) sensors a key instrument in perceiving the vehicles' vicinity. However, for a reliable operation, current perception systems require LiDAR sensors to deliver high-resolution point clouds at a high frame rate, resulting in millions of data points per second and consequently high-speed data handling requirements.

This dissertation proposes ALFA-Pi, a hardware solution to handle high-speed data output of automotive LiDAR sensors that send data through an Ethernet interface. By using dedicated hardware accelerators to interface and decode LiDAR data, ALFA-Pi is able to handle the high-throughput of multiple LiDAR sensors, simultaneously. The proposed system also provides a common output, independent of sensor type or manufacturer, allowing for higher levels of abstraction for the data processing units. Moreover, the ALFA-Pi features various run-time configuration capabilities, accessible by a Robot Operating System (ROS) package.

The system passed through a rigorous validation process, which compared the software-based LiDAR setup (native ROS driver; native Ethernet port; Linux network stack; no acceleration support) with the proposed hardware-based solution (hardware Ethernet port; tweaked network stack/interface; ALFA-Pi hardware modules). The results reveal that ALFA-Pi features better performance metrics (packet decoding time) than the native solutions, demonstrating the performance capabilities of fully specialized hardware systems, and that the ALFA-Pi solution features low hardware resource costs and great scalability.

Keywords: Autonomous vehicles, LiDAR, FPGA, Data representation, Point cloud.

Resumo

ALFA-Pi: Interface Ethernet Genérica para sensores LiDAR

Nos últimos anos, o grande crescimento da indústria automóvel, aliado aos rápidos avanços tecnológicos em áreas como a eletrónica, permitiu o desenvolvimento de automóveis mais robustos e seguros. Este crescimento permitiu ainda uma mudança de paradigma no que diz respeito à condução autónoma. Um sistema de condução autónoma normalmente refere-se a um sistema capaz de se mover com pouca ou nenhuma intervenção humana. Para isso, um sistema deste tipo necessita de sensores que sejam capazes de avaliar o ambiente em torno do veículo, de forma a poder tomar decisões que não ponham em risco os seus ocupantes ou o próprio veículo. Geralmente, estes sistemas sensoriais incluem câmaras, sistemas *Radio Detection And Ranging* (RADAR), e mais recentemente dispositivos *Light Detection And Ranging* (LiDAR), sendo estes últimos considerados elementos-chave para o futuro da condução autónoma. No entanto, a necessidade de grandes resoluções e altos *frame rates* que estes sistemas apresentam, faz com que sensores LiDAR modernos produzam milhões de pontos por segundo, o que representa um problema para as unidades de tratamento de dados.

Esta dissertação propõe ALFA-Pi, uma solução baseada em tecnologias de aceleração para o tratamento de dados de sensores LiDAR que usam uma interface Ethernet. Usando aceleradores em hardware, o ALFA-Pi é capaz de receber e descodificar os dados de múltiplos sensores LiDAR, simultaneamente. Adicionalmente, o sistema usa um formato de dados genérico para representar os dados descodificados dos sensores, independentemente do seu tipo (2D ou 3D). Esta abordagem, permite que aplicações de alto nível que usem os dados produzidos por ALFA-Pi tenham níveis mais altos de abstração, simplificando o seu desenvolvimento e implementação. Finalmente, usando uma interface *Robot Operating System* (ROS), o sistema oferece várias configurações em tempo de execução, dando mais flexibilidade ao ALFA-Pi.

O sistema desenvolvido foi submetido a um rigoroso processo de validação, que o avaliou em termos de precisão dos dados produzidos, performance e custos de hardware. O sistema foi ainda submetido a uma avaliação final que o comparou com a solução oferecida pelos fabricantes (driver nativo do ROS; porta Ethernet nativa; pilha de rede do Linux; sem capacidades de aceleração). Todos estes testes revelaram que o ALFA-Pi apresenta uma melhor performance que a solução nativa, enquanto oferece uma boa escalabilidade e baixo uso de recursos de hardware.

Palavras-chave: Condução autónoma, LiDAR, FPGA, Representação de dados, Point cloud.

Contents

List of Figures

List of Tables

Glossary

ADAS	Advanced Driver-Assistance Systems
ALFA	Advanced LiDAR Framework for Automotive
AMBA	Advanced Microcontroller Bus Architecture
AMCW	Amplitude Modulated Continuous Wave
AR	Angular Resolution
ASCII	American Standard Code for Information Interchange
AXI	Advanced eXtensible Interface
BRAM	Block RAM
CAN	Controller Area Network
CMOS	Complementary Metal–Oxide–Semiconductor
CORDIC	COordinate Rotation DIgital Computer
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
DMA	Direct Memory Access
ECU	Electronic Control Unit
EIS	Ethernet Interface System
FF	Flip-Flop
FIFO	First In First Out
FMC	FPGA Mezzanine Card

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

FMCW Frequency Modulated Continuous Wave

FoV Field of View

FPGA Field-Programmable Gate Array

FPS Frames Per Second

GEM Gigabit Ethernet Controller

GPS Global Positioning System

IP Intellectual Property

IPv4 Internet Protocol version 4

LiDAR Light Detection And Ranging

LUT Look-Up Table

LUTRAM Look-Up Table Random Access Memory

MAC Medium Access Control

MCU MicroController Unit

MDIO Management Data Input/Output

MEMS MicroElectroMechanical Scanner

MII Media Independent Interface

MIO Multiplexed Input/Output

NMEA National Marine Electronics Association

OPA Optical Phased Arrays

OS Operating System

OSI Open Systems Interconnection

PHY Physical Layer Device

Pi Platform interface

PL Programmable Logic

PPS	Packets Per Second
PS	Processing System
RADAR	RAdio Detection And Ranging
RD	Response Delimiters
RGMII	Reduced Gigabit Media Independent Interface
RMII	Reduced Media Independent Interface
ROS	Robot Operating System
RT	Request Terminator
SAE	American Society of Automotive Engineers
SCIP	Sensor Communication Interface Protocol
SLAM	Simultaneous Localization And Mapping
SoC	System on Chip
SPI	Serial Peripheral Interface
TCP	Transmission Control Protocol
ToF	Time of Flight
UDP	User Datagram Protocol

1. Introduction

Nowadays, and just more than a decade after the first self-driving car winning the DARPA Challenge [1], the interest in developing and deploying fully autonomous vehicles is at a great pace [2]. According to optimistic projections, by 2030, autonomous vehicles will be sufficiently reliable, affordable, and widely spread on our public roads, replacing most human driving tasks [3]. Nonetheless, most vehicles on our roads today are still manually controlled, and to make them fully autonomous, they still need to go through different levels of driving automation, e.g., the six levels of automation defined by the American Society of Automotive Engineers (SAE) [4]. While in levels 0, 1, and 2, the driver is required to actively monitor the driving activities, with levels 3, 4, and 5 the automated vehicle should be able to autonomously navigate the environment. Most of the biggest players in the automotive industry already include level-2 capabilities into their vehicles through Advanced Driver-Assistance Systems (ADAS), endowing them with the ability to steer, accelerate, and brake automatically under the driver's active supervision [5]. However, levels 3, 4, and 5 require reliable multi-sensor perception systems to monitor the surrounding environment, which include RAdio Detection And Ranging (RADAR) devices, cameras, and Light Detection And Ranging (LiDAR) sensors [6, 7]. When combined, they can detect the distance and speed of nearby objects, as well as recreate a 3D map of the surroundings [8, 9].

Despite the adoption of LiDAR sensors in the automotive sector being relatively new, they are assumed as a key technology for the future of autonomous driving due to their ability to recreate the surroundings through a high-resolution 3D point cloud representation in real-time [10, 11]. Additionally, LiDAR sensors work with active illumination, which provides good performance ratios up to hundreds of meters, even in low-light environments. Hence, these sensors expand the applications in which driver assistance systems can operate while filling existing gaps between RADAR and camera-based technologies. However, for a reliable operation, modern perception systems require LiDAR sensors to provide high-resolution point clouds at a high frame rate, resulting in millions of data points (up to several gigabit per second of raw data). Tackling such output can be quite challenging [12], since current autonomous vehicle implementations can simultaneously use several LiDAR sensors within a single vehicle [13]. Hence, storage and transmission of data generated by LiDAR sensors are some of the most challenging aspects of their deployment [14].

1.1 Main Goal

This dissertation aims at developing ALFA-Platform interface (Pi), an agnostic and standard interface capable of driving automotive LiDAR sensors that output data via an Ethernet interface. Following a hardware/software co-design, the ALFA-Pi solution not only tackles the high data rates of LiDAR sensors, but also the lack of industry-standard protocols by proposing a generic output format that supports both 2D and 3D sensors, offering higher levels of abstraction for data processing modules present in multi-sensor systems. Therefore, to successfully develop the ALFA-Pi, different requirements were identified:

1. The system must be able to drive multiple high throughput LiDAR sensors simultaneously, which may result in high-speed constraints. These constraints motivate the use of hardware acceleration solutions such as Field-Programmable Gate Array (FPGA) technology, which can provide parallelism and enhanced performance.

2. ALFA-Pi must support and implement a standard output format that can abstract any data processing module, facilitating the integration of the proposed solution with systems that use LiDAR data from multiple different sensors.

3. The hardware must be provided as an Intellectual Property (IP) Core that encapsulates the entire system. Such IP module must be completely customizable through software or at the hardware level, allowing for an increased system flexibility.

4. The developed system must include a Robot Operating System (ROS) interface for system configuration and control, promoting interoperability and easier integration with high-level applications. Furthermore, most of native sensor drivers are ROS-enabled, which makes this feature almost mandatory.

To validate the proposed solution, the evaluation of the ALFA-Pi includes a complete comparison between the native (software-based) and the custom (software/hardware) approach developed throughout this work. Since the system provides acceleration capabilities, it is expected that the developed approach presents better performance than the native solution while offering the generic output interface and a user-friendly customization interface through ROS.

1.2 Document Structure

The remaining of this document is structured as follows: Chapter 2 introduces and explores the main concepts needed for the development of this work, starting by exposing the evolution of LiDAR technology, its working principle and internal components, followed by the LiDAR technology's impacts and challenges in automotive applications and the future of autonomous driving. It ends by discussing and reviewing implementations that accelerate the decoding and/or the reception of LiDAR data. Chapter 3 discusses the platform options and their operation, followed by an overview of the supported LiDAR sensors. Chapter 4 presents the design of the proposed system, starting with an overview of ALFA-Pi, followed by an explanation of the protocols used by the supported sensors. Next, Chapter 5 includes a detailed explanation of the system's implementation. Chapter 6 describes the system's hardware requirements, provides an evaluation of ALFA-Pi, and compares it to the respective native solutions. Finally, Chapter 7 concludes this work and proposes future improvements.

2. Background and State of the Art

This Chapter starts by addressing the main concepts of LiDAR technology for automotive applications. While Section 2.1.1 describes the operation principle of a LiDAR sensor, Section 2.1.2 shows the different architectures and imaging techniques used to collect data from the real world. Next, Section 2.2 exposes the main trends of LiDAR technology in the automotive field followed by the inherent challenges of using this emerging technology in different driving-assistance and autonomous applications. Finally, Section 2.4 reviews and discusses implementations similar to the work presented in this dissertation.

2.1 Automotive LiDAR Technology

A LiDAR sensor works by measuring distances to objects through the round-trip time of light pulses. The utilization of this technique is not new and dates back to before the laser's invention [15], where in 1930, Synge proposed a method to measure the density of the upper atmosphere, which 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 [16]. Later in 1935, this method was improved by using a pulse width modulated signal, allowing easier detection of the transmitted signal and better noise filtering [17].

Only introduced as a range measuring technique in 1953 [18], this technology started to develop quickly with the invention of the laser in 1960 [19]. Still, in 1960, Hughes Aircraft Company, which was specialized in electronic defense systems, created an internal competition to design a laser-based range finder for military use, demonstrating early the capabilities of this technology [20]. About a decade later, all of the basic LiDAR techniques had been suggested and tested [15]. Since 1976, with the publishing of the first textbook on LiDAR [21], improvements have been tied to optical and electronic progress, especially laser technology. This connection is assigned to the high requirements imposed by some LiDAR techniques. These requirements range from laser technology, e.g., laser power, and beam shape, to computer systems that can process large amounts of data [15], raising the need for custom-tailored hardware/software solutions.

Over the years, LiDAR technology has been established as a key instrument in the fields of topography [22], robotics [23], and others [24]. However, its desirable characteristics to describe the surrounding environment have pushed it into new fields, such as the automotive field [5]. The utilization of LiDAR technology in the automotive field started around 2004 with the first edition of the DARPA Grand Challenge.

Within this competition, every team had to build a fully autonomous vehicle capable of navigating rural and urban environments. In the first edition, none of the participants were able to complete the race. Nonetheless, the vehicle that went furthest provided a great benchmark for autonomous driving research, exposing some flaws of camera-based systems, and demonstrating the requirement of real-time obstacle avoidance systems that incorporate LiDAR sensors [25]. Consequently, in the second edition five vehicles completed the challenge. The winner that year was Stanley, the first vehicle to achieve 244 km in seven hours containing one camera, one RADAR, and five LiDAR sensors [1]. These challenges represented a great step towards commercial autonomous driving features, attracting big players to this area. Today various solutions in the market are based on the use of LiDAR sensors, such as the Google's (and Waymo's) approach to these types of systems, which is mainly based on LiDAR [26]. In conclusion, autonomous vehicles are soon becoming a reality, and LiDAR sensors are viewed as a key component.

2.1.1 Basic Concepts

A LiDAR sensor is composed of two main components, the transmitter and the receiver. The transmitter is the unit responsible for emitting the light pulses with wavelengths ranging between 250 to 1600 nanometers. However, the two wavelengths for automotive LiDAR currently in use are 905 nanometers and 1550 nanometers [5]. By its turn, the receiver generally includes a telescope to collect the photons, an optical analyzer capable of filtering specific wavelengths, and a data acquisition module that calculates the elapsed time.

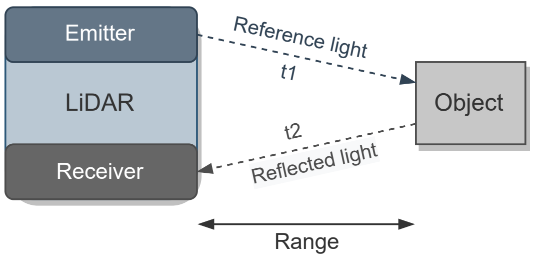


Figure 2.1: LiDAR's operation principle.

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) [27]. As depicted in Figure 2.1, the transmitter sends a light pulse, and then collects the reflected light. The distance, R, is calculated based on equation 2.1, where τ represents the ToF between the emission of the pulse and the reception of the corresponding reflection, and c is the speed of light in the medium.

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

Multiple measurement principles allow the calculation of the ToF. Some of the more prominent techniques are the pulsed ToF [28], Amplitude Modulated Continuous Wave (AMCW) [29], and Frequency Modulated Continuous Wave (FMCW) [10]. Simple architectures like the pulsed ToF often provide cheaper implementations on the cost of accuracy and immunity to noise. On the other hand, more robust implementations like FMCW and AMCW usually require more complex circuitry and electronic systems. Despite the existence of several techniques to calculate the ToF, standard metrics like range, Field of View (FoV), angular resolution, and frame rate are commonly used to characterize a LiDAR sensor [30].

Range

The range can be seen as the minimum and maximum distances where the sensor can detect an object, i.e., detect the backscattered light pulses. These values are usually measured in a controlled environment by pointing the laser to targets with 80% of diffuse reflectivity at different distances. 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 [29].

Field of View and Angular Resolution

FoV is defined as the angle that can be covered by the sensor, i.e., the angle in which the light pulses are emitted and received. In a 3D LiDAR sensor, the FoV is specified with horizontal and vertical angles (Figure 2.2), while for a 2D sensor only the horizontal angle is needed. Several sensor systems can already provide an horizontal FoV of 360 degrees, which is possible by employing a rotator-based LiDAR sensor or combining multiple LiDAR sensors with a smaller FoV [5, 29]. Another important aspect is the lateral or Angular Resolution (AR), 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 [10].

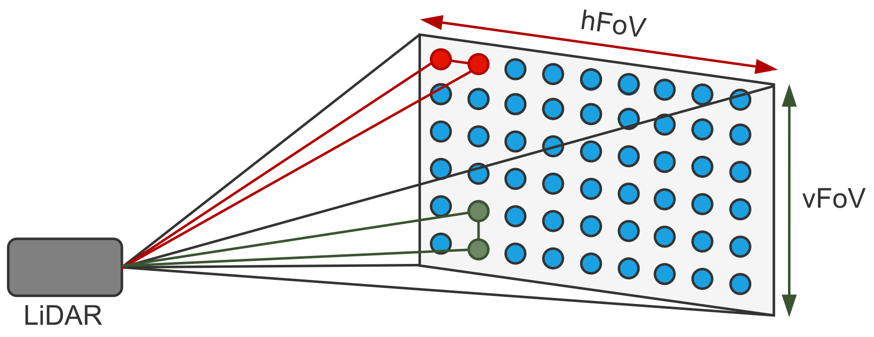


Figure 2.2: Field of view and angular resolution representation.

Frame Rate

The frame rate of a LiDAR sensor is defined as the measurement of how quickly the data from a full rotation appears within a second. High frame-rates are essential, 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 [31]. 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, reaching up to a few hundred Hz [32].

Beam Divergence

In LiDAR 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 [22], as Figure 2.3 illustrates.

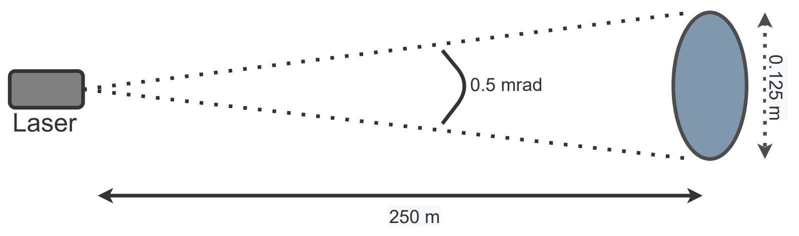


Figure 2.3: Beam divergence illustration.

Although this can be seen as an undesirable effect, as the same amount of energy is dispersed over a larger area, it 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 the sensor is capable of receiving. Nonetheless, this effect is only noticeable at very high ranges.

2.1.2 Imaging Techniques

LiDAR technology can accurately represent the surrounding environment by creating an image with multiple points, also known as point cloud. Hence, different imaging techniques can be used to capture all of these points in an FoV. These techniques are currently divided into scanners, and detector arrays, also known as Flash LiDAR [29]. Scanning-based LiDAR sensors use beam steering components to sweep various angular positions of the FoV, and can be categorized 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 but are cheaper to produce and offer higher frame rates. Flash LiDARs illuminate the entire FoV and use an array of receiving elements, each capturing a different angular section, to create a point cloud of the surroundings [33]. Table 2.1 summarizes the various techniques’ characteristics alongside their main advantage and main limitation [31].

Table 2.1: Brief summary of the imaging techniques.

	Mechanical Scanners	**MEMS**	**OPAs**	**Flash**
Working Principle	Rotating lasers, prisms or mirrors	MEMS micromirrors	Phased array of emitters	Illumination of the entire FoV
Main Advantage	Long range, 360° of HFoV	Compact, lightweight, low power consumption	High frame rate, On-chip solutions possible	High frame rate, no moving parts
Main Limitation	Moving elements, bulky, expensive	Power management, limited range	Long range not commercially available	Limited range

Mechanical Scanners

LiDAR systems based on this technique usually have a rotating mirror or prism, giving this type of sensor a 360 degree view with relative ease. The mechanical actuator in this type of sensor rotates around a single axis to obtain one dimension. The second dimension is usually obtained by adding sources and detector pairs with some angular difference between them [34]. As Figure 2.4 exemplifies, Velodyne VLP-16 creates a 30 degrees VFoV by powering 16 individual pairs at different vertical angles [35]. Those pairs, also known as channels, are rotated horizontally, creating the typical 360 degree HFoV.

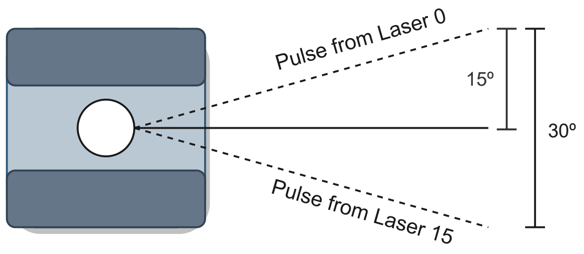


Figure 2.4: Representation of Velodyne VLP-16 side view.

This type of sensor provides the advantage of a reasonably efficient and straightforward optical arrangement, 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 [31]. Although they are very efficient in long-range applications, the setup presents numerous disadvantages, including limited scanning speeds, poor resistance to shock and vibration, large power consumption, bulky design, and price. Nonetheless, it is the most mature imaging technique, making it the primary choice for autonomous research and development [30].

Microelectromechanical Scanners

MicroElectroMechanical Scanner (MEMS) are an emerging solid-state solution since there are no rotating mechanical parts in their setup. Instead, tiny mirrors with a few mm in diameter are tilted at various angles when a stimulus is applied [31]. This stimulus depends on the actuation technologies, and the required performance usually determines which technology is used [36]. Despite offering a smaller FoV than mechanical scanners, there are already several designs that fuse the data of multiple MEMS scanners [37, 38], mitigating this problem.

This type of sensor presents poor resistance to shocks, and the power limitations that arise to avoid mirror damage limit the maximum range. Regardless, due to the lightweight, compactness, and low power consumption, MEMS-based scanners are receiving increasing interest from the automotive world [36], making them 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 MEMS and mechanical scanners are based on beam steering techniques, OPA-based scanners use beam shaping to achieve the same effects. These sensors 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 [39], enabling the control of the shape and orientation of the wave-front, as demonstrated in Figure 2.5 [5]. Because 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 [31], 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 [40].

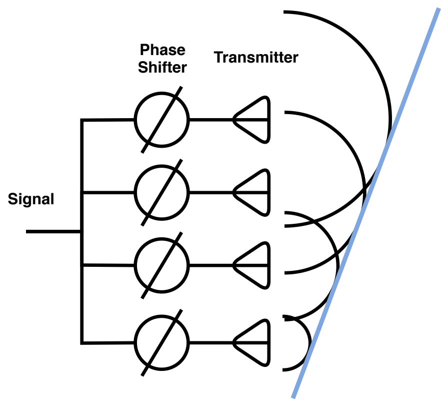


Figure 2.5: Optical phased array principle.

Flash-based sensors

Flash LiDAR are considered full solid-state systems as they have no moving parts, being their operation comparable to standard cameras. Unlike scanners, flash LiDAR sensors use a laser or array of lasers to illuminate the entire FoV and a detector array to capture the reflected signals [31]. As a consequence, the array's density highly influences the angular resolution, making this solution expensive for large FoV [41]. The use of a flashing technique makes the emitter need higher power lasers to illuminate the entire FoV, also increasing the power requirements [10]. In addition, eye-safety regulations limit the power levels, decreasing the maximum detection range to 50-100 meters. Nevertheless, 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 they only have static parts, the potential to create miniaturized systems exists, which can be a significant advantage [33, 42]. Therefore, since these sensors are very straightforward in terms of mechanical complexity, they present a good solution for low to medium-range applications.

2.1.3 Sensor 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. 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 interfacing a LiDAR sensor. Furthermore, connecting a LiDAR sensor to the processing system requires a high bandwidth interface, which increases the challenges of connecting and interfacing high-resolution sensors. Nowadays, most LiDAR sensors for automotive use Ethernet and Controller Area Network (CAN) interfaces.

CAN

CAN is a field bus-level communication network for exchanging short real-time messages [43]. Currently, it is used in most automotive applications and is supported by several manufacturers of integrated circuits and subsystems [44]. It features built-in error detection, robustness, and extensive flexibility, and its best use case is for the exchange of small control messages [45]. However, its low bandwidth makes it only suitable for sensors with very limited FoV or low angular resolution, limiting its utilization in the automotive field [46]. Nonetheless, it is still the primary interface used by automotive and industrial solutions, including some 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) [47]. Since its invention, the Ethernet interface has been improved to support higher bit rates (+100Gbps), more nodes, and longer link distances [48]. Its high bandwidth and versatility make the Ethernet interface the main interface of current LiDAR solutions in the market. For interoperability purposes, LiDAR sensors output their data using the User Datagram Protocol (UDP) or Transmission Control Protocol (TCP), providing, in some cases, an HTTP-based interface for the configuration and management operations, as table 2.2 depicts.

Table 2.2: Interface comparison between different types of sensors.

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

2.2 LiDAR Applications

Currently, autonomous driving capabilities are only possible using multi-sensor data fusion since different variables need to be monitored simultaneously [49]. The three main 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. Consequently, 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 [7].

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. Since traffic signs have high reflectivity values, LiDAR sensors can detect them fairly easily, making them useful in traffic sign detection. Both in Weng et al. [50] and Gargoum et al. [51], this property is explored as the point cloud is filtered by intensity values and then clustered to detect traffic signs. However, the most promissory designs for automotive are the ones that use sensor fusion, allowing the detection systems to achieve high accuracy and performance both in detection and recognition [52, 53].

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 focus on detecting the road boundaries and ground plane [54]. Recent works take advantage of the high reflectivity values road marks present to detect them efficiently [55, 56]. However, weak road maintenance and weather conditions that conceal the road surface, e.g., heavy rains, may cause the system to yield weak predictions. As a solution, Lee et al. [9] proposed a system that combines LiDAR and camera data. This solution presented higher stability, a higher level of redundancy, and better results when faced with adverse conditions.

Obstacle Detection

An autonomous system must be able to identify and detect static and moving objects around the car. Furthermore, detecting their speed, direction, and position helps keep the safety of both passengers and the vehicle. Since this is a data-demanding task, systems with a single sensor struggle to give feasible results [57]. In Gohring et al. [8], an algorithm that combines RADAR and LiDAR data is proposed. This method increases detection speeds and precision when compared to a system that only uses LiDAR. More recently, Lee et al. [58] presented a solution with camera and LiDAR data fusion that achieves higher success rates while meeting real-time constrains. It uses LiDAR for quick obstacle detection/tracking and a convolutional neural network-based detection algorithm specialized for image processing for object classification.

2.3 LiDAR Challenges

Although the usage of LiDAR sensors in the automotive industry is relatively new, they are expected to be an important technology for fully autonomous vehicles. However, there are still several challenges being tackled by industry and academic research. Due to their working principle, the sparse 3D point clouds produced by an automotive LiDAR sensor can be corrupted by a variety of noise sources, such as light interference [10], adverse weather conditions [59, 60], among others [61, 62, 63], limiting LiDAR technology's scope of application. In addition, current LiDAR systems can generate millions of data points per second because of the need for high-resolution point clouds at high frame rates. For this reason, storage and transmission of data generated by LiDAR sensors are some of the most challenging aspects of their deployment [14]. Two types of solutions can be introduced to address this challenge: (1) the use of compression algorithms to reduce the strain on the communication interfaces [14, 64]; and (2) the design of faster interfaces to handle data transmission and decoding [65, 66]. The following Section outlines some solutions currently proposed to help mitigate this problem.

Data Compression

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 [67]. However, these techniques are mainly indicated for point cloud storage, making their use advantageous for airborne geographical mapping and applications without real-time constraints [64].

In Caillet et al. [14] 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 efficiency, as only an AND operation is needed for the masking, and the possibility of using any compression algorithm without losing the claimed accuracy, make this method a viable approach for real-time systems.

Despite presenting good results in reducing transmission stress, compression tasks may heavily penalize real-time applications that rely on the sensor output [5]. Additionally, for the best performance possible, the compression needs to occur as close to the LiDAR as possible, ideally inside the same package. Current LiDAR solutions do not present this feature natively, which makes the validation of these systems harder.

Sensor Interfaces

The necessity for high-bandwidth transmission solutions has pushed the development of new communication technologies in the automotive field. In addition, current automotive LiDAR sensors use Ethernet as the main output interface because of the high bandwidth requirements. Ethernet has become a recognized network technology for in-car communications, for adoption in specific domains, like multimedia/infotainment and ADASs [68], as current automotive network typologies cannot provide enough bandwidth for these applications. Another advantage is the standardization of the complex communication introduced by the various network technologies that support the different automotive domains, e.g., powertrain, chassis [69, 70]. 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 operate in, such as extreme temperatures and high electromagnetic radiation, forcing standard Ethernet networking solutions to be adapted. [71]. Another problem is the hard real-time constraints that automotive applications impose. Standard Ethernet networking solutions are non-deterministic because of the possibility for collisions, eliminating the time-critical predictability in possible time-sensitive data. However, several standards and configurations that address this issue have already been created and are under development [68].

Currently, multiple IEEE standards define the Automotive Ethernet, which supports various speeds and can 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. Lastly, when it comes to the remaining networking stack, the Internet Protocol is seen as the possible network layer for use in Automotive Ethernet networks [72]. The main advantages of the Internet Protocol are its maturity and widespread use, which helps to reduce complexity and provides the flexibility of using established transport layer protocols [73].

2.4 State of the Art on LiDAR Interface Acceleration

As demonstrated in the previous Section, Ethernet is becoming highly recognized as a reliable communication interface for in-car communications. Furthermore, most current LiDAR Commercial Off-The-Shelf (COTS) sensors use an Ethernet interface as their main output. Therefore, and since compression may penalize the system in real-time applications, this dissertation aims to study and provide an FPGA-based approach for driving automotive LiDAR sensors that are Ethernet interface enabled. Also, it must support both 2D and 3D sensors since both types can be used inside the same car for different applications. However, despite the widespread use of hardware-based data decoding techniques and coordinate conversion systems in other domains, to the best of the author's knowledge, there are only a few implementations on LiDAR-based sensor systems. This may be due to the fact that this technology is still relatively new, and multi-sensor perception systems based on LiDAR sensors are slowly becoming to appear. Nonetheless, to better highlight the existent work concerning the acceleration of decoding and/or receiving of LiDAR data, the following Section alongside Table 2.3 exposes these works.

Table 2.3: Current LiDAR packet decoding and data reconstruction approaches.

Contribution	Sensors	Operation	Approach
Fan et al. [65] (2018)	VLP-16	Decoding and reconstruction	Software
Okunsky et al. [74] (2018)	VLP-16	Decoding and reconstruction	Software
Fan et al. [75] (2019)	VLP-16	Decoding and reconstruction	SoC
Sun et al. [66] (2019)	SICK LMS111	Packet reading	FPGA
Fan et al. [76] (2022)	HDL-64	Decoding and reconstruction	Software

2.4.1 Data Reading 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. When deployed on an Operating System (OS), this software stack can introduce unwanted delays in the transfer of data, reducing the interface's bandwidth. Several implementations of these stacks ranging from specialized chips [77] to custom IP Cores [78, 79] exist and could be a reliable solution to eliminate the need for the stack to be present in an OS. As a solution, Sun et al. [66] proposed a system that uses a specialized chip that implements a TCP/Internet Protocol version 4 (IPv4) stack, the W5500, to accelerate the reading of LiDAR data. The chip supports various transport layer protocols and the 10BaseT/100BaseTX Ethernet standards. The communication with the FPGA is done via Serial Peripheral Interface (SPI) for both the configuration and transfer of data. The supported sensor is the SICK LMS111, but any sensor can be integrated since this system does not implement any data decoding. Nevertheless, extra engineering effort would have to be done to interface LiDAR sensors without requiring a software layer.

2.4.2 Data Decoding Acceleration

The problems with high-speed decoding and reconstruction of LiDAR data are quickly becoming more relevant as multi-sensor systems start to appear. Zheng et al. propose a system design for a data packet decoding System on Chip (SoC) capable of processing data packets from a VLP-16 functioning in dual-return mode [80]. The system uses a module that parses and decodes the data, feeding it to a lookup-table COordinate Rotation DIgital Computer (CORDIC) system capable of transforming the data into the cartesian coordinates system. The system's main goal was for it to be designed using the TSMC 0.18 um process and to operate at 100 MHz clock speed since a digital chip would ease the implementation of the system into a perception system. In 2019, Fan et al. implemented the SoC proposed in the previous work and improved its area requirements by implementing a more efficient CORDIC design capable of computing the value of trigonometric functions in only seven rotations independently of the number of input digits [75]. Since the final system was implemented in a SoC, reducing the area occupied by the lookup table approach used before was mandatory. The final SoC presents good performance metrics in decoding and reconstructing RAW data from VLP-16 sensors, needing only 0.012 ms to decode an entire packet and 0.912 ms to decode a full frame at the chip's maximum frequency, 100 Mhz.

2.4.3 Software-Based Approaches

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 advantageous in special-purpose applications [81]. In 2018, Fan et al. proposed a system architecture that can decode, reconstruct and segment LiDAR RAW data from a VLP-16 LiDAR sensor [65]. The system can process the RAW data of the sensor in both functioning modes, transform the decoded data into the cartesian coordinate system, and perform static object segmentation with a euclidean clustering method. After applying these methods, OpenGL was used for point cloud image construction and display. Despite presenting good results in terms of RAW data decoding and object segmentation, further testing is necessary since the tests were only conducted with the sensor running at 10 Hz and no performance values were assessed. Still in that year, Okunsky et al. also devised an algorithm with all the necessary steps to decode data from a Velodyne VLP-16 in dual return mode and reconstruct it into the cartesian coordinate system [74]. Although the software results were promising concerning data correctness, further development and testing in real-time data are needed. Finally, Fan et al. designed and implemented an algorithm to decode and reconstruct LiDAR RAW data from a HDL-64 sensor [76]. The system was tested with various data captures of the sensor working at 10 Hz in varying environments. It achieves good performance metrics taking an average of 7.678 ms to process an entire frame containing 348 packets.

3. Platform, Tools and Sensors

This Chapter addresses the platforms, tools, and sensors necessary for the concretization of this work. It starts by describing the Advanced LiDAR Framework for Automotive (ALFA) platform in which this work is inserted, followed by the tools used. Lastly, it ends with an explanation regarding the sensors supported by this work.

3.1 Advanced LiDAR Framework for Automotive

ALFA is an open-source framework for automotive that aims to offer a multitude of helpful features for the validation and development of LiDAR-based solutions for automotive. These features include:

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

ALFA features a modular design, where its blocks are independently configurable, expandable, and removable, boosting the functionality and flexibility of the ALFA framework. These blocks can be divided into two types: Units and Extensions. The core components that provide resource management, control, and interface are defined as ALFA Units and together form the ALFA Core. Developers can conceive algorithms for specific domains on top of the ALFA Core, creating new extensions to ALFA. By eliminating the hassle of developing an entire system to validate an algorithm or part of it, ALFA speeds up the development time of testing and designing new LiDAR-based algorithms. For instance, ALFA currently supports a weather denoising extension for removing noise from point clouds corrupted by adverse weather conditions. In this extension, several state-of-art algorithms are supported both in their native implementations and FPGA-accelerated versions since ALFA supports a software/hardware co-design. One of ALFA's main goals is creating an abstraction layer between extensions and platform-specific elements like processing units or even LiDAR sensor type or manufacturer. Thus, as Figure 3.1 exposes, the sensor interface proposed in this work is incorporated as one of the central units of ALFA Core. Although this dissertation work is easily integrable into other platforms, the primary purpose is for the suggested implementation to be a part of the ALFA Core. Since the core is inserted in ALFA's hardware layer, the system suggested by this dissertation is even more helpful, as previously discussed.

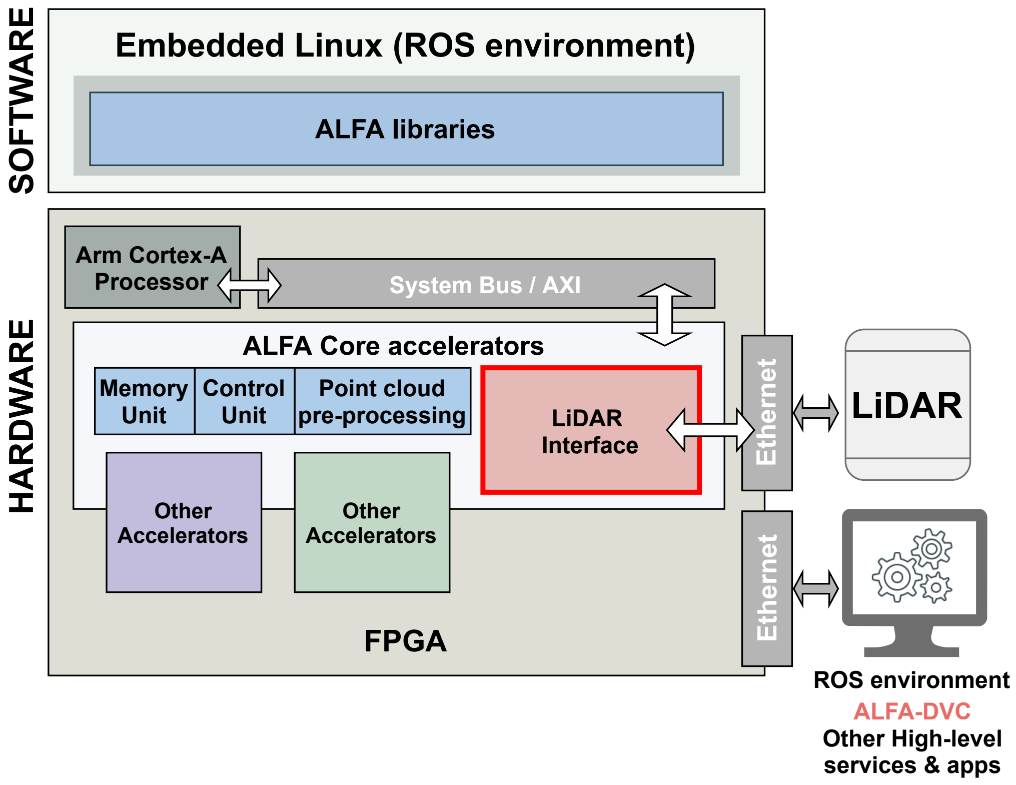


Figure 3.1: ALFA architecture block diagram.

3.2 Hardware Platform

FPGA technology enables the development of custom hardware tailored for a specific purpose. Due to its features such as reprogrammability, it provides fast prototyping at reduced costs, which, in the context of this work, can be used to develop and deploy hardware accelerators. Additionally, using a MicroController Unit (MCU) allows the deployment of higher-level applications that can control and configure the hardware layer in real-time, boosting integrability and debugging. Thus, the chosen platform is the Zynq UltraScale+ EV, which merges the two technologies and is tailored for embedded vision applications such as ADAS, machine vision, and others. While the MCU allows deployment of higher-level applications that control and configure the hardware layer, the FPGA allows dedicated hardware acceleration. The following Section exposes the protocols and hardware platforms used in this work.

3.2.1 AMBA Advanced eXtensible Interface

The Advanced Microcontroller Bus Architecture (AMBA) is an open-standard on-chip interconnect specification for connecting and managing functional blocks in SoC designs. It defines various communication protocols that make functional blocks communicate with each other [82]. One of the AMBA standards is the Advanced eXtensible Interface (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. At the time of writing, the latest AXI version is the AXI5 specification. However, since the IPs provided by the development tools are only AXI4-compliant, the version of the AXI protocol used was the AXI4.

The AXI protocol defines three different types of interfaces: AXI-Full, AXI-Lite [82], and AXI-Stream [83]. The AXI-Full and AXI-Lite interfaces are considered memory-mapped interfaces, where the read and write operations are issued on a specific address. As Figure 3.2 depicts, both interfaces have individual channels for read and write operations, each with independent control signals and address/data buses. Despite their similar structure, the AXI-Lite interface is more limited than the AXI-Full interface since it only allows bursts limited to one data transfer (which technically means no burst support), and all the accesses use the entire width of the data bus. The AXI-Stream interface is considered a point-to-point interface, where data is transfered from one component to another with no specific address constraints, making it ideal for streaming large data buffers without implementing an address control system. However, the AXI-Stream interface only allows write operations from the master to the slave, unlike the other interfaces.

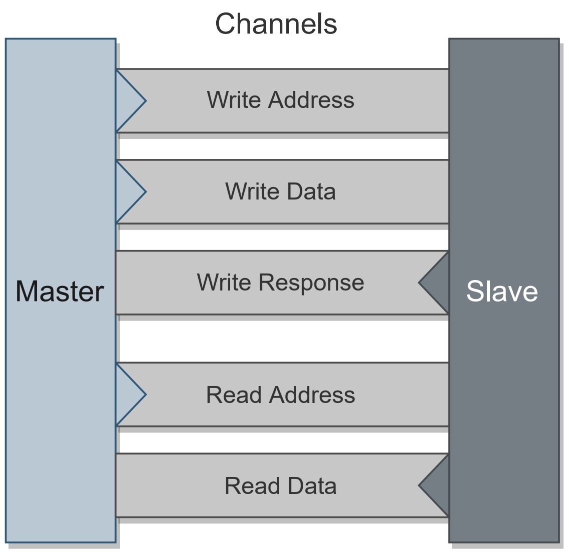


Figure 3.2: Memory-mapped interfaces channel overview.

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 occurs and when it occurs. Another characteristic in common is the 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.

3.2.2 Zynq UltraScale+ EV

The board used for this work is the Zynq UltraScale+ MPSoC ZCU104 [84] which belongs to the Zynq UltraScale+ EV family. As shown in Figure 3.3, the Zynq UltraScale+ EV family features two distinct systems: the Processing System (PS) and the Programmable Logic (PL). The PS includes a quad-core ARM Cortex-A53, a dual-core Cortex-R5, on-chip memory, external memory interfaces, and directly connected I/O peripherals and the PL features configurable logic blocks, direct access to memory-oriented blocks (BRAM, URAM), access to high-speed I/O solutions, and integrated video codecs.

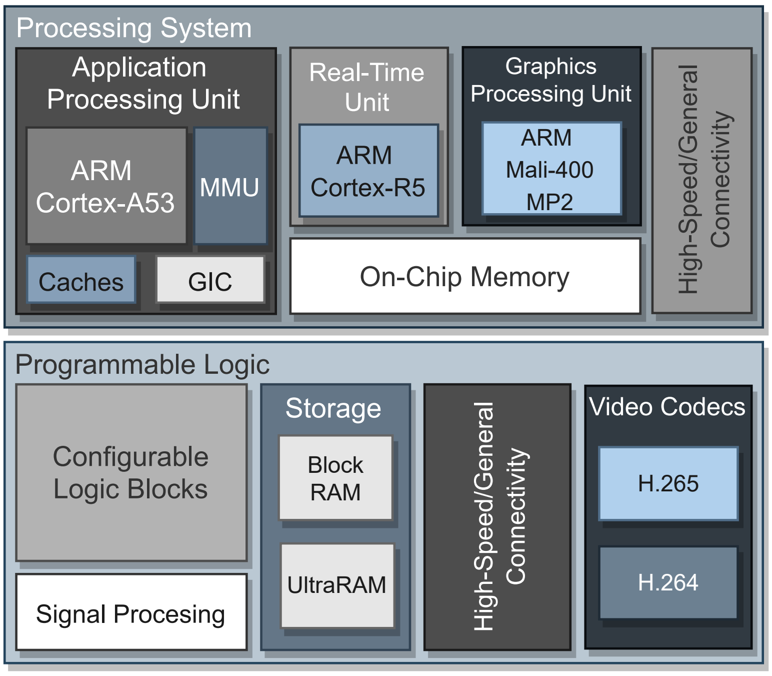


Figure 3.3: Zynq UltraScale+ EV block diagram.

This combination of features extends the functionality of hardware- and software-oriented solutions, making the software/hardware co-design process easier. However, to make this co-design possible, special interfaces between the two systems are needed. Therefore, multiple interfaces based on the AXI protocol are provided for this communication. Finally, using the Zynq UltraScale+ EV allows access to Xilinx's vast library of premade IP cores, accelerating and easing the development process.

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 [85]. It supports a Media Independent Interface (MII), a Reduced Media Independent Interface (RMII), and a Reduced Gigabit Media Independent Interface (RGMII), allowing it to connect to a Physical Layer Device (PHY) chip. Lastly, it features a Management Data Input/Output (MDIO) interface for access to the PHY management registers, useful to configure the PHY to different functioning modes and speeds.

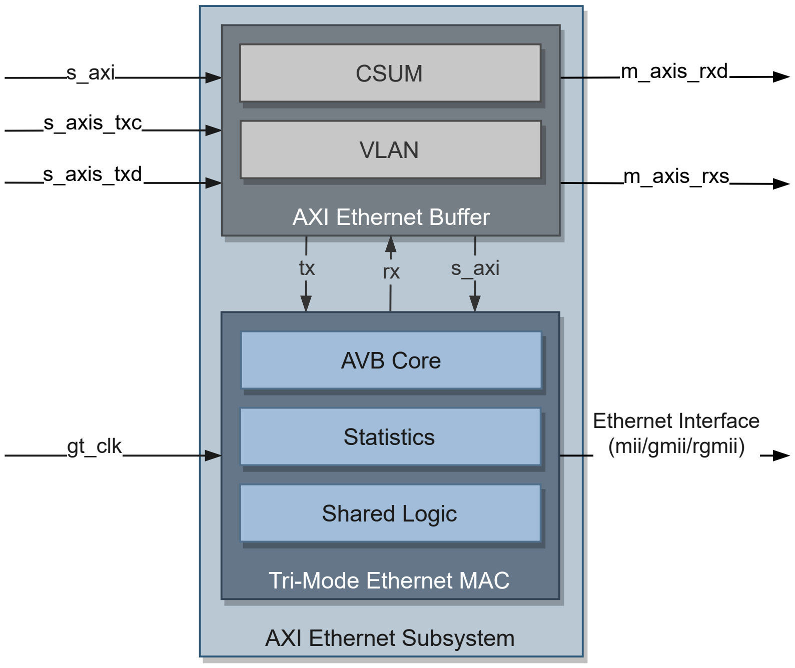


Figure 3.4: AXI Ethernet Subsystem block diagram.

As observed in Figure 3.4, the subsystem comprises two main blocks: a Medium Access Control (MAC) and a AXI Ethernet Buffer that handles data buffering and MAC control [85]. An AXI-Lite interface is provided for simple access to the IP core configuration registers. Also, 32-bit AXI-Stream buses are provided for moving both the transmitted and received Ethernet data. When the Ethernet interface receives data, the data is provided on the *m_axis_rxd* port, meaning that any custom IP core can access it by using the AXI-Stream protocol. The same is valid for sending data to the Ethernet interface; however, the port that must be used is the *s_axis_txd*.

AXI Direct Memory Access

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 and AXI-Full protocols. The AXI DMA provides two operation modes that differ mainly on their features and possible configurations: the Register Mode and the Scatter/Gather modes [86]. The Register Mode contains less features that the Scatter/Gather mode; however, it provides a much simpler configuration interface, making it ideal for simple communication use cases. Both modes feature several AXI-Full and AXI-Stream interfaces, which handle the communication from the memory and custom PL peripherals.

AXI4-Stream Broadcaster

The AXI4-Stream Broadcaster is an IP core developed by Xilinx that offers up to 16 outbound AXI-Stream interfaces from a single AXI-Stream input [87]. In the AXI-Stream protocol, data flows from a single master to one slave. However, some applications may require data broadcast to multiple slave peripherals. Therefore, instead of creating a custom solution to handle this problem, the AXI4-Stream Broadcaster IP core can be used. This IP core also supports the use of a *TUSER* signal, allowing the mapping of data to a specific set of slave interfaces instead of broadcasting it in real-time.

3.2.3 Hardware Ethernet Interface

As stated in Section 2.4.1, there are advantages of processing the Ethernet data as close to the hardware as possible. This not only reduces unnecessary delays caused by data transmission between layers but also bypasses delays introduced by the OS. For these reasons, we chose an expansion board that enables the interception of Ethernet data in the hardware layer.

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 [88]. The dual Ethernet PHYs are the ADIN1300 that feature MII, RMII, and 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, is the ZCU104.

3.3 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. Because ROS is open source, it has a large and engaging community of developers and companies. Its primary 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 that execute a task or set of tasks. These nodes can be connected using the message and service infrastructure provided by ROS, which 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. To be considered part of the network, all the nodes and their message exchanging buses (topics) must be registered in the Master. A node can also have services, which are indicated to activate a specific task or change configurations within a node, contrary to topics, which are better for continuous data exchange. ROS was chosen because both platforms of sensors in use feature a ROS driver. Thus, using ROS within the developed system is an advantage, not only because it is a good communication base for the system, but also because it is a well-known framework with plenty of applications inside the automotive world.

3.4 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, the ROS layer must be added to include ROS capabilities into a Linux distribution. From there, the recipes for ROS packages are included in the project and can be integrated into the final Linux distribution. The layer/recipe based approach makes it easy to make custom-tailored Linux distributions with only the desired packages and functionalities.

3.5 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 LiDAR sensors that Hokuyo produces. Although the use of 2D LiDAR sensors in the automotive world is still evolving [89], the relative price when compared to 3D LiDAR solutions and applicability in applications such as blind spot assistance [5], make it a desirable option [90].

Hokuyo sensors feature a wide range of operation modes together with a relatively high horizontal FoV and high working frequencies. 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 section of the entire sensor’s FoV. For example, if a sensor has an FoV of 250° and 1000 steps, each step represents $\frac{250}{1000} = 0.25^\circ$. Also, some sensors have a dead zone where they can not make readings, despite having steps in those zones. A representation of a sensor FoV can be observed in Figure 3.5. Lastly, the Hokuyo sensors feature multiple operation modes that range from distance only information to information on multiple returns combining distance and intensity data. The sensors supported by this work are all the sensors inside the UST, UTM and URG line-ups, in all their operation modes. However, any other Hokuyo sensor might be supported as long as they feature an Ethernet interface together with the Sensor Communication Interface Protocol (SCIP) 2.0 protocol [91], further explained.

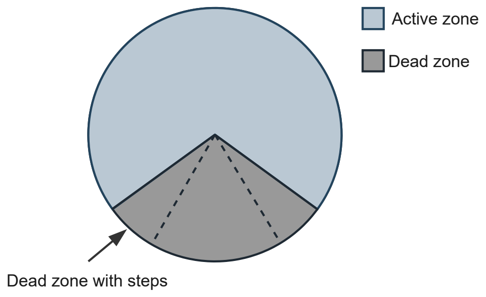


Figure 3.5: Hokuyo's field of view zones.

3.6 Velodyne Lidar

Velodyne Lidar provides powerful LiDAR solutions for autonomous vehicles, driver assistance systems, and several other fields. Velodyne Lidar has a vast list of products as well as hundreds of active customers, making their products a worldwide reference. In addition, there are multiple implementations of autonomous vehicles that use Velodyne sensors with great success [92, 93]. For these reasons, Velodyne Lidar was chosen to showcase the support for 3D LiDAR sensors.

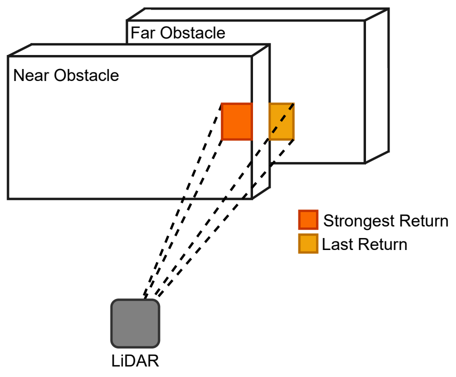


Figure 3.6: Dual return working principle illustration.

Velodyne's line-up of mechanical scanners is based on the ToF principle. All of the sensors are capable of producing 3D point clouds with a 360° horizontal FoV [94], featuring various configuration possibilities, accessible via a web interface hosted by the sensor. In addition, the sensors provide three return modes: Strongest Return, Last Return, and Dual Return. As Figure 3.6 depicts, the strongest return represents the return where the biggest amount of the pulse returned to the receiver, while the last return represents the furthest point the pulse had to travel. The Dual Return mode provides the information on both types of returns at the same time, the strongest and the last. The proposed system currently supports all mechanical-based Velodyne Lidar sensors in all their operation modes, and any sensor that uses the same data packet structure.

4. ALFA-Pi Design

This Chapter addresses the proposed ALFA-Pi architecture, followed by a deep explanation of the protocols used by the supported sensors. Lastly, the generic output format designed for supporting 2D and 3D LiDAR solutions is addressed.

4.1 System Architecture

This dissertation's main goal is to study and provide a generic LiDAR Ethernet-based interface using an FPGA-based approach for interfacing automotive LiDAR sensors. Figure 4.1 depicts the proposed system architecture, ALFA-Pi, responsible for the high-speed decoding and reading of 2D and 3D LiDAR data. The system is composed of several hardware building blocks, providing easy maintainability and configurability features.

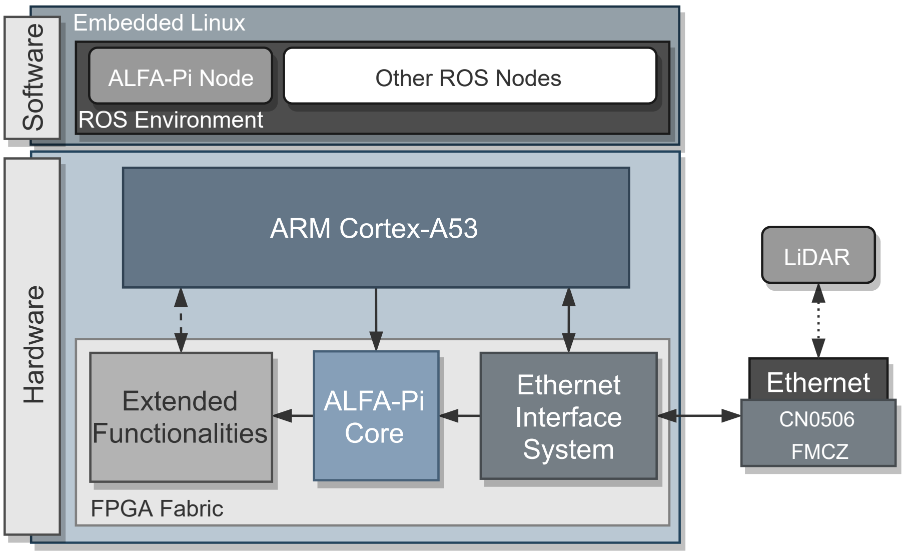


Figure 4.1: ALFA-Pi block diagram.

The system is divided into two main layers, software and hardware. A ROS environment running on top of an embedded Linux operating system defines the software layer. Using a custom ROS package, this layer can configure and control the various ALFA-Pi hardware cores in real-time, expanding the system's flexibility.

In the hardware domain, the system is divided in two main blocks: the Ethernet Interface System (EIS) and the ALFA-Pi Core. The EIS encapsulates all the necessary IP cores required for a fully functional Ethernet interface accessible on the hardware layer. Additionally, it is responsible for providing the received Ethernet data to the ALFA-Pi Core. The ALFA-Pi Core is responsible for processing all the received Ethernet data. There are two major data processing tasks in this core. First, it filters the Ethernet data that does not belong to a supported sensor, and identifies the type of sensor on accepted data. The second task is the decoding, reconstruction, and output of the received data.

4.2 LiDAR Output

As mentioned in the previous Chapter, the manufacturers supported by the ALFA-Pi system are Hokuyo and Velodyne Lidar. Their sensors are very distinct both in features and data-output formats. Therefore, this section exposes and analyzes the data provided by these sensors, which will help in successfully decoding their data. This section exposes and explains these output formats as they are a vital component to the ALFA-Pi's design.

4.2.1 Hokuyo Output

The Hokuyo sensors use the SCIP2.0 protocol, a communication protocol compliant with the SCIP protocol. This protocol does not depend on a particular type of sensor, and it is based on a "command-response" principle. In this type of control, the host unit sends a request, and the sensor answers back according to the requested command. Those commands are expressed by two American Standard Code for Information Interchange (ASCII) characters, that alongside the parameters, define the sensor's behavior. There are two types of commands:

- **Measurement commands** instruct the sensor to execute one or more scans and return the resulting data.
- **Non-Measurement commands** configure the sensor and/or retrieve the sensor's status and settings information.

Also, the SCIP2.0 features a specific set of constructs used to define the type of data present in each message, how data is arranged and encoded inside a message, and error detection. These constructs are explored further in this section.

Message Types

SCIP2.0 protocol defines three types of messages to represent the control sequences between the host system and the sensor. These message types are Request Message, Response Message, and Scan

Response Message, being their fields different depending on the command and stage of the host-sensor communication.

Request Message: A request message defines any message the host system sends to the sensor. As Figure 4.2 depicts, request messages are composed of a command code, command parameters, a user-defined string, and a Request Terminator (RT). The user-defined string is an optional element used for command sequences identification. Finally, the RT can either be a carriage return character, a line feed character, or both, signaling the end of a request message.

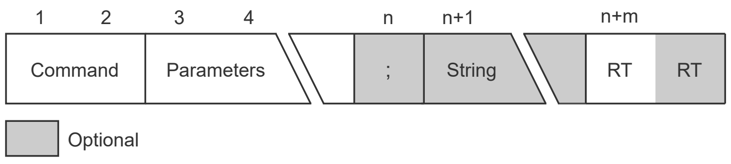


Figure 4.2: Request message structure.

Response Message: This type of message defines the message sent by the sensor if the requested command was a single scan command or a non-measurement command. A single scan command is a measurement command that triggers the sensor to perform a scan, returning the captured data and stopping the operation. As Figure 4.3 depicts, a response message is composed of an echo of its request message, followed by status information and data dependent on the requested command. Finally, the last two bytes are the Response Delimiters (RD) that signal the end of the received message.

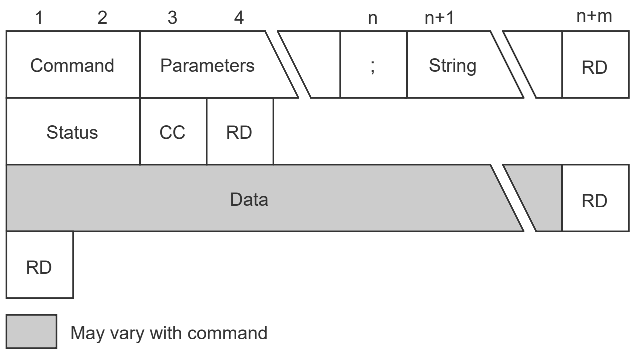


Figure 4.3: Response message structure.

Scan Response Message: A scan response message is the type of message the sensor sends when it receives a continuous scanning command from the host system. A scan response message has the same structure as a response message. However, the echo section corresponding to the request message is partially different. All the continuous scanning commands have a parameter that defines the number of scans the host system wants the sensor to perform. The sensor changes this parameter in a scan response message so the host system can identify which scan the message represents. Also, when a continuous scanning command is requested, the sensor sends a response message without any data before starting the continuous transmission of scan response messages.

Encoding

In SCIP2.0, the numerical representation of the scanning data is encoded to reduce the stress on the communication interface. The encoding is performed by 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 ASCII representation of the integer value. If, after encoding, the result has two characters, it is called 2-character encoding, if it has three characters, it is called 3-character encoding, and so on and so forward. Figure 4.4 demonstrates an example of the character encoding with the number 2607.

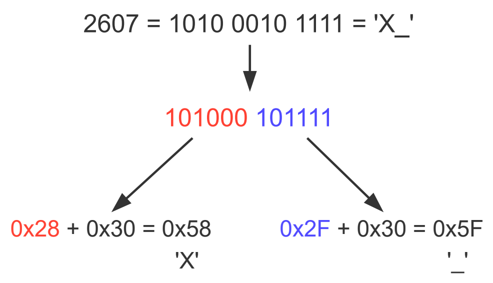


Figure 4.4: 2-character encoding example.

Check Code

Check codes are used in numerous settings to help in error detection that may have occurred during data transmission or storage. SCIP2.0 implements a check code to verify errors during the transmission process of a data block. The algorithm that calculates the check code starts by adding all data inside a determined block. Lastly, it uses the last 6 bits of the previous operation and adds the number 0x30 to it (character encoding). An example of the check code calculation with the string 'ABC' can be observed in Figure 4.5, where the final check code is 0x36.

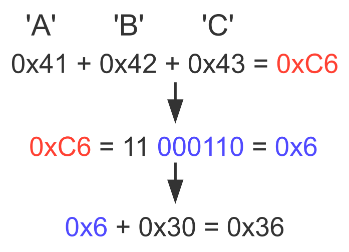


Figure 4.5: Check code calculation example.

Measurement Data

The Hokuyo sensors can operate in a variety of modes that change the type of data they provide. SCIP2.0 defines four different types of scanning data: distance, distance-intensity pair, multi-echo distance, and multi-echo distance-intensity pair. Which data type the sensor provides at any specific moment is command-dependent.

- **Distance:** In this mode, the sensor only provides distance information. Depending on the command, the sensor can either use 18 bits or 12 bits of data to define a point. The 18-bit mode provides bigger ranges at the cost of using 3-character encoding. Contrarily, the 12-bit mode offers a much lower range but only uses 2-character encoding, reducing the network load.
- **Distance-Intensity Pair:** Some sensors have commands that enable them to provide both the distance and the reflected signals' intensity values. The intensity values represent the energy the receiver detected from the reflected signal. These two values form the distance-intensity pair. As Figure 4.6 depicts, the first three bytes correspond to the distance values, and the remaining three correspond to the intensity values. Note that the intensity value is directly proportional to the energy the receiver detects.

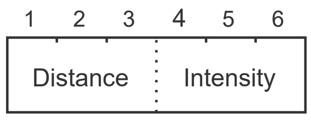
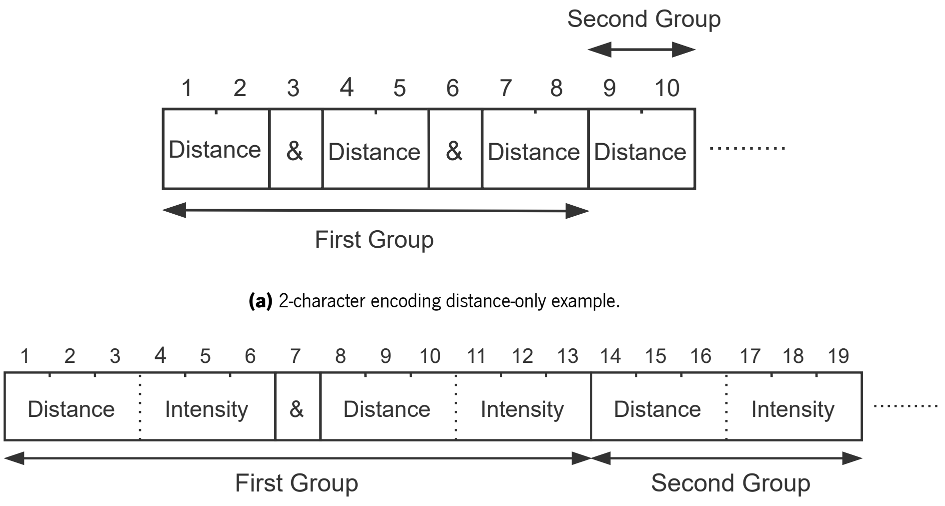


Figure 4.6: Distance-intensity pair structure.



(a) 2-character encoding distance-only example.

(b) Distance-intensity pair example.

Figure 4.7: Multi-echo structure.

- **Multi-echo:** Hokuyo sensors can provide the results for several returns in the same step. SCIP2.0 manages this feature by using the multi-echo structure. For each step, the data of each return comes organized in one of the base structures mentioned above. However, the returns that belong to the same step are separated by a '&' character. Figure 4.7 depicts examples using both base structures.

Message and Block Splitting

SCIP2.0 divides a sensor's message internally into various blocks for easier parsing and error detection. It implements a block splitting method that organizes data into 64 character blocks with individual check codes and response delimiters. However, since the message size is not always a multiple of 64, the last block may not have the complete length, as Figure 4.8 depicts. Additionally, a message can be externally divided into several TCP packets depending on its dimension. The packet splitting does not affect the internal message structure and is only performed to reduce bandwidth consumption since the messages can become long depending on the sensor's FoV and AR.

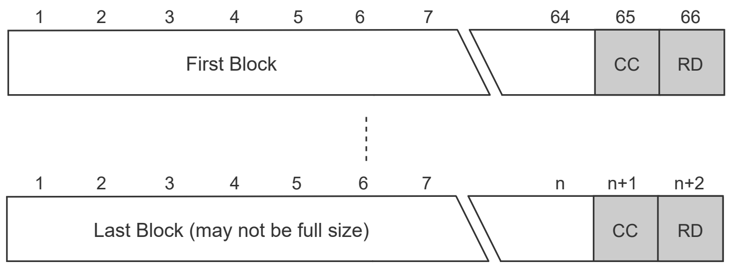


Figure 4.8: Block splitting structure.

4.2.2 Velodyne Lidar Output

Velodyne Lidar sensors use a proprietary data protocol based on the UDP that defines 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. Therefore, these packets do not represent any data the sensor can generate, being only a feature for easier integrability into Simultaneous Localization And Mapping (SLAM) systems. As the position packets are out of the scope of this work, this section will not discuss them.

Contrarily, data packets contain the information the sensor collects from the surrounding environment. Like the protocol presented for the Hokuyo sensors, the protocol used by Velodyne Lidar features a specific data packet structure with various data definitions to better organize data. The following section discusses these structures and data definitions.

Data Definitions

The protocol used by Velodyne Lidar sensors defines various data definitions that help in the decoding and organization tasks. The sensors report the data points using the spherical coordinate system, consisting of a radius (distance), elevation (vertical angle), and azimuth (horizontal angle). Therefore, the data definitions and packet structure are all built around this coordinate system.

- **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 model of the sensor. For example, in the VLP16, a firing sequence corresponds to all 16 lasers being fired. In sensors with higher number of lasers, the lasers are not all fired simultaneously and instead fired in groups. Lastly, a firing sequence is not allowed to span through multiple data packets.

- **Laser Channel:** A laser channel represents a single emitter and detector pair fixed at a specific vertical angle relative to the horizontal plane. Each laser channel has an ID, which can be inferred by its location inside a data packet. Thus, the sensor does not report the elevation angle of a particular data point, reducing the unnecessary data transmission since the vertical angles are already known. Also, in some sensors, the laser channels are horizontally misaligned to reduce cross-talk. Consequently, these channels have a fixed azimuth offset between them, much like their vertical angle.

- **Data Point:** A data point represents the information the sensor collected from a laser channel. Three bytes of data represent a data point. The first two bytes are referent to distance information, and the last byte is the calibrated reflectivity. The distance is an unsigned integer, and its granularity depends on the sensor. For example, the VLP-32C has a granularity of four millimeters, meaning that a value of 2500 represents a distance of 10000 millimeters. In contrast, the calibrated reflectivity is defined 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 a perfect reflection.

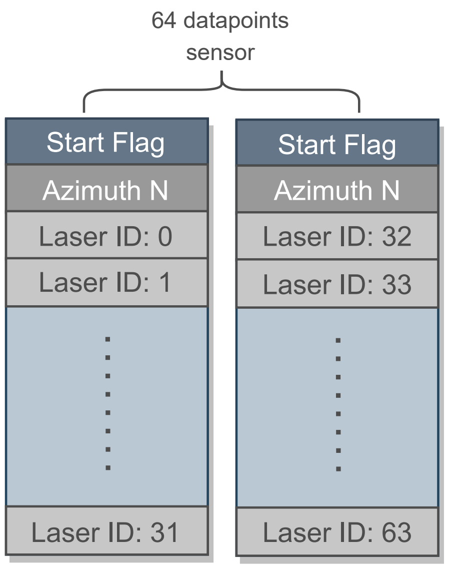
- **Azimuth:** The azimuth is an unsigned integer represented by two bytes at the beginning of each data block. This value represents an angle in hundredths of a degree, allowing a high angular resolution without the need for floating points. This value can range from 0 to 35999, with only one azimuth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

- **Factory Bytes:** The last two bytes of every packet are called the Factory Bytes. The first byte has information about the return mode, and the second indicates the model of the sensor that sent the message. Since different sensors present different features and azimuth/vertical offset values, these values make data packets self-contained. Table 4.1 demonstrates the different sensors' codes and modes.

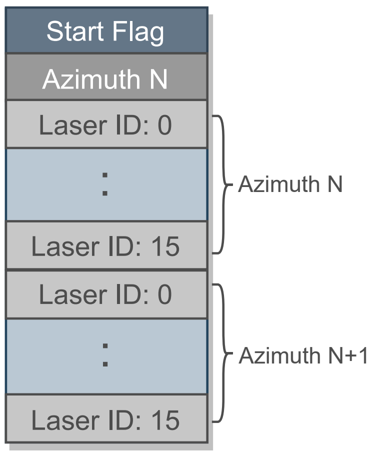
Table 4.1: Velodyne's factory bytes meaning.

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)

- **Data Block:** Each data block is comprised of one hundred bytes of binary data. 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 these blocks comes in a little-endian configuration. When a sensor with more than 32 laser channels is in single return mode, consecutive data blocks hold the same azimuth value. Contrarily, in sensors that feature less than 32 laser channels, multiple azimuth values can be inferred in the same data block. For example, on the Puck LITE (a sensor with 16 laser channels), the last 16 data points inside a data block have an azimuth value equal to the block azimuth plus the angular resolution of the sensor. Figure 4.9 demonstrates examples of these two configurations.



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



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

Figure 4.9: Data Block examples of sensors with more or less than 32 laser channels.

Data Packet Structure

Each data packet comprises 1248 bytes divided into twelve data blocks, four timestamp information bytes, two factory bytes, and an UDP header. There are two possible data packet formats: the single return and the dual return, referent to the sensor's modes. In dual return mode, the sensor provides two returns per laser channel, sending a pair of data blocks for each azimuth, both containing information on the same laser channels but with distinct returns. Figure 4.10 depicts the structure used in this mode, where the odd-numbered blocks include the strongest return and the even-numbered blocks hold the last return. Contrarily, 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, previously demonstrated in Figure 4.9.

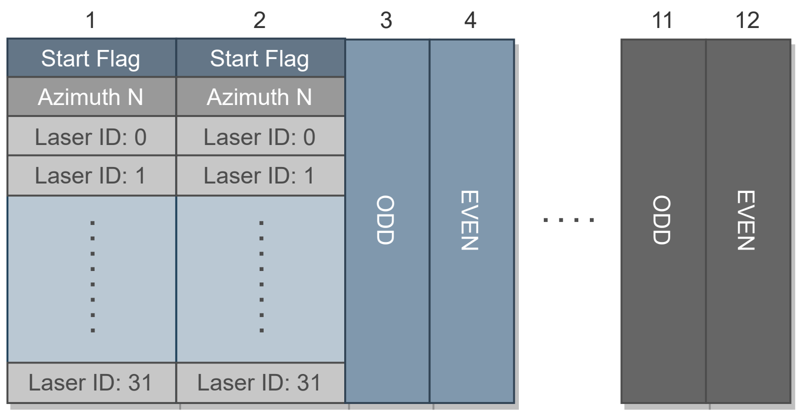


Figure 4.10: Dual return packet structure.

4.3 ALFA-Pi Output Format

As stated, this dissertation aims to produce a generic LiDAR interface capable of driving multiple automotive LiDAR sensors while creating an abstraction layer between the type of sensor used and the data-consuming modules. When it comes to the type of data the sensors provide [35, 95, 96, 97], a trend starts to emerge across the major LiDAR manufacturers. Almost all of them provide their data in a spherical coordinate system. As represented in Figure 4.11, three values are needed to describe a point inside a three-dimensional space using the spherical coordinate system: radius r, elevation ω, and azimuth α. Hence, ALFA-Pi employs the same spherical coordinate system design, following the industry trends. As Figure 4.12 depicts, the data structure starts with the End Flags field, which indicates when a packet has ended and when a frame has ended, followed by 6 bits for sensor identification. Next, a 16-bit azimuth value and a 16-bit elevation angle represent the angle values in the hundredths of a degree, e.g., the value 2007 represents an angle of 20.07°. Furthermore, the format supports two returns for the same data point since it is the most common number of returns supported by LiDAR sensors. Consequently, the format features two distance values and two extra fields, explained further. The two distance values,

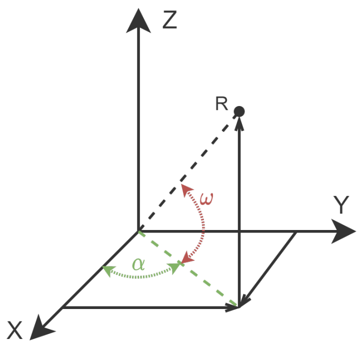


Figure 4.11: Spherical coordinate system.

each with 20 bits, represent the distance detected by the sensor for each return in millimeters. Finally, the extra fields, each with 8 bits, are responsible for transporting extra values the sensor might produce. For example, Velodyne Lidar sensors, alongside distance, also provide reflectivity values for each return. Therefore, each extra field holds these values for each return.

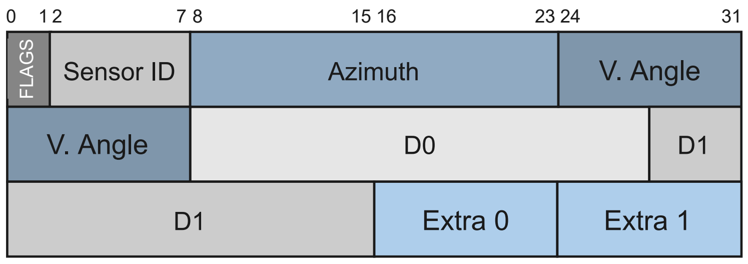


Figure 4.12: Generic output structure.

With this structure, 2D sensors are also supported by ignoring the elevation angle field, effectively removing it from the output. Furthermore, a 20-bit resolution for the distance data allows representing values much higher than what current LiDAR solutions can provide, preparing the system for future LiDAR solutions.

5. ALFA-Pi Implementation

This Chapter explores the implementation of ALFA-Pi, composed of software and hardware modules, presenting the components in each layer. Section 5.1 displays the implementation and design decisions of the hardware layer where the main ALFA-Pi cores can be found, followed by Section 5.2, which exposes the software modules.

5.1 Hardware

As Figure 4.1 demonstrates, the ALFA-Pi core alongside the EIS compose the hardware layer. This Section is divided into two subsections, each referent to these main system modules. The ALFA-Pi Core is responsible for the decoding and pre-processing tasks of the system, receiving its data from the EIS, which handles the connection to the sensor using the EVAL-CN0506-FMCZ expansion board. These modules communicate with each other using an AXI-Stream interface, which handles the high-speed data streaming tasks.

5.1.1 Ethernet Interface System

The EIS allows the reading of sensor data directly from the Ethernet EVAL-CN0506-FMCZ board interface, accelerating the data reading task without the need of the OS intervention, making it directly available to the ALFA-Pi Core. The following subsections explain the design decisions and the IP Cores that combined accomplish the EIS.

5.1.1.1 MAC and PHY Connection

The first task of the EIS is to intercept Ethernet data as early as possible in the network stack. The EIS resorts to several Xilinx IP Cores to implement the PHY and the MAC layers. Figure 5.1 describes the Open Systems Interconnection (OSI) model, which characterizes and standardizes the communication operations of a telecommunication system, independently of its underlying internal structure or technology used. In the proposed solution, the network stack uppermost layers (3 and up), e.g., IPv4, UDP, are controlled by the network stack manager provided by an embedded Linux OS. Although these functionalities could be deployed in the hardware layer, they would not add significant improvements to the packet decoding process. Therefore, by keeping the upper layers of the network stack in the software, the system gets debugging and deployment flexibility. Furthermore, controlling the lower levels in the hardware allows the system to intercept data as soon as possible and bypass any unwanted delay caused by the OS, allowing faster reading speeds.

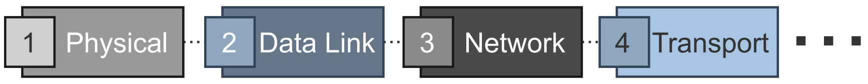


Figure 5.1: Open systems interconnection model.

Regarding the hardware platform, the 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 [84]. The onboard PHY connects to a Gigabit Ethernet Controller (GEM) available in the PS through a Multiplexed Input/Output (MIO) Interface. This interface is the only method provided by this platform to get the Ethernet Data collected through the RJ-45 connector. Thus, it requires transmitting data from the PS to the PL to process it, implying even more delays due to an inefficient data handling. Consequently, the proposed solution uses the EVAL-CN0506-FMCZ board attached to the ZCU104 to make the data fully accessible in the hardware layer without requiring the PS. The EVAL-CN0506-FMCZ provides two PHY that can be directly accessed using an FMC-LPC connector. To control these PHYs, handle Ethernet framing protocols, and perform error detection, the AXI 1G/2.5G Ethernet Subsystem is used. Henceforth, the MAC considered by the system is the one inside the AXI 1G/2.5G Ethernet Subsystem. These components' capabilities and applications were explored in sections 3.2.3 and 3.2.2, respectively.

Figure 5.2 depicts how the MAC connects to the EVAL-CN0506-FMCZ board PHY. Several interfacing options were available to connect them. However, the RGMII interface was chosen since it is the only one capable of supporting data rates of 1Gb/s, necessary for one of the supported sensors. Nonetheless, the RGMII interface requires a transmit clock of 125MHz. Since the EVAL-CN0506-FMCZ already provides one, it is used to clock the MAC block/interface, reducing the hardware cost of implementing one extra clock in FPGA. Alongside this, the MDIO interface of the MAC connects to the MDIO interface of the PHY for communication control.

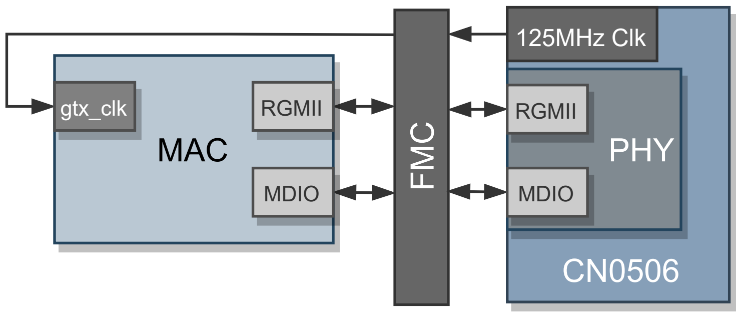


Figure 5.2: EVAL-CN0506-FMCZ and MAC connection diagram.

5.1.1.2 Configuration and Data Transmission

Xilinx provides device drivers to enable the Linux distribution to use the AXI 1G/2.5G Ethernet Subsystem seemingly, easing the implementation process. However, this device driver is dependent on using a DMA engine to receive and send Ethernet data via the AXI 1G/2.5G Ethernet Subsystem. Thus, two approaches are possible: 1) create a "man-in-the-middle" peripheral between the AXI 1G/2.5G Ethernet Subsystem and the DMA, to decode the sniffed data; or 2) broadcast the received data both to an independent IP core and the DMA at the same time.

In the first approach, the custom IP is the one that regulates DMA transfers and needs to provide data buffering. Not only is this much more resource-intensive, but it also introduces higher delays. Therefore, the final system follows the second approach. Figure 5.3 illustrates the EIS Block, where the AXI 1G/2.5G Ethernet Subsystem IP Core makes data available to an AXI4-Stream Broadcaster IP Core, enabling data flow into the ALFA-Pi Core and to the DDR4 memory via the AXI DMA. This way, the data transfer control is assured by the AXI 1G/2.5G Ethernet Subsystem IP Core and the AXI DMA, where the AXI4-Stream Broadcaster IP Core only introduces one clock cycle delay. Finally, data transmission to external devices using the EIS is handled exclusively by the PS using the DMA connected to the data input port of the AXI 1G/2.5G Ethernet Subsystem IP Core. Two AXI-Lite interfaces allow the PS to configure the AXI DMA and the AXI 1G/2.5G Ethernet Subsystem IP Cores, where each has a slave interface that connects to an AXI master interface in the Zynq MPSoC. Therefore, using Xilinx Linux device drivers, the PS can automatically configure these IP cores.

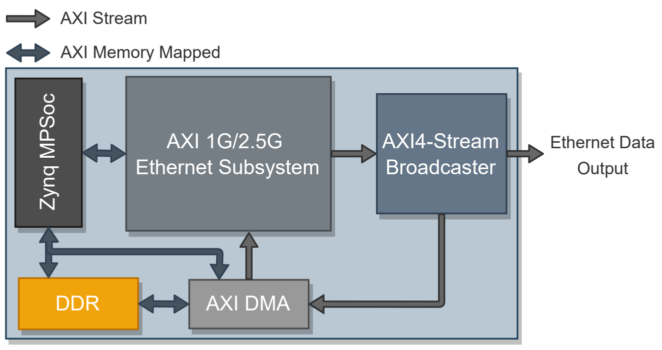


Figure 5.3: Ethernet interface system block diagram.

5.1.2 ALFA-Pi Core

After all the Ethernet data arrives to the PL, it is necessary to perform the filtering, decoding, and data reconstruction steps. The ALFA-Pi Core comprises three modules, as depicted in Figure 5.4, each with a specific set of tasks. Also, the ALFA-Pi Core offers configuration features to save FPGA resources and change the core's functionalities, where each sensor driver can be individually enabled or disabled, saving FPGA fabric resources when only one type of sensor is present. The following sections discuss the ALFA-Pi Core modules' functionalities.

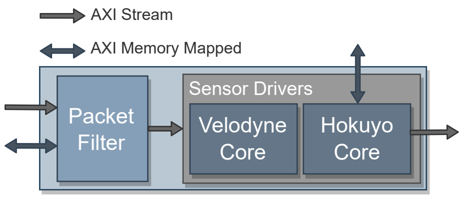


Figure 5.4: ALFA-Pi Core system block diagram.

5.1.2.1 Packet Filter

The Packet Filter module filters Ethernet data that does not belong to the sensors the system is actively decoding and verifies the sensor type to decide the correct sensor driver for decoding. Furthermore, it presents various header filters, e.g., MAC, UDP, allowing the sensor drivers to only focus on decoding tasks. As Figure 5.5 depicts, the AXI Lite Interface, the Sensor Look-Up Table, and the Filter sub-modules form this module.

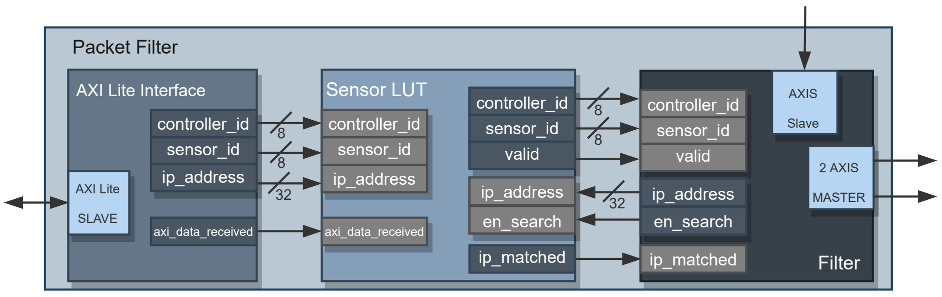


Figure 5.5: Packet Filter sub-modules block diagram.

AXI Lite Interface

One stage of the filter applied to the Ethernet data uses the sensors' IPv4 addresses to filter their messages. Those IPv4 addresses can be changed in real-time by the PS using an AXI-Lite slave interface, making the module more versatile. Two 32-bit registers containing the sensor's ID information and IPv4 address are passed to other modules using the designated output ports as depicted in Figure 5.6. Lastly, this sub-module flags the reception of an AXI write message in the *axi_data_received* port, signaling other modules that a new configuration is available.

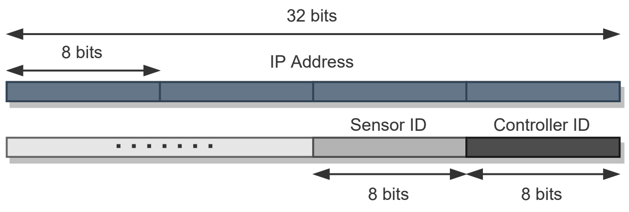


Figure 5.6: AXI-Lite message structure.

Sensor Look-Up Table

One of the objectives of the proposed solution is to support multiple sensors simultaneously. Therefore, storing all sensors' IPv4 addresses and ID information is necessary. The Sensor Look-Up Table sub-module is responsible for this task, storing those two parameters for each configured sensor. Figure 5.7 represents the memory layout of one position inside this sub-module. The following subsections further explain how other modules use the sensor ID and controller ID for correct decoding and data filtering.

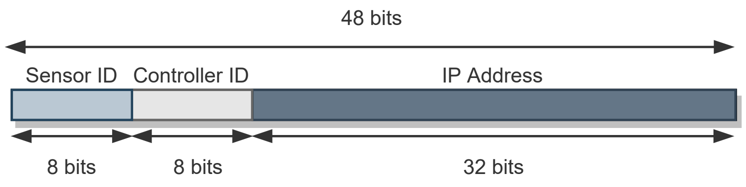


Figure 5.7: Sensor Look-up Table entry layout.

The Sensor Look-Up Table receives the *axi_data_received* trigger from the previously mentioned AXI Lite Interface sub-module and proceeds to store the information in the correct position, using the received sensor ID. However, the Filter sub-module only has access to the sensor's IPv4 address and not its ID. As a result, it is necessary to look up if there is an ID associated with the IPv4 address. As further explained, the Filter block does not require the ID information when it knows the IPv4 address of the packet's source. Therefore, the goal is not to get the ID information for the Filter step but rather to interpret if the IPv4 address of the packet's source is stored or not. For this reason, the Sensor Look-Up Table sub-module performs the detection of the IPv4's validity instantly using combinational logic. Additionally, this sub-module fetches the IDs sequentially, using the method depicted in Figure 5.8. This approach saves hardware resources with a negligible performance penalty, as the Filter can still function until it needs the IDs.

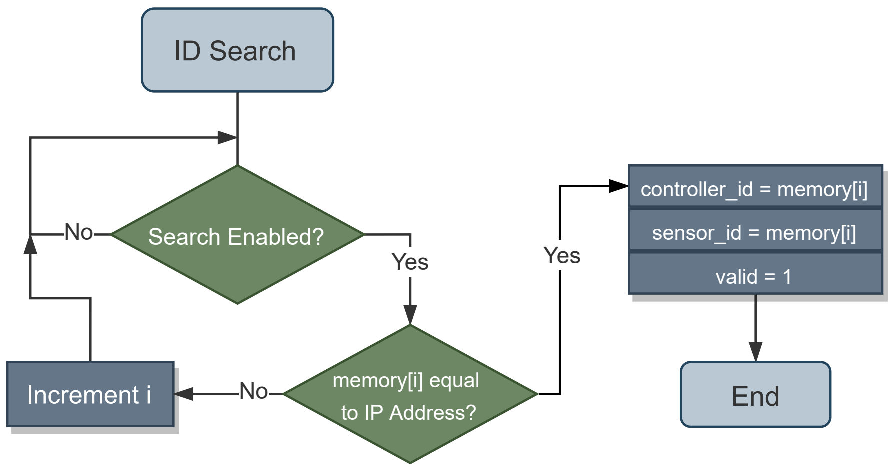


Figure 5.8: IP search flowchart (memory[i] represents the position i inside the Look-Up Table).

Filter

The Filter sub-module is responsible for filtering the received packets and forwarding the accepted data to the correct driver for decoding. Since it handles network stack packets, this sub-module is a multi-layer filter, acting on all the different packet layers. The state machine depicted in Figure 5.9 is responsible for controlling this multi-layer filter.

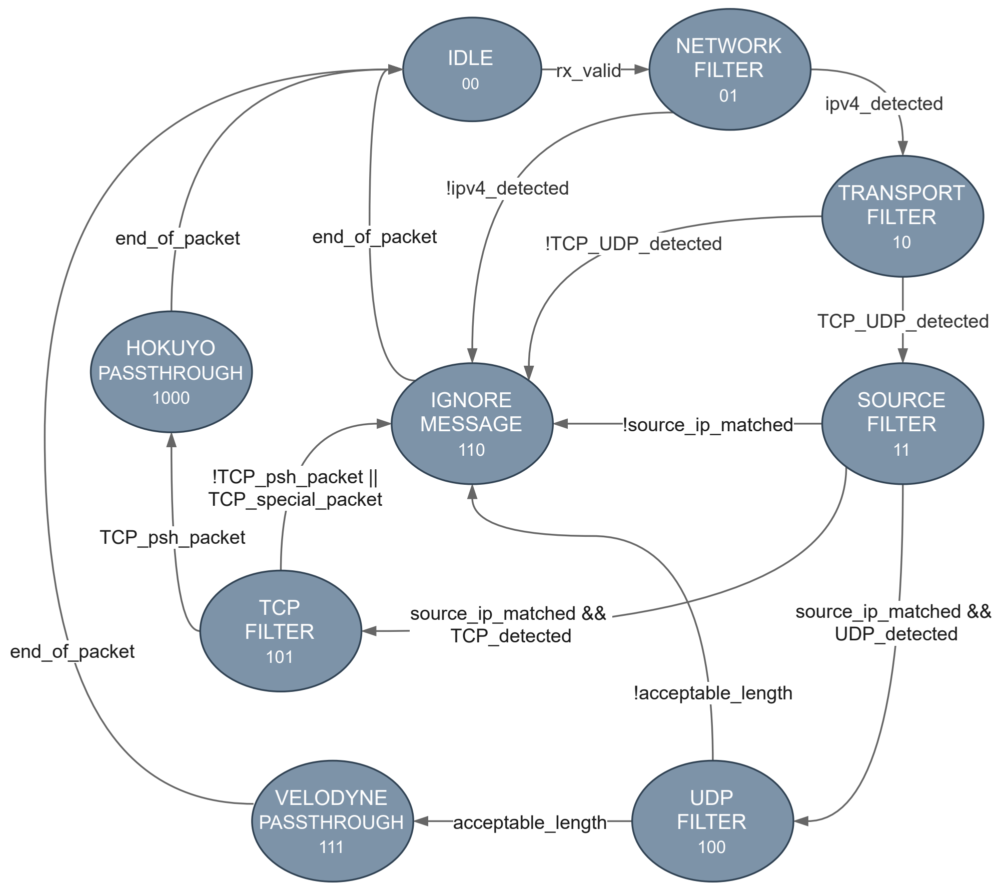


Figure 5.9: Filter state machine.

IDLE: The Filter sub-module communicates with the AXI4-Stream Broadcaster mentioned in Section 5.1.1.2. The IDLE state is not responsible for performing any task, acting as a “waiting” state until an AXI-Stream communication is initiated in the AXI-Stream slave interface present in the Filter sub-module. When communication starts, the state machine passes to the first filtering state, the NETWORK FILTER state.

NETWORK FILTER: This state acts on the MAC header information for packet filtering. The first 12 bytes of the MAC header correspond to the MAC addresses of the packet’s source and destination. Although the MAC addresses can be used for filtering, the system discards them since it uses the IPv4 address instead. Nevertheless, the last two bytes of the MAC header, the EtherType field, define which protocol encapsulates the frame’s payload. The line-up of supported sensors uses IPv4 for this purpose.

Consequently, when in the NETWORK FILTER state, the sub-module detects if these two bytes contain the 0x0800 value corresponding to IPv4. If this value is detected, the state machine proceeds to the TRANSPORT FILTER state; otherwise, the state machine advances to the IGNORE MESSAGE state.

TRANSPORT FILTER: The IPv4 header depicted in Figure 5.10 contains valuable information for the Filter sub-module. The analysis of this header is divided into two states. One of them is the TRANSPORT FILTER state, which determines the protocol used in the transport layer using the Protocol field. In the line-up of supported sensors, the Protocol field defines either the TCP or UDP, used by Hokuyo and Velodyne, respectively. For this reason, when in this state, the sub-module detects if the value inside the Protocol field is 0x06 (TCP) or 0x11 (UDP). If the sub-module detects one of these values, the state machine proceeds to the SOURCE FILTER state. In contrast, if none of the mentioned values is found, the state machine advances to the IGNORE MESSAGE state.

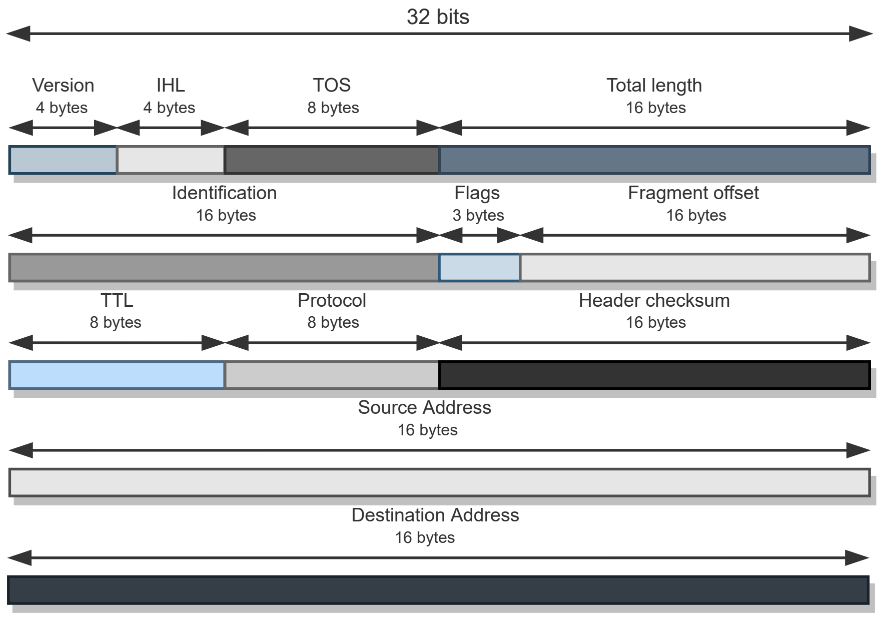


Figure 5.10: IPv4 header structure.

SOURCE FILTER: This state is responsible for controlling the filtering of data packets using the source IPv4 address present inside the IPv4 header. When the sub-module has access to the source IPv4 address, it sends a pulse using the *en_search* output port, activating the Sensor Look-Up Table sub-module mentioned earlier. If on the next clock cycle, the *ip_matched* signal contains the value one, the system sets the *source_ip_matched* flag to one. From here, the state machine has three possible transitions. If the detected transport protocol was UDP and the *source_ip_matched* flag is set to one, the state machine advances to the UDP FILTER state. On the contrary, if the detected protocol was TCP and the *source_ip_matched* flag is set to one, the state machine advances to the TCP FILTER state. Lastly, if the *source_ip_matched* flag has value zero, the state machine proceeds to the IGNORE MESSAGE state.

UDP FILTER: While in this state, the sub-module handles the filtering when the detected transport protocol is UDP. Currently, only the Velodyne supported sensors use UDP. As the Velodyne sensors have all packets with a specific size, the state machine only advances to the VELODYNE PASSTHROUGH state if the packet has the correct dimensions. Conversely, if the length is invalid, the state machine progresses to the IGNORE MESSAGE state.

TCP FILTER: When the received packet uses TCP, this state is responsible for controlling the filtering task. Currently, the supported Hokuyo sensors use TCP as the transport layer protocol. TCP supports several packet formats, and their types are indicated by a field inside the TCP header, denominated Flags, which contains 9 bits, each symbolizing a different packet format. The flags the system uses are SYN, FIN, and PSH. The communicating nodes use the SYN and FIN flags for starting and ending communication. The first packet sent from each node must have the SYN flag set to one, and the last packet must have the FIN flag set to one. When in this state, the system analyzes these two flags. If one of them contains the value one, the state machine advances to the IGNORE MESSAGE state, as these messages do not contain any sensor information. The Hokuyo sensors use the PSH flag to signal that the packet contains sensor data inside. Thus, when the received packet has the PSH flag with value one, the state machine proceeds to the HOKUYO PASSTHROUGH state. Otherwise, the state machine progresses to the IGNORE MESSAGE state.

VELODYNE PASSTHROUGH: When the sub-module reaches this state, it means no more filtering is needed and that a Velodyne sensor sent the received packet. Here, the system waits until the *valid* input port is set to one by the Sensor Look-Up Table sub-module, meaning that the *sensor_id* input port and the *controller_id* input have data correspondent to the current packet. Also, the state machine reaching this state means that all the data that is going to be received until the end of the packet corresponds to the actual payload sent by the sensor. Therefore, this sub-module introduces one byte, the sensor ID, at the beginning of this data stream and passes it through to the Velodyne driver. With this approach, the driver only receives the sensor's payload and sensor ID. Finally, the state machine returns to the IDLE state when the data stream ends.

HOKUYO PASSTHROUGH: This state is very similar to the VELODYNE PASSTHROUGH state mentioned above. When the system reaches this state, it means an Hokuyo sensor has sent the current packet. All of the procedures are similar to the mentioned in the VELODYNE PASSTHROUGH state. However, when in this state, the system also adds the *controller_id* to the beginning of the message alongside the *sensor_id*.

IGNORE MESSAGE: As the name implies, while in this state, the sub-module ignores a packet that was determined invalid by the filter. The state machine stays in this state until the streaming of the packet has ended, passing to the IDLE state.

5.1.2.2 Hokuyo Core

As the name suggests, the Hokuyo core transforms the Ethernet data from Hokuyo sensors into azimuth, distance, and intensity information. Several components compose this core, as Figure 5.11 demonstrates. In order to improve the core throughput, those components feed a data processing pipeline. Since Hokuyo sensors generally devide single scan messages into multiple packets, the core features several buffers inside, each storing multiple TCP/IPv4 packets for each sensor until the system receives a complete scan message. Storing all the packets for a specific sensor together inside the same buffer can result in higher memory utilization, but it is a trade-off for much easier control and faster operation. Finally, these buffers use the AXI-Stream protocol to communicate with other sub-modules, easing the design process.

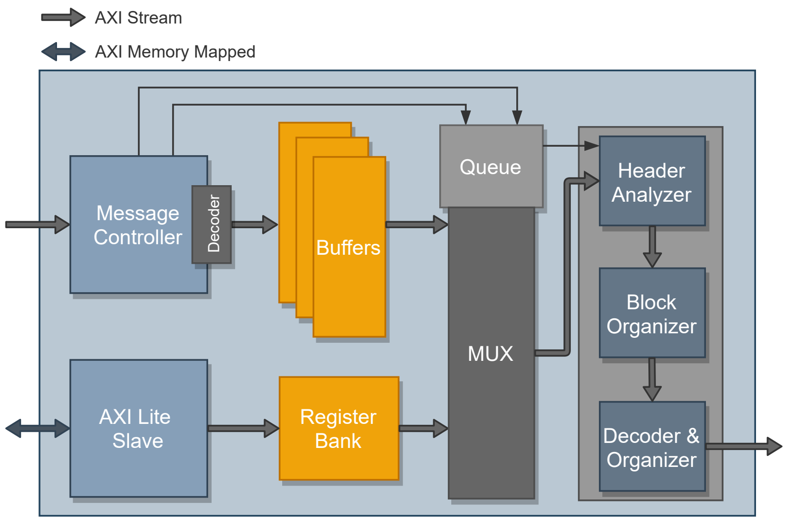
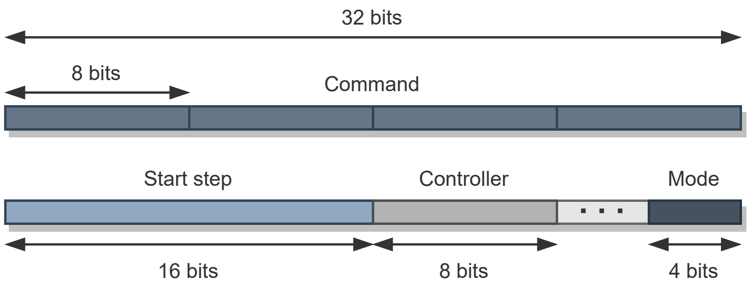


Figure 5.11: Hokuyo Core system overview.

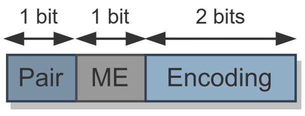
The Hokuyo Core also contains a Register Bank for storing sensor information required for decoding. The data is arranged based on the sensor's controller ID, determining which buffer belongs to which sensor. The ID information is defined by the user in real-time, using the ROS node (further explained).

AXI Lite Interface

The Hokuyo sensors feature multiple operation modes, which affect their data output. Consequently, the Hokuyo Core must be capable of supporting all of these modes. Additionally, the module needs to adapt on-the-fly since the modes can be changed in run-time. For these reasons, the AXI Lite Interface sub-module features a control and customization interface for the Hokuyo Core, allowing the software in the upper layers to customize the hardware. Also, this sub-module is responsible for populating the Register Bank with the sensor's information based on the controller ID received in the message. Figure 5.12a depicts the message layout used for communication between the software and this sub-module. The message contains the first four characters of the user command (used for error detection), the start step, controller ID, and the mode used. In Figure 5.12b, the mode field structure can be observed. The



(a) Control segment structure.



(b) Mode field structure.

Figure 5.12: Control segment.

four bits word contains all the information needed to decode the messages correctly, as follows:

- *Pair:* In one bit, Pair has information about which data comes inside the payload. The Hokuyo sensors can send distance information or distance-intensity information. When this bit has the value one, it means that the sensor is sending distance and intensity, otherwise only distance is being transmitted.
- *ME:* Also in one bit, ME indicates if the sensor is in multi-echo mode, as this affects how data is distributed inside the payload.
- *Encoding:* Lastly, within two bits, Encoding informs about the type of encoding the sensor is using. Depending on the command, the sensor can be using two character encoding or three character encoding.

Message Controller

When the system receives a new packet, the Message Controller sub-module is responsible for selecting the corresponding buffer for storing the new data and notifying when the entire message has been received. The former is achieved by analyzing the controller ID present in the received data. Lastly, as depicted in Figure 5.13, the decoder connects the AXI-Stream master interface of the Message Controller to the correct AXI-Stream slave interface present in one of the buffers. It is important to highlight that the Message Controller act as a passthrough since it does not control any aspect of the AXI-Stream communication, enabling the use of the AXI-Stream control built into the buffers without reimplementing it. Regarding detecting the end of the frame, the module follows the structure defined in SCIP2.0, in which a message is considered finished when two RD characters are found consecutively. Thus, the Message Controller monitors the data until this pattern is detected, flagging a signal on the *end_of_frame* output port.

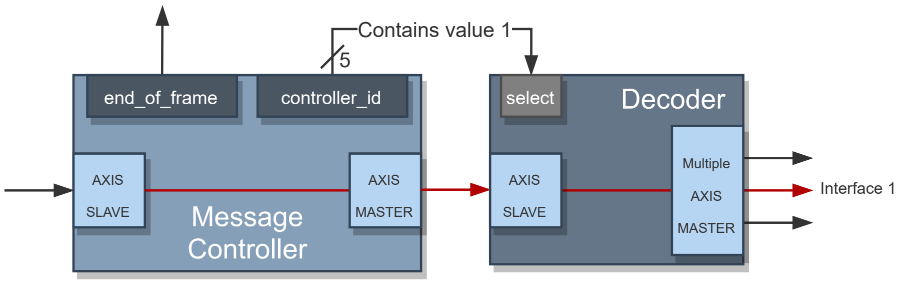


Figure 5.13: Message Controller interface overview.

Queue & Multiplexer

The Queue and Multiplexer sub-modules are responsible for starting the data processing sub-modules and redirecting data into them. While the former stores information about which buffer has received an entire message, the latter is responsible for forwarding the correct data and outputting it into the Header Analyzer. Usually, data processing tasks start after the *end_of_frame* pulse emitted by the Message Controller. However, these triggers can occur while the system is still processing data related to other sensors. In order to guarantee that the system catches all triggers, the Queue sub-module stores the controller ID whenever the pulse appears. Consequently, the Header Analyzer only starts operating when the Queue is not empty by using the *not_empty* output port. As Figure 5.14 depicts, the Queue sub-module connects to the *select* port of the Multiplexer sub-module using the *o_controller_id*, which indicates the value present in the first position of the queue.

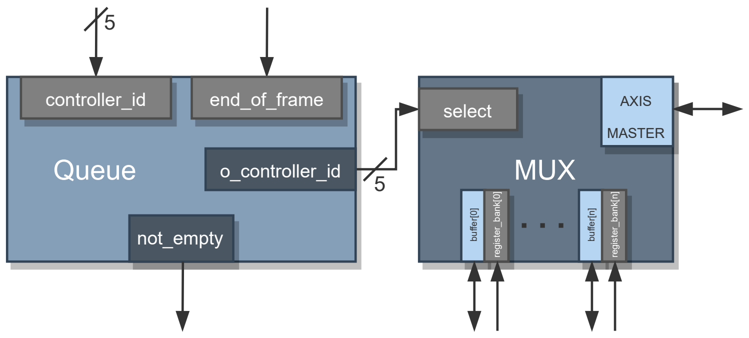


Figure 5.14: Queue and Multiplexer interface.

Header Analyzer

The Header Analyzer is the first component of the Hokuyo Core module that processes data, therefore, it is the first stage in the processing pipeline. It is responsible for analyzing the header of SCIP2.0 messages, detecting if the command received is correct, detecting response messages, and detecting possible errors in the middle of a scan message. In Figure 5.15, the state machine that drives this module can be observed.

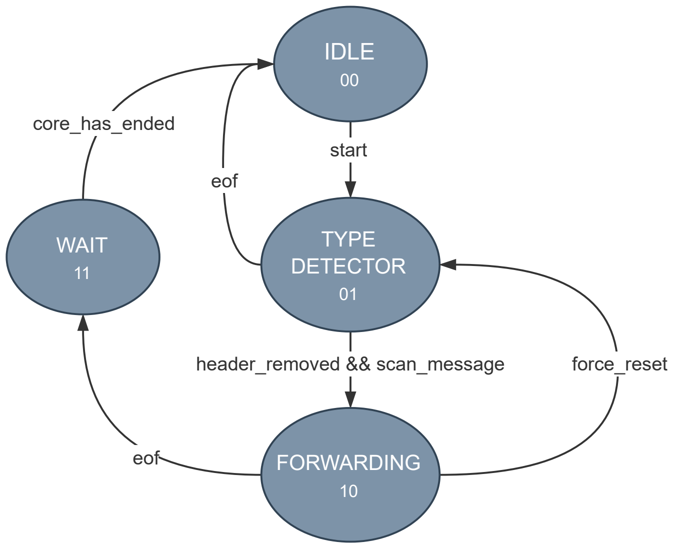


Figure 5.15: Header Analyzer state machine.

IDLE: The Header Analyser sub-module stays in the IDLE state until the Queue sub-module has elements inside. Then, it advances to the TYPE DETECTOR state.

TYPE DETECTOR: While in the TYPE DETECTOR state, this sub-module filters the SCIP2.0 headers and then detects the type of message received. Since the header starts with an echo of the command received by the sensor, the Header Analyzer compares it with the information stored in the Register Bank. This detection makes sure that if the sensor is functioning in a different mode than the one declared in the Register Bank, the system ignores the messages. As mentioned in Section 4.2.1, the sensor sends a response message with no data at the start of a continuous scan. As these messages contain no valuable data, the system detects them by searching for two consecutive RD characters at the end of the header. If this condition is detected, the state machine returns to IDLE since it is a response message with no information. On the other hand, if the system determines the message contains valuable information, the state machine continues to the FORWARDING state.

FORWARDING: In this state, the Header Analyzer receives data from the buffers and forwards it to the Block Organizer. Nonetheless, the data is still being monitored in order to detect possible headers within the message. If new headers are found while in the FORWARDING state, the Header Analyzer forces a pipeline reset, returning to the TYPE DETECTOR state. This mechanism protects the system from misleading data formed by failed transmissions.

WAIT: Since there is a forced delay between the Header Analyzer sub-module and the Decoder & Organizer sub-module, this state acts as a transitioning state to satisfy the data processing pipeline. Therefore, the Header Analyzer stays in this state until the Decoder & Organizer tasks ends.

Block Organizer

Acting as the second stage of the pipeline, this sub-module is responsible for verifying the checksum codes on each data block. As mentioned before, data comes organized in blocks of sixty six bytes divided by a checksum byte, used to detect problems that may have occurred during data transmission, and an RD character. Section 4.2.1 references the mechanism used to calculate the check code in this sub-module.

Decoder & Organizer

After filtering non-important data and verifying the correctness of the remaining one, the pipeline's last sub-module decodes and organizes data into the final generic output structure, already mentioned in 4.3. The sensors' intensity values are placed in the Extra field present in the generic output format. The Decoder & Organizer sub-module is divided into two tasks: decoding data and organizing the output. The decoding part is responsible for receiving data and outputting the decoded form using the method stated in Section 4.2.1. The Register Bank's Encoding field (detailed in Section 4.2.1) is used to determine the decoding type used. The system counts the amount of data it receives and when the threshold is achieved, the sub-module starts the decoding process. Then, the data is arranged onto the correct places of the output buffer.

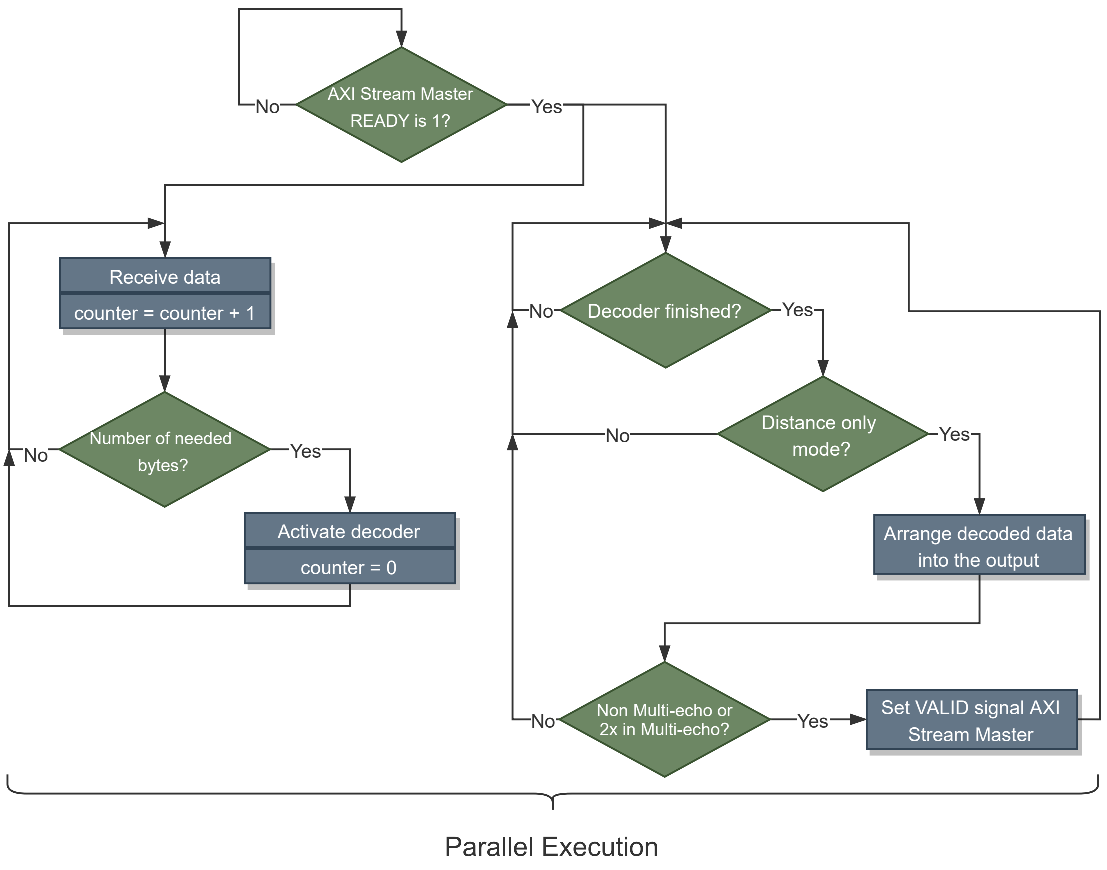


Figure 5.16: Decoder & Organizer flowchart.

As Figure 5.16 depicts, when the sensor is operating in distance only mode, the process mentioned before is performed once per data point. In the event the sensor is in distance-intensity mode, the process happens twice per data point to extract all the necessary data. Finally, when in multi-echo both these operations can be doubled depending on the number of returns per data point and the type of data the sensor is providing. When the sub-module detects the last output word, it activates the *end_frame* portion of the output structure alongside the *packet_ended* flag. Then, the *core_has_ended* flag is set to one, signaling all the previous sub-modules that the system has finished processing the entire sensor's message, and the pipeline resets.

5.1.2.3 Velodyne Core

The Velodyne Core reconstructs the Ethernet data received from Velodyne Lidar sensors into azimuth, vertical angle, distance, and reflectivity information. As Figure 5.17 demonstrates, it comprises several First In First Out (FIFO) sub-modules, which allows the storage of data in a temporal order until needed. The four individual FIFO are used for data separation, allowing parallelization in the later stages of data

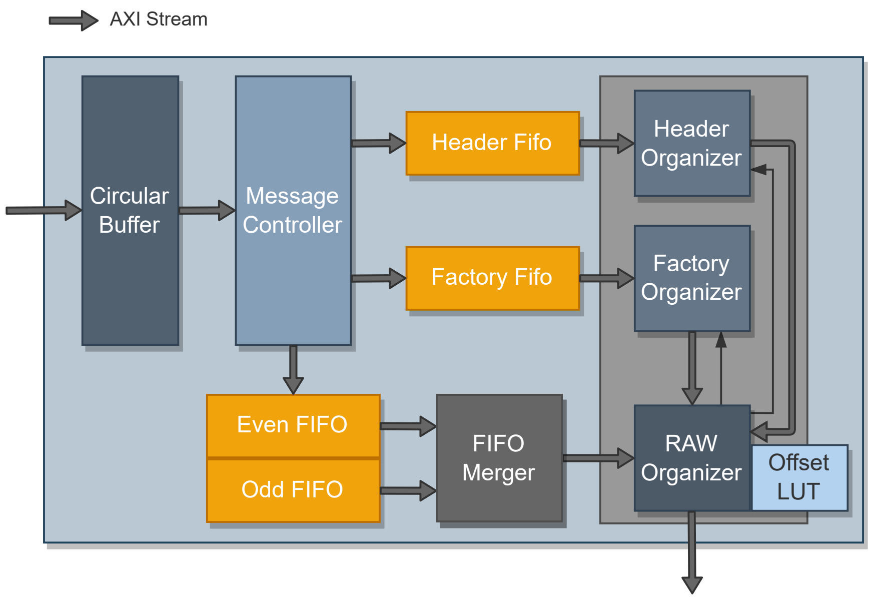


Figure 5.17: Velodyne Core system overview.

processing. Alongside the parallelization possibilities, these sub-modules also isolate the data input of the Velodyne Core from the data processing nodes, bringing data management advantages discussed further ahead.

As discussed in Section 4.2.2, a Velodyne sensor data packet is divided into twelve data blocks, each containing a header and data point information. Also, there is a distinction between even- and odd-numbered data blocks when the sensor is working in dual-return mode. Finally, the timestamp and factory code information can be found in the last field of the packet. Therefore, the system divides these sections into their correspondent FIFO for further processing: the Header FIFO stores the header section, the Even/Odd FIFO store the correspondent type of data block, and the Factory FIFO stores timestamping information and the factory code section.

Circular Buffer

Unlike the Hokuyo Core, the Velodyne Core does not require individual sensor buffers. As a result, if downstream modules cease data consumption from the ALFA-Pi Core, it is necessary to hold the incoming data packets until data processing resumes. As depicted in Figure 5.18, the Circular Buffer sub-module consists of an array containing multiple data buffers. Data enters through the AXI-Stream slave interface and is redirected to a buffer and stored. The sub-module uses two signals to control the data flow: the *tx_index*, which indicates the buffer that receives the data, and the *rx_index*, which specifies the buffer from where data is retrieved. When the sub-module receives an entire packet, it increments the *tx_index* signal, and when a buffer is emptied using the AXI-Stream master interface, the sub-module increments the *rx_index* signal. When one of these signals reaches the final array position, its value returns to zero creating a circular buffer composed of buffers. This flow occurs until there is no more data inside any buffer.

However, the modules that consume data from the Velodyne Core might stall it, or the system may not be consuming data at the same rate it is receiving, filling the buffers before they are emptied. Thus, when the *tx_index* reaches the *rx_index*, and the system is not reading data, the sub-module forces an increment of the *rx_index*, opening space for new data. This procedure allows the module to ignore old data in favor of new one, which is temporally more relevant. Also, when a reading operation starts, it is guaranteed that it will not stall (explained further), so the system can wait for it to end.

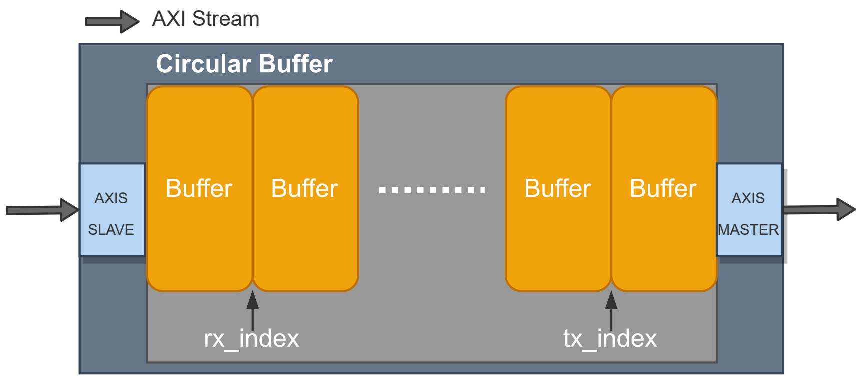


Figure 5.18: Circular Buffer block diagram.

Message Controller

The Message Controller separates the different packet components and distributes them through the respective FIFO sub-modules. As data is always in a specific place inside a packet, separating it is just a matter of counting the number of bytes that pass through the Message Controller. Figure 5.19 depicts the designed state machine that controls the Message Controller.

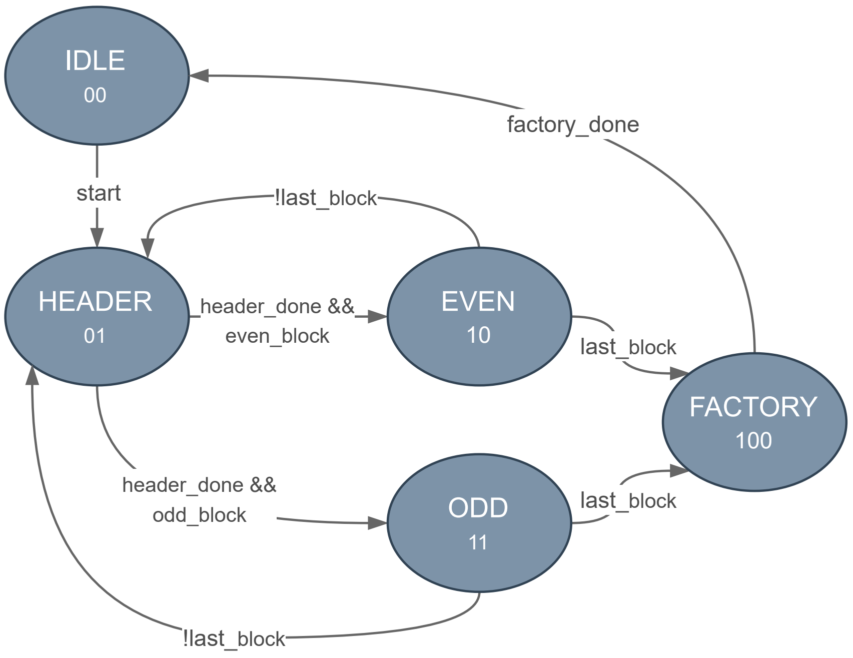


Figure 5.19: Message Controller state machine.

IDLE: The state machine starts in the IDLE state. The *start* signal triggers when space for an entire packet inside the FIFO sub-modules is available, causing the state machine to move to the HEADER state. This condition assures that the reading of the Circular FIFO never stalls.

HEADER: Within the HEADER state, the module filters the headers of data blocks. The sub-module uses a counter to know when four bytes have been transmitted to the Header FIFO, as this is the size the flag and the azimuth values occupy. When the counter reaches the value four, the state machine advances either to the EVEN state or the ODD state, depending on the type of the block.

EVEN/ODD: The EVEN and ODD states behave relatively similarly. Both are in charge of directing data blocks to the appropriate FIFO, each state being accountable for a single one, either Even or Odd. The state machine returns to the HEADER state until it processes all twelve data blocks present inside a data packet; otherwise, it transitions to the FACTORY state.

FACTORY: After processing all data blocks, the timestamp and factory code are the only data left to handle. These two elements are composed of six bytes. Like the HEADER state, a counter identifies when this data has been transmitted to the Factory FIFO. Lastly, the state machine returns to the IDLE state when the transmission ends.

FIFO Merger

If the sensor is in dual return mode, the information on both returns is needed simultaneously by the RAW Organizer. Since different FIFOs store the correspondent returns, it is necessary to merge their data. As Figure 5.20 depicts, the FIFO Merger features three modes. When the sub-module is in the first mode, only data from the Even FIFO is directed to the sub-module output. Contrarily, if the sub-module is in the second mode, data from the Odd FIFO is directed to the output. In both these modes, data is placed in the first 24 bits of the master interface, containing 48 bits of width (a data point contains three bytes). Lastly, when in the third mode, the sub-module merges the data from both Even and Odd FIFOs into the same output, allowing the RAW Organizer to access both returns concurrently, boosting the system's performance.

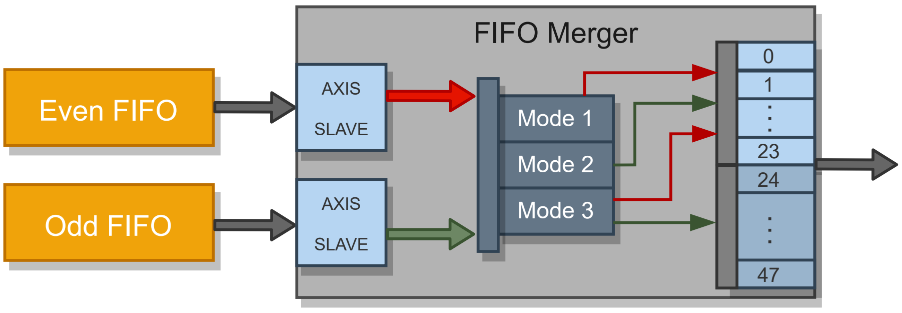


Figure 5.20: FIFO Merger behaviour.

Header Organizer

The Header Organizer is responsible for organizing and pre-processing header data. As depicted in Figure 5.21, the Header Organizer features a register bank with 12 positions, allowing quick access to specific azimuth values of different data blocks. The sub-module uses the *azimuth* port to output azimuth information to the RAW Organizer. Those values are chosen using the *data_block* input port, which is responsible for selecting the right position from the register bank.

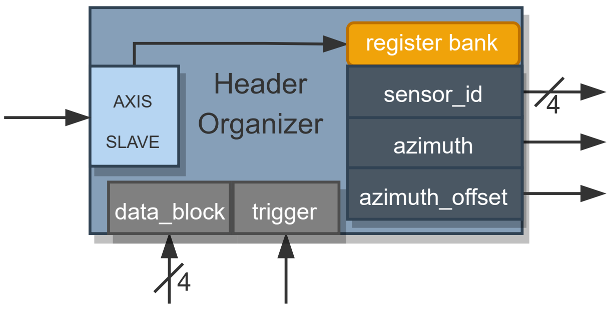


Figure 5.21: Header Organizer block diagram.

As previously mentioned, when the sensor has less than 32 laser channels, the sensor's resolution causes an offset in the azimuth of the second section of the data block. Based on Equation 5.1, the sub-module calculates this azimuth resolution offset by getting two consecutive headers, subtracting the first azimuth, $azimuth_1$, to the second, $azimuth_2$, and dividing the result by two. This operation is required because when the sensors operate at different rotation speeds, their horizontal resolution changes, causing the offsets to also change.

$$azimuth\_resolution\_offset = \frac{azimuth_2 - azimuth_1}{2} \tag{5.1}$$

Lastly, when the sub-module receives a signal through the *trigger* input port, it starts communication over the AXI-Stream slave interface, populating the register bank. The RAW Organizer is responsible for this trigger, sending it at the beginning of the data packet process, updating the register bank with 12 new header values, previously present inside the Header FIFO.

Factory Organizer

The Factory Organizer is very similar to the Header Organizer, regarding functionality. However, the Factory Organizer is responsible for processing the timestamping information and factory codes. When the RAW Organizer triggers the Factory Organizer sub-module, it starts communicating with the Factory FIFO. Next, this sub-module processes the data and outputs the factory code and mode from the sensor.

RAW Organizer

The RAW Organizer is the sub-module that processes the received data points and organizes them into the generic output format. As stated in Section 4.2.2, Velodyne Lidar sensors achieve their vertical angles by fixing the laser channels with different elevations around the same azimuth. In addition, the laser channels are also vertically misaligned. However, each sensor model has these values fixed, which aids in the application of these offsets. Therefore, a look-up table stores these different model values for real-time access. The Offsets Look-Up Table (LUT) features two tables for each sensor the ALFA-Pi Core supports from Velodyne Lidar. This sub-module uses the factory code, which specifies the type of sensor that sent the received packet, and the laser channel ID, to select which offsets are to be applied. The factory code selects the tables, and the laser channel ID chooses the position inside those tables from where data is taken. Alongside this information, the Offsets LUT also indicates how many laser channels the identified sensor features and the sensor's granularity.

One of the most important tasks the RAW Organizer must perform to decode the received data is correctly identifying the laser channel ID. This task is different depending on the mode the sensor is operating in. The laser channel IDs come distributed sequentially, where the first data point of the first data block corresponds to the laser channel ID zero, and from there, each consecutive data point features the previous laser channel ID plus one. Lastly, the value returns to zero when the number of laser channels is reached. If the sensor is in single-return, this is observed from data block to data block. Contrarily, if the sensor is in dual-return, it happens in pairs of two data blocks. Examples of the data block structures can be consulted in Figures 4.9 and 4.10, and a flowchart of the algorithm used to determine the correct laser channel ID is represented in Figure 5.22.

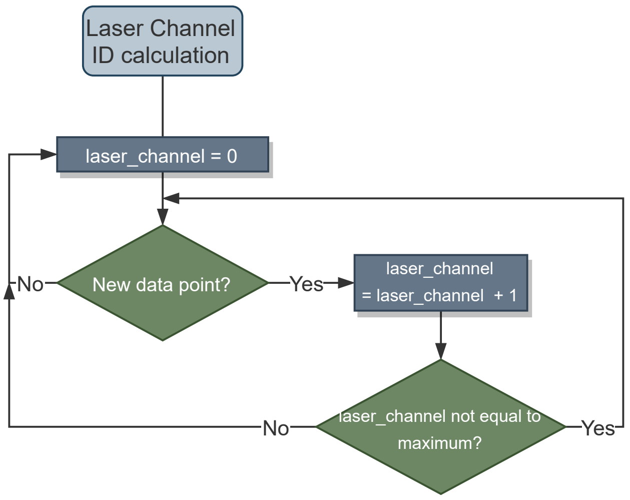

Laser Channel ID calculation
laser_channel = 0
New data point?
No
Yes
laser_channel = laser_channel + 1
laser_channel not equal to maximum?
No
Yes

Figure 5.22: Laser channel ID calculation flowchart.

After calculating the laser channel ID, the next step is to calculate the azimuth with all the offsets mentioned earlier: the azimuth offset correspondent to the laser misalignment and the azimuth offset caused by the angular resolution in sensors with less than 32 laser channels. The header present in a specific data block determines the base azimuth value to which the offsets are added. Each data block contains 32 data points, therefore, for every 32 data points, the data block is incremented, and the base azimuth changes. This value is extracted from the Header Organizer, as stated earlier. The pseudo-code for this calculation can be consulted in Algorithm 1.

Algorithm 1 Azimuth Calculation pseudo-code

```
Require:
    azimuth_res_offset: Azimuth offset caused by the angular resolution in sensors with less than 32
    laser channels
    azimuth_mis_offset : Azimuth offset correspondent to laser misalignment

1: base_azimuth = header_organizer_azimuth
2: while counter < 32 do
3:     if counter > number_laser_channels then
4:         azimuth = base_azimuth + azimuth_mis_offset + azimuth_res_offset
5:     else
6:         azimuth = base_azimuth + azimuth_mis_offset
7:     end if
8: end while
```

This sub-module's last task is to get the correct distance information and reflectivity values. The RAW Organizer receives the data point information from the FIFO Merger sub-module for the decoding process. The distance data is adjusted using the sensor granularity given by the Offset LUT, and the reflectivity data is placed inside the correspondent Extra field, present inside the generic output format. Figure 5.23 depicts the state machine that controls the RAW Organizer. In the states, SR_ANALYZE and DR_ANALYZE, the sub-module calculates the azimuth, laser channel ID, and distance/reflectivity values, as addressed.

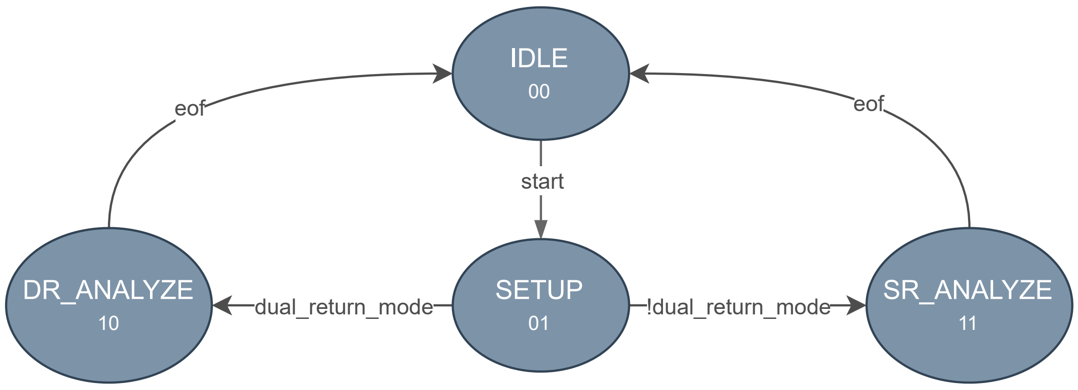


Figure 5.23: RAW Organizer state machine.

IDLE: The state machine starts in the IDLE state. When the Factory FIFO has data inside, meaning that an entire packet is available since it is the last FIFO to receive data, the *start* signal triggers, and the state machine advances to the SETUP state.

SETUP: The SETUP state is used as a transitioning state while the Header Organizer and the Factory Organizer fetch data. In this state, the *trigger* signals, mentioned in the explanation of those sub-modules, is set to one, activating them. After fetching the return mode from the Factory Organizer, the state machine moves to the DR_ANALYZE or the SR_ANALYZE, depending on if the sensor is in single or dual return.

SR_ANALYZE: The SR_ANALYZE controls the RAW Organizer when the mode is single-return. As previously stated, all necessary values are calculated using the processes already described. The only thing to note is the control of the FIFO Merger sub-module. When in this state, the mode of the FIFO Merger sub-module oscillates between one and two depending on the data block (even or odd). Finally, when the RAW Organizer detects it is on the last data block and the last data point, it triggers the *end_packet* flag of the general output format, and the state machine returns to IDLE.

DR_ANALYZE: The DR_ANALYZE is responsible for the RAW Organizer when the mode is dual-return. When in this state, the RAW Organizer processes both sensor returns simultaneously. For this reason, the FIFO Merger is used in the third mode, allowing the parallelization of this task. Consequently, the system increments the data block counter in sets of two instead of one, unlike the SR_ANALYZE. Lastly, when the sensor is the VLS-128, a particular operating flow is used. When this sensor's model is in dual-return mode, the last four data blocks do not contain any data; therefore, they must be discarded. Finally, when on the final data block and data point, the *end_packet* flag is set to one, and the state machine returns to IDLE.

5.2 Software

The software layer in this work is based on a ROS environment (ROS1 melodic distribution) running on top of a Linux operating system (5.4-xilinx-v2019.2). The software is supported by the developed hardware platform, with the Central Processing Unit (CPU) running at a clock speed of 1.2 GHz and the DDR4 memory running at 535 MHz. In addition, a ROS package allows the software layer to configure the ALFA-Pi Core's hardware modules.

5.2.1 Linux Configurations

Several kernel configurations and device tree modifications had to be performed to allow the system to integrate the new Ethernet interface. The Linux distribution was created and customized using the OpenEmbedded Framework present in Yocto. The following configuration options had to be activated in the kernel to enable the use of the deployed hardware Ethernet Interface on Linux:

- CONFIG_ETHERNET
- CONFIG_NET_VENDOR_XILINX
- CONFIG_XILINX_AXI_EMAC
- CONFIG_XILINX_PHY

Lastly, an entry inside the device tree had to be added for the AXI 1G/2.5G Ethernet Subsystem, allowing Linux to add the Ethernet interface as an Ethernet device inside the system. From here, Linux can configure the AXI 1G/2.5G Ethernet Subsystem IP core with various possible configurations and speeds, using Xilinx's device drivers.

5.2.2 ALFA-Pi Package

The ALFA-Pi Package is a ROS package that features one node and several services, which configure the ALFA-Pi Core and add/remove sensors from the interface. The node receives the service requests and writes the new information in the correct memory positions present in the hardware, configuring it. It also keeps track of this information and reports it if the user desires.

- **Node:** *alfa_pi_node*

  This node is responsible for implementing the service servers, responding to the service requests, and configuring the hardware.

- **Service**: add_velodyne_sensor **Parameters**: ip_address sensor_id

  This service adds a new Velodyne Lidar sensor to be processed by the ALFA-Pi Core. As only the IPv4 address is needed to decode a Velodyne sensor, the only thing to provide to this service is the IPv4 address of the sensor and the desired sensor ID.

- **Service**: add_hokuyo_sensor **Parameters**: ip_address distance_intensity sensor_id

  This service adds a new Hokuyo sensor for the ALFA-Pi Core to process. This service accepts one more parameter to configure the sensor to send distance-intensity or distance data. This service configures the hardware and starts a thread responsible for the TCP communication with the sensor since TCP is a connection-oriented protocol. However, the thread does not use the received information in the software layer, only maintaining the connection with the sensor.

- **Service**: remove_sensor **Parameters**: sensor_id

  This service removes the sensor with the specified ID from the ALFA-Pi Core. Alongside this, if the sensor is an Hokuyo sensor, the quit command will be sent to the sensor, stopping it from sending any more data.

- **Service**: sensor_info **Parameters**: sensor_id

  This service receives a sensor ID and reports its state and IPv4 address. In addition, if the requested sensor ID belongs to an Hokuyo sensor, the controller ID is also reported.

6. Evaluation and Results

This Chapter presents the tests performed on the developed system to validate, evaluate, and compare it to the currently available solutions. The tests start by assessing the impacts the ALFA-Pi system causes to the network performance, followed by a validation of the decoding process against the sensors' ROS native drivers. Next, Section 6.3 summarizes a full evaluation of the system performance compared to the native solutions provided by the original equipment manufacturer. Finally, the resource consumption of the different components of ALFA-Pi is exposed. All the tests were performed on top of an embedded Linux (5.4-xilinx-v2019.2) with a ROS environment (ROS1 melodic distribution), and the hardware platform combines the Zynq UltraScale+ MPSoC ZCU104 board and the EVAL-CN0506-FMCZ Ethernet expansion board, with the CPU running at a clock speed of 1.2 GHz and the FPGA fabric running at 100 MHz.

6.1 Ethernet Interface System

As with any interface, it is necessary to validate if the EIS can perform communication tasks with other devices, the quality of this communication, and if the Ethernet speeds suffer from having all of the ALFA-Pi Core components attached. After validating the basic functioning of the system using pings and simple TCP and UDP clients, the iPerf3 tool was used to get various performance metrics on the interface. The iPerf3 provides active measurements of the maximum achievable bandwidth on IPv4 networks, alongside the number of lost datagrams, jitter, and others. The tests with iPerf3 consist of using a server and a client between the two communicating nodes, where the iPerf3 tool keeps track of the transmission's performance metrics and displays them. Each test took five minutes to complete, with sampling every second, and was performed using an external desktop computer with a 1 Gbit/s enabled Ethernet interface. The tests were conducted on three different setups: (1) native Ethernet port of the ZCU104 Evaluation Kit; (2) EIS alone; and (3) EIS with ALFA-Pi accelerators. The network stack is managed by the Linux network manager in all configurations and both UDP and TCP data streams were tested. Table 6.1 summarizes the results for all test configurations. The maximum bandwidth rate achieved for TCP is roughly 941 Mbits/s in all three setups, while the maximum bandwidth for UDP is nearly 955 Mbits/s, also in all three configurations. The number of TCP retries is always zero, and the lost packet ratio in UDP tests is as low as 0.001 %. The jitter measured is always 0.018 ms across the three setups.

Table 6.1: Network Performance Results.

	Metrics	Native Ethernet	Ethernet Interface System	Complete Solution
TCP	Bandwidth (Mbits/s)	941.243	941.904	941.302
	Retries	0	0	0
UDP	Bandwidth (Mbits/s)	955.937	955.929	955.971
	Packet Loss (%)	0.002	0.001	0.002
	Jitter (ms)	0.018	0.018	0.018

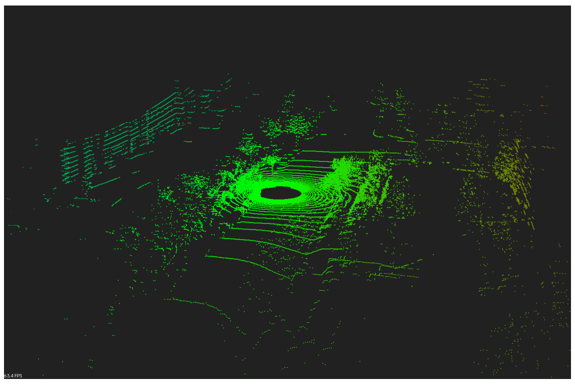
6.2 ALFA-Pi Core Validation

To validate the correctness of the ALFA-Pi Core's output, it is necessary to compare it with the ROS native solutions provided by the manufacturers. Comparing point clouds from two different systems is not as straightforward as it might seem. When a data transformation occurs in two different systems, their output can present slight differences because of different rounding methods, architecture, and others. Therefore, evaluating if point x on point cloud A is identical to point y on point cloud B requires a different approach. The used method gets one point from point cloud A (ROS native) and checks the obtained point's position inside point cloud B (ALFA-Pi). From here, the system searches a radius around that position, which grows until a point is found. When the system encounters a point, the distance between its position and the original position is added to the total error. The test only ends when all the points in point cloud A have been processed. The application calculates the average error between the two point clouds by dividing the total error distance by the number of processed points, and the results are satisfactory if the error is below the sensor's accuracy. Table 6.2 displays the results for the four tested sensors.

Table 6.2: Point cloud error ratios.

Sensors	Average Point Cloud Size Manufacturer	Average Point Cloud Size ALFA-Pi	Minimum Error (m)	Maximum Error (m)	Average Error (m)
UST-10LX	1080	1080	1.8e-8	5.1e-3	1.4e-3
VLP-16	16112	29184	2.4e-9	9.7e-5	3e-7
HDL-32E	62163	69504	8.8e-11	3.9e-4	5.2e-7
VLS-128	195494	240384	6.2e-11	6.1e-6	8.6e-8

For the Hokuyo results, the maximum error between point clouds was $5.1e^{-3}$ m, below the sensor's best accuracy of 40 mm. Unfortunately, it was only possible to test the Hokuyo Core with a single sensor, the UST-10LX. However, considering the positive results, it is safe to conclude that the hardware driver works correctly with other Ethernet-enabled Hokuyo sensors as long as they use the SCIP2.0 protocol. In contrast to the tests performed for the Hokuyo sensor, the Velodyne driver was validated using network captures from multiple sensors. Since Velodyne Lidar sensors use a connectionless network protocol, it is possible to use network captures with the same effect as having the actual sensor in the network. Figure 6.1 displays images extracted from the testing, which belong to an HDL-32E sensor on a car parked at the side of the road. Visually, and since the achieved error average is as low as $5.2e^{-7}$ m, the results from the two drivers are almost identical.


(a) Point cloud extracted using Velodyne's ROS driver.

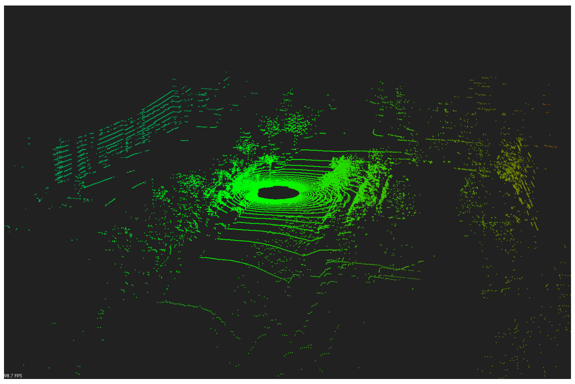

(b) Point cloud extracted using proposed system.

Figure 6.1: Comparison of Velodyne driver's point clouds.

The size of the point clouds is also displayed since they differ from one implementation to the other because of the pre-processing Velodyne's ROS driver performs. For example, the Velodyne's ROS driver excludes all the points present at a distance of less than 0.4 m from the source. However, this feature is not implemented in the proposed driver since it can be advantageous to know the number of points that failed to meet a target. The highest error between the three tested sensors is $3.9e^{-4}$ m, with average errors well bellow the sensor's accuracy.

6.3 ALFA-Pi Core Performance

The two supported manufacturers present very distinct output data formats. Consequently, each decoding block features different performance metrics due to the design decisions taken to accommodate and handle their respective data formats, making the ALFA-Pi Core performance very dependent on the sensor model/manufacturer. Therefore, it is necessary to compare each driver inside the ALFA-Pi core with the respective manufacturer's ROS native driver. After the Ethernet interface receives the point cloud data, the native ROS sensor's drivers read it from the Linux network stack manager, reconstruct the point cloud in software using the cartesian coordinate system, and make it available to high-level applications. On the other hand, the ALFA-Pi hardware blocks read sensor data straight from the Ethernet interface, decode the packets into the spherical coordinate system, and make the data available to other hardware modules. Thus, it is necessary to expand the ALFA-Pi's functionalities to fairly compare its performance with the ROS native drivers. Since the ALFA-Pi uses industry-standard protocols to communicate in the hardware (AXI-Stream) and a generic output format, it is only necessary to add the modules that provide the needed functionalities without any configuration or modification. To allow the basic ALFA-Pi system to provide data to the software layer in the cartesian coordinate system, a trigonometry core from the ALFA framework was integrated into the system, alongside an AXI DMA IP core. The trigonometry core allows fast transformation between coordinate systems in the hardware, accelerating a computationally heavy task, and the DMA core enables the transfer of data from the hardware into DDR4 memory, making it accessible by high-level applications.

Table 6.3 summarizes the performance metrics of the native ROS driver, the normal ALFA-Pi system, and ALFA-Pi with the extended functionalities. With the Hokuyo sensor and its native ROS driver, it takes approximately 2.642 ms to process a point cloud of 1081 points/frame when the sensor outputs only the point's coordinates (horizontal angle and distance), and 2.647 ms when the sensor outputs the point's intensity values as well. Conversely, the Velodyne ROS driver requires around 8.64 ms for the VLP-16, 21.58 ms for the HDL-32, and 72.11 ms for the VLS-128.

Table 6.3: Comparison of the native ROS drivers vs. ALFA-Pi with software support and ALFA-Pi hardware.

Sensor	Point Cloud Size	Native ROS driver (ms)	ALFA-Pi hardware (ms)	Gain1	ALFA-Pi software (ms)	Gain2
UST-10LX (Distance)	1081	2.642	0.032	81.55x	0.0435	60.73x
UST-10LX (Distance-Intensity)	1081	2.647	0.064	40.86x	0.0759	34.87x
VLP-16	29184	8.64	0.61	14.16x	0.929	9.3x
HDL-32	69504	21.58	1.46	14.78x	2.21	9.76x
VLS-128	240384	72.11	5.06	14.2x	7.65	9.42x

When resorting to ALFA-Pi, the system can achieve performance gains (column Gain1) of up to 81.55x for the Hokuyo sensor in distance mode and 40.86x for distance-intensity. The message format used by Hokuyo, which requires several data packets to be received and stored in memory, can explain the large amount of performance gain when comparing the two systems since the ROS driver needs to perform multiple memory accesses, slowing down the decoding process. On the other hand, the improvement is around 14.16x for the VLP-16, 14.78x for the HDL-32, and 14.2x for the VLS-128. The similar gains between Velodyne sensors are due to the fact that the system takes the same amount of time to process a data packet independently of the sensor model, meaning that the only performance conditioning factor is the number of data packets needed to define a full frame. In addition, the gains from ALFA-Pi with the extended functionalities (column Gain2) demonstrate that even when transferring data between operating layers, the raw performance from the basic ALFA-Pi system can compensate for the delays introduced by data transmission. The exposed times reveal that the ALFA-Pi can dramatically reduce the latency in accessing decoded LiDAR data, even when the data consumers are higher-level applications.

6.4 Hardware Resources Utilization

Like any hardware system, ALFA-Pi offers its features in exchange for hardware resources. It presents various configuration possibilities that alter its behavior and resource utilization, which raises the necessity to assess the resource utilization of the different modules separately. Moreover, testing different configurations for the number of controlled sensors (between 1 and 16 sensors) allowed the determination of the system's scalability. Table 6.4 summarizes all the required FPGA fabric resources (from the available in the Zynq UltraScale+ MPSoC) regarding LUT, Look-Up Table Random Access Memory (LUTRAM), Flip-Flop (FF), and Block RAM (BRAM).

Table 6.4: Hardware resources utilization.

	Number of Sensors	**LUTs (230400)**	**LUTRAMs (101760)**	**FFs (460800)**	**BRAMs (312)**
Ethernet Interface System	-	4683 (2.03%)	556 (0.546%)	9213 (2%)	5.5 (1.76%)
Core	1	107 (0.046%)	0 (0%)	262 (0.057%)	0 (0%)
	2	218 (0.095%)	0 (0%)	334 (0.072%)	0 (0%)
	4	264 (0.12%)	0 (0%)	427 (0.093%)	0 (0%)
	8	358 (0.16%)	0 (0%)	612 (0.13%)	0 (0%)
	16	567 (0.25%)	0 (0%)	981 (0.21%)	0 (0%)
Velodyne Core	1	419 (0.18%)	16 (0.016%)	360 (0.078%)	4 (1.28%)
	16	419 (0.18%)	16 (0.016%)	360 (0.078%)	4 (1.28%)
Hokuyo Core	1	196 (0.085%)	16 (0.016%)	421 (0.091%)	2 (0.64%)
	2	316 (0.14%)	16 (0.016%)	514 (0.11%)	5 (1.6%)
	4	490 (0.21%)	16 (0.016%)	700 (0.15%)	10 (3.2%)
	8	865 (0.38%)	16 (0.016%)	1072 (0.23%)	20 (6.41%)
	16	1586 (0.68%)	16 (0.016%)	1816 (0.39%)	40 (12.82%)

The Ethernet Interface System does not feature customization parameters that change the number of resources used and is independent of the number of sensors the system controls. Each independent Ethernet port requires 4683 LUTs, 556 LUTRAMs, 9213 FFs, and 5.5 BRAMs.

The Core component encapsulates the Packet Filter and all the other modules that integrate the ALFA-Pi Core and can not be individually disabled/changed. The resources utilized by the Core component increase with every sensor added to the system since the Packet Filter module needs more memory positions to store sensor information, affecting the amount of FFs and LUTs used. Nonetheless, connecting 16 sensors to the system keeps the resource consumption as low as 567 LUTs, 981 FFs, and zero LUTRAMs and BRAMs units, independently of the model/type of the sensor connected.

The Velodyne Core has the best scalability on the system. Since the Velodyne packets are totally independent from each other and the Velodyne Core gets all the information it needs directly from the received packet, there is no need to store any data regarding each sensor. Consequently, adding more Velodyne sensors to the system does not influence the number of resources used, which are as low as 419 LUTs, 16 LUTRAMs, 360 FFs, and 4 BRAMs.

The Hokuyo Core resource consumption is very dependent on the number of sensors the system is controlling. The most critical rise in resources used occurs with BRAMs since each sensor has a specific buffer for packet storage until a complete message arrives. Consequently, the amount of necessary BRAMs doubles when the number of sensors doubles. Notwithstanding, if the system starts being limited by BRAM scarcity, the storage type can be easily changed to, for example, LUTRAM. Therefore, despite the quick rise in BRAM consumption, the Hokuyo Core can still be very scalable since the amount of other resources used is relatively low, with only 1586 LUTs, 16 LUTRAMs, and 1072 FFs used when controlling 16 sensors.

6.5 Discussion

As the performance results demonstrate, ALFA-Pi can easily control dozens of sensors at their maximum speed. Although these performance values are not strictly necessary, the faster a driver can decode the data from a sensor, the sooner the decoded data arrives at the processing modules. In an automotive environment, every split second matters since a vehicle can be traveling at speeds well above the 100 km/h mark. Modern ADAS are very complex systems that rely on the interaction of various modules to make a decision, with each module introducing a delay, which can result in several milliseconds until all the processing tasks finish. Thus, a system that can control various sensors at very high speeds, in the order of microseconds, can help mitigate this problem by accelerating the decoding tasks. Furthermore, with the rise of multi-sensor perception systems, it is necessary to employ custom solutions capable of not only handling the vast amounts of data LiDAR sensors output, but also control different types of sensors simultaneously. Therefore, one of the biggest advantages of ALFA-Pi is the support for distinct sensors that can be used in different automotive LiDAR applications. For instance, Velodyne sensors are 3D sensors commonly used in long-range driving assistance tasks while the Hokuyo 2D sensors can be used in short-range tasks such as parking and blind spot assistance.

7. Conclusion

Nowadays, interest in developing and deploying fully autonomous vehicles is rising at an incredible pace [3]. An autonomous vehicle requires complex multi-sensor perception systems, including RADAR devices, cameras, and LiDAR sensors, to create a reliable representation of its surroundings [6, 7]. Despite the recent adoption of LiDAR sensors in the automotive field [1], LiDAR technology is already assumed as an essential element because of its ability to create a high-resolution representation of the surrounding environment in real-time [10, 11]. However, several challenges are still being tackled by academia and industry. Modern automotive LiDAR sensors generate high-resolution point clouds at a high frame rate, resulting in millions of data points per second [12]. As a consequence, data storage and transmission are two of the most problematic aspects of LiDAR sensor deployment [14]. This dissertation tackles these challenges by proposing and developing the ALFA-Pi, an agnostic and standard interface capable of driving automotive LiDAR sensors that output data via an Ethernet interface.

Like any other system, ALFA-Pi had some requirements to fulfill. Several hardware components were designed and implemented using FPGA technology, allowing the system to achieve good performance metrics and solve the high-speed needs imposed by the first requirement. As Chapter 6 discusses, ALFA-Pi can successfully handle multiple sensors from different manufacturers at the same time while the sensors operate at their maximum frequency. Also, as demonstrated in Chapter 4, ALFA-Pi uses a standard output format to output its data independently from sensor type or manufacturer, fulfilling the second requirement. To achieve the third requirement, all the hardware modules were encapsulated inside the ALFA-Pi Core, which features several AXI-Lite interfaces for real-time configuration and various compilation parameters that allow saving FPGA fabric resources by deactivating unnecessary functionalities. Lastly, and to satisfy the last requirement, the ALFA-Pi solution features a ROS package that uses the mentioned AXI-Lite interfaces for real-time configuration.

Additionally, the ALFA-Pi solution features some extra traits that appeared from the design approach. The system is highly modular, allowing easier maintainability and integration of new functionalities or sensors to support. Furthermore, the system is very scalable in the use of FPGA fabric resources, and due to the low baseline resource usage, it is also very portable to other platforms. However, like any system with hardware accelerators, the performance benefits come at the cost of FPGA fabric resources, which can be a major constraint when designing a system. Also, the system does not feature timestamping support, using instead other frame control mechanisms that mitigate this problem. However, when fusing data from multiple sensors, it can be advantageous to have timestamping information, to chronologically order the frames. Nonetheless, the modular approach taken to develop the system allows the integration of these capabilities either as an add-on module or built into the drivers themselves with relative ease.

In conclusion, this dissertation introduces the ALFA-Pi, a platform-independent and standard interface for driving multiple automotive LiDAR sensors that output data over an Ethernet interface. ALFA-Pi demonstrates that decoding LiDAR data in the hardware layer is a viable solution to tackle the high-speed requirements of multi-sensor LiDAR systems. It also presents good performance metrics compared to the manufacturer's native ROS drivers, even when the data consumer is present in the software layer. These reasons, allied with the standard output independent of sensor type and manufacturer, make ALFA-Pi a good solution for systems that use multiple LiDAR sensors.

7.1 Future Work

Although ALFA-Pi meets all of the original requirements, possible system improvements were detected in its design and development phases. However, it was not possible to integrate them into the final system due to time limitations. The following points are possible system improvements for future work:

- **Add support to more sensors:** Currently, the proposed system supports a large number of sensors. However, only sensors within the Velodyne Lidar and Hokuyo portfolio (UST and UTM) are supported. The LiDAR world is seeing the rise of new manufacturers with exciting and promising new technologies. Therefore, supporting more manufacturers would give the system more integrability and expandability.

- **Introduce timestamping capabilities:** Timestamping capabilities are necessary to successfully fuse point clouds from two sensors running at distinct frequencies. Although it is possible to implement this fusion with the framing control offered by the proposed output format, it is easier to perform it with timestamps of the generated point clouds. Therefore, implementing a timestamping mechanism either as an add-on module or into the ALFA-Pi Core would benefit the system.

- **Integration into the ALFA framework:** The proposed system was designed and developed for integration into the ALFA framework. Although the system uses generic AXI interfaces for communication, sometimes integrating a system into another is not straightforward. Therefore, when the main cores of the ALFA framework are completely implemented, the integration task can begin, which may or may not require changes to the proposed system.

References

[1] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, and P. Mahoney, "Stanley: The robot that won the DARPA Grand Challenge.," *J. Field Robotics*, vol. 23, pp. 661–692, 2006.

[2] I. Marques, J. Sousa, B. Sá, D. Costa, P. Sousa, S. Pereira, A. Santos, C. Lima, N. Hammerschmidt, S. Pinto, and T. Gomes, "Microphone array for speaker localization and identification in shared autonomous vehicles," *Electronics*, vol. 11, no. 5, 2022.

[3] T. Litman, *Autonomous vehicle implementation predictions*. Victoria Transport Policy Institute Victoria, Canada, 2021.

[4] "Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles," tech. rep., SAE International, Geneva, CH, 2021.

[5] 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.

[6] J. Guerrero-Ibáñez, S. Zeadally, and J. Contreras-Castillo, "Sensor Technologies for Intelligent Transportation Systems," *Sensors (Basel, Switzerland)*, vol. 18, no. 4, p. 1212, 2018.

[7] 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, no. 4, pp. 94–108, 2019.

[8] 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.

[9] 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.

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

[11] Y. Li and J. Ibanez-Guzman, "Lidar for Autonomous Driving: The Principles, Challenges, and Trends

for Automotive Lidar and Perception Systems," *IEEE Signal Process. Mag.*, vol. 37, no. 4, pp. 50–61, 2020.

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

[13] Waymo, "Waymo Driver." Accessed on: December 21, 2021. [Online] Available: `https://waymo.com/waymo-driver/`.

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

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

[16] 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.

[17] 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, no. 4, pp. 452–454, 1935.

[18] W. E. K. Middleton and A. F. Spilhaus, "Meteorological Instruments," *Quarterly Journal of the Royal Meteorological Society*, vol. 80, no. 345, p. 484, 1954.

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

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

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

[22] 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.

[23] U. Weiss and P. Biber, "Plant detection and mapping for agricultural robots using a 3D LIDAR sensor," *Robotics and Autonomous Systems*, vol. 59, no. 5, pp. 265–273, 2011.

[24] S. P. Healey, P. L. Patterson, S. Saatchi, M. A. Lefsky, A. J. Lister, and E. A. Freeman, "A sample design for globally consistent biomass estimation using lidar data from the Geoscience Laser Altimeter System (GLAS)," *Carbon Balance and Management*, vol. 7, no. 1, pp. 1–9, 2012.

[25] C. Urmson, J. Anhalt, M. Clark, T. Galatali, J. Gonzalez, J. Gowdy, A. Gutierrez, S. Harbaugh, M. Johnson-Roberson, P. Koon, K. Peterson, and B. Smith, "High Speed Navigation of Unrehearsed Terrain: Red Team Technology for Grand Challenge 2004," *Robotics Inst., Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-04-37*, 2004.

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

[27] 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.

[28] T. Fersch, R. Weigel, and A. Koelpin, "A CDMA Modulation Technique for Automotive Time-of-Flight LiDAR Systems," *IEEE Sensors Journal*, vol. 17, no. 11, pp. 3507–3516, 2017.

[29] 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.

[30] 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.

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

[32] Cepton, "Sora™-P Family." Accessed on: December 21, 2021. [Online] Available: `https://www.cepton.com/products/sora-p`.

[33] 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, 2017.

[34] 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.

[35] *VLP-16 User Manual*, rev. E, Velodyne Lidar, Sep. 1, 2018.

[36] 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.

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

[38] 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, 2017.

[39] 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.

[40] 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, no. 9, p. 1477, 2009.

[41] 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, no. 1, pp. 147–156, 2017.

[42] 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, 2002.

[43] 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.

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

[45] 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.

[46] LeddarTech, “Leddar IS16.” Accessed on: March 26, 2021. [Online] Available: `https://leddartech.com/solutions/is16-industrial-sensor`.

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

[48] 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.

[49] J. Guerrero-Ibáñez, S. Zeadally, and J. Contreras-Castillo, *Sensor technologies for intelligent transportation systems*, vol. 18. MDPI AG, 2018.

[50] 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.

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] 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.

[56] 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.

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

[58] 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.

[59] R. Roriz, A. Campos, S. Pinto, and T. Gomes, "DIOR: A Hardware-assisted Weather Denoising Solution for LiDAR Point Clouds," *IEEE Sensors Journal*, vol. 22, no. 2, pp. 1621 – 1628, 2021.

[60] A. M. Wallace, A. Halimi, and G. S. Buller, "Full Waveform LiDAR for Adverse Weather Conditions," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 7, pp. 7064–7077, 2020.

[61] J. Jiménez, "Laser diode reliability: Crystal defects and degradation modes," *Comptes Rendus Physique*, vol. 4, 2003.

[62] G. Atanacio, J.-J. Gonzalez-Barbosa, J. Hurtado-Ramos, F. Ornelas-Rodriguez, H. Jiménez-Hernández, T. Garcia-Ramirez, and R. Gonzalez-Barbosa, "LiDAR Velodyne HDL-64e calibration using pattern planes," *Int. J. of Adv. Robotic Systems*, vol. 8, 2011.

[63] D. Kim, T. Chung, and K. Yi, "Lane Map Building and Localization for Automated Driving Using 2D Laser RangeF," *2015 IEEE Intelligent Vehicles Symposium (IV)*, no. Iv, 2015.

[64] I. Maksymova, C. Steger, and N. Druml, "Review of LiDAR Sensor Data Acquisition and Compression for Automotive Applications," *Proceedings*, vol. 2, no. 13, p. 852, 2018.

[65] 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.

[66] T. Sun, Y. Liu, and Y. Wang, "Design and implementation of a high-speed lidar data reading system based on fpga," *2019 IEEE International Conference on Real-Time Computing and Robotics, RCAR 2019*, pp. 322–327, 2019.

[67] 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.

[68] 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.

[69] 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.

[70] 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.

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

[72] BMW Group, “Security in embedded IP-based systems,” 2009.

[73] N. Dimitrakopoulos and H. Dudragne, “Automotive Ethernet: The Future for In-Vehicle Networks.” Rohde and Schwarz, Apr. 3, 2019.

[74] M. V. Okunsky and N. V. Nesterova, “Velodyne LIDAR method for sensor data decoding,” *IOP Conference Series: Materials Science and Engineering*, vol. 516, p. 012018, 2019.

[75] Y.-C. Fan, Y.-C. Liu, and C.-A. Chu, “Efficient cordic iteration design of lidar sensors’ point-cloud map reconstruction technology,” *Sensors*, vol. 19, no. 24, 2019.

[76] Y.-C. Fan and S.-B. Wang, “Three-Dimensional LiDAR Decoder Design for Autonomous Vehicles in Smart Cities,” *Inf.*, vol. 13, no. 1, 2022.

[77] 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., 2018.

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

[79] 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., 2014.

[80] 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.

[81] 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.

[82] *AMBA AXI and ACE Protocol Specification AXI3, AXI4, and AXI4-Lite ACE and ACE-Lite*, rev. E, ARM, Feb. 22, 2013.

[83] *AMBA AXI4-Stream Protocol Specification*, rev. A, ARM, Mar. 3, 2010.

[84] *ZCU104 Evaluation Board: Product Guide*, ver. 1.1, Xilinx, Oct. 9, 2018.

[85] *AXI 1G/2.5G Ethernet Subsystem v7.2: Product Guide*, PG138, Xilinx, Sep. 20, 2021.

[86] *AXI DMA v7.1 LogiCORE IP: Product Guide*, PG021, Xilinx, Jun. 14, 2019.

[87] *AXI4-Stream Infrastructure IP Suite v3.0*, PG085, Xilinx, Nov. 17, 2021.

[88] *CN0506 Circuit Note*, ver. 0, Analog Devices, 2020.

[89] T. Zou, G. Chen, Z. Li, W. He, S. Qu, S. Gu, and A. Knoll, "KAM-Net: Keypoint-Aware and Keypoint-Matching Network for Vehicle Detection from 2D Point Cloud," *IEEE Transactions on Artificial Intelligence*, pp. 207–217, 2021.

[90] G. Chen, F. Wang, S. Qu, K. Chen, J. Yu, X. Liu, L. Xiong, and A. Knoll, "Pseudo-Image and Sparse Points: Vehicle Detection With 2D LiDAR Revisited by Deep Learning-Based Methods," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 12, pp. 7699–7711, 2021.

[91] *Communication Protocol Specification For SCIP2.0 Standard*, rev. 4, HOKUYO AUTOMATIC CO. LTD., Mar. 21, 2008.

[92] Velodyne Lidar, "Local Motors: Sustainable, Accessible Transportation Solution For All." Accessed on: Sep. 27, 2021. [Online] Available: `https://velodynelidar.com/case-studies/local-motors/`.

[93] Velodyne Lidar, "Velodyne Lidar Announces Autonomous Driving Collaboration with Ford Otosan." Accessed on: Sep. 27, 2021. [Online] Available: `https://velodynelidar.com/press-release/autonomous-driving-collaboration-with-ford-otosan/`.

[94] Velodyne Lidar, "Velodyne Product Lineup." Accessed on: September 18, 2021. [Online] Available: `https://velodynelidar.com/products/`.

[95] *Ouster - Software User Manual: Firmware v2.2.x for all Ouster sensors*, Ouster, Feb. 2, 2022.

[96] *RS-Ruby Users' Manual*, rev. 3.0.2, RoboSense, Oct. 7, 2020.

[97] *Pandar64: 64-Channel Mechanical LiDAR User Manual*, ver. 640-en-211010, Hesai, Sep. 10, 2021.