UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

telecos
BCN

Development of a Vehicle with an FPGA-Based Board and Positioning with Ultrasonic Transducers.

Bachelor's Degree Thesis

Submitted to the Faculty at the

Escola Tècnica Superior

d'Enginyeria de Telecomunicació de Barcelona

de la Universitat Politècnica de Catalunya

by

Manel Marin Sort

In partial fulfillment

of the requirements for the

DEGREE IN ELECTRONIC ENGINEERING

Advisor: *Juan Antoni Chávez Domínguez*

Barcelona, October 2024

Degree in Electronic Engineering

Resum

Impulsat pel meu interès personal en comprendre les complexitats de la tecnologia FPGA, especialment en com combinar de manera efectiva la Lògica Programable (PL) i el Sistema de Processament (PS), vaig dissenyar aquest projecte per explorar aquestes àrees en profunditat.

Aquest projecte se centra en el desenvolupament d'un sistema robòtic basat en FPGA utilitzant un System on a Chip (SoC) que inclou un microprocessador, amb l'objectiu principal de crear una sèrie d'eines educatives per a les assignatures de Sistemes Digitals Configurables i Sistemes Encapsulats.

El disseny inclou descripció de Hardware en VHDL per generar senyals PWM per al control dels motors, així com la gestió dels sensors de distància i una pantalla LCD. La comunicació entre el PL i el PS dins del SoC s'aconsegueix mitjançant ports AXI. Integrant aquests components, el projecte no només demostra les capacitats de la tecnologia SoC en sistemes encastats en temps real, sinó que també serveix com a recurs pràctic per a estudiants. Aquest projecte té com a objectiu proporcionar recursos clars i detallats que facilitin l'aprenentatge en disseny FPGA, integració maquinari-programari i robòtica.

Resumen

Impulsado por mi interés personal en comprender las complejidades de la tecnología FPGA, especialmente en cómo combinar de manera efectiva la Lógica Programable (PL) y el Sistema de Procesamiento (PS), diseñé este proyecto para explorar estas áreas en profundidad.

Este proyecto se centra en el desarrollo de un sistema robótico basado en FPGA utilizando un System on a Chip (SoC) que incluye un microprocesador, con el objetivo principal de crear una serie de herramientas educativas para las asignaturas de Sistemas Digitales Configurables y Sistemas Embebidos.

El diseño incluye la descripción de Hardware en VHDL para generar señales PWM para el control de los motores, así como la gestión de los sensores de distancia y una pantalla LCD. La comunicación entre el PL y el PS dentro del SoC se logra mediante puertos AXI. Al integrar estos componentes, el proyecto no solo demuestra las capacidades de la tecnología SoC en sistemas embebidos en tiempo real, sino que también sirve como un recurso práctico para estudiantes. Este proyecto tiene como objetivo proporcionar recursos claros y detallados que faciliten el aprendizaje en diseño FPGA, integración hardware-software y robótica

Summary________________________________

Driven by my personal interest in understanding the complexities of FPGA technology, particularly in how to effectively combine Programmable Logic (PL) and the Processing System (PS), I designed this project to explore these areas in depth.

This project focuses on developing an FPGA-based robotic system using a System on a Chip (SoC) that includes a microprocessor, with the primary goal of creating a series of educational tools for courses on Configurable Digital Systems and Embedded Systems.

The design includes writing VHDL to generate PWM signals for motor control, as well as managing distance sensors and an LCD display. Communication between the PL and PS within the SoC is achieved through AXI ports. By integrating these components, the project not only demonstrates the capabilities of SoC technology in real-time embedded systems but also serves as a practical resource for students and enthusiasts. This project aims to provide clear and detailed resources that facilitate learning in FPGA design, hardware-software integration, and robotics.

I would like to I dedicate this project to my cousin Charly, who was the first to introduce me to the fascinating world of electronics and ignite a curiosity within me that has continued to grow. I have always admired his passion and dedication. To my parents, who have unconditionally supported me in every aspect of my life. I also want to express my gratitude to all my friends, who have made these years at university a unique experience. Their friendship and support have been invaluable, and I am grateful to carry them with me for a lifetime.

.

Acknowledgements

I want to thank everyone who has helped me to grow and learn as a student during this time at collage that is getting to the end. This includes all the professors who have taught me, from the foundational concepts in my first year to the final project going through theoretical concepts. In particular, I would like to mention Juan Antonio Chávez, who has not only taught me in theoretical classes and laboratories but has also guided me throughout my final project.

Revision history and approval record

Revision	Date	Author(s)	Description
1.0	10/08/2024	MMS	Document creation
1.1	5/09/2024	JAC, MMS	Error correction
2.0	17/09/2024	JAC, MMS	Revised after review
3.0	3/10/2024	JAC, MMS	Final version

LLISTA DE DISTRIBUCIÓ DEL DOCUMENT

Role	Surname(s) and Name
Student	Manel Marin Sort
Project Supervisor 1	Juan Antoni Chávez Domínguez

Written by:		**Reviewed and approved by:**	
Date	0.3/10/2024	Date	05/10/2024
Name	Manel Marín Sort	Nome	Juan Antoni Chávez Domínguez
Role	Project Author	Role	Project Supervisor

Contents

List of Figures

List of Tables

Abbreviations

ETSETB Escola Tècnica d’Enginyeria de Telecomunicació de Barcelona

AXI Advanced eXtensible Interface

FPGA Field Programmable Gate Array

PS Processing System

PL Programable Logic

VHDL Vhsic Hardware Description Lenguage

PWM Pulse with modulation

LCD Liquid Crystal Display

1. Introduction

CHAPTER

1

Introduction

In the field of electronics, FPGA-based systems are becoming increasingly significant due to their flexibility and customization capabilities. Within this domain, the integration of FPGA and System on a Chip (SoC) technology is emerging as a powerful solution for designing complex systems. Specifically, SoCs that combine the Programmable Logic (PL) of an FPGA with a Processing System (PS) offer a robust platform for real-time applications and high-performance computing.

The versatility of SoCs allows for the combination of the FPGA's parallel processing capabilities with the computational power of the PS, making it ideal for implementing complex algorithms and handling multiple tasks simultaneously.

At the UPC multiple subjects are related to digital design and embedded systems. Particularly the subjects Digital Design, Configurable digital systems and High-performance systems are taught at the electronic telecommunications degree, using FPGAs. On the other hand, in the Embedded systems subject a microprocessor is programmed.

This project aims to provide the university with a valuable resource for teaching and demonstrating the integration of hardware and software in modern embedded systems ensuring that students not only learn the technical aspects of FPGA and SoC technology but also understand how to use these tools to create innovative solutions.

In this project, VHDL will be used to generate Pulse Width Modulation (PWM) signals for motor control, manage distance sensors, interface with an LCD display and manage input signals to control the robot.

Communication between the PL and PS is facilitated through AXI (Advanced eXtensible Interface) ports, demonstrating how efficient data exchange between hardware and software can be managed within an SoC environment.

The PS will execute an obstacle avoidance algorithm, allowing the robot to navigate autonomously by interpreting data from distance sensors and making real-time decisions. Additionally, the robot will feature a manual mode controlled via a radio control system.

1.1. Work goals

The primary objective of this work is to develop a small vehicle using an FPGA-based board that runs autonomously with distance sensors, creating a modular and comprehensive resource that can be effectively used in university courses such as Digital Design and Embedded Systems focused on FPGA technology.

Built this resource around a System on Chip (SoC) that integrates two essential components: an FPGA and a microprocessor. The FPGA component offers students practical experience in designing digital systems, helping them understand how digital circuits and systems are created and implemented. Meanwhile, the microprocessor component serves as a platform for programming and the analysis and design of microprocessor-based systems.

Build a project so is engaging and accessible, with a structure that divides learning into smaller, manageable goals showcasing the potential and possibilities of FPGA boards. This approach facilitates incremental learning, allowing students to build their knowledge step by step. Students should concentrate on learning concepts related to digital design, hardware description languages, and microprocessor programming. Moreover, they will use complex commercial tools to connect different modules that require different resources like synthesizer, compilers and simulators among others. This approach aims to help students appreciate the potential of combining programmable logic with microprocessors.

Use different sensors and peripherals such as distance sensor, LCD , motors which drivers are implemented using hardware description language like VHDL.

Additionally, emphasizes the use of AXI (Advanced eXtensible Interface) communication, which is essential for understanding how the FPGA's Programmable Logic (PL) interacts with the Processing System (PS). By mastering AXI, students will gain a deeper understanding of the differences between programming firmware, which controls hardware behavior, and programming hardware, which configures the circuits themselves.

1.2. Requirements and specifications

The project must follow the following requirements and specifications. The requirements are as follows:

- The procedures and building processes have to be well defined so it can be reproduced for college students.
- The project must utilize a System on Chip (SoC) that integrates both an FPGA and a microprocessor
- The project utilizes both the PL and PS parts through AXI ports, clearly demonstrating their different functionalities and possibilities both separately and together.
- The robot must feature two operational modes: autonomous mode for self-navigation and manual mode.
- The vehicle, in the autonomous mode, has to avoid obstacles and move towards where there is space.
- The robot must include a mechanism to display the distance of obstacles surrounding it.
- All components must be integrated into a single PCB to facilitate the easy assembly of the vehicle.

The specifications that this project must follow are as follows:

- The project will be modularly built so it can have a future utility in laboratory sessions of subjects like named earlier.
- The vehicle must be compact and of a size that can be easily reproduced and constructed within a typical college laboratory setting.
- PL will be written in VHDL and PS in C**.** VHDL will be used for designing digital circuits and managing peripherals, while C will be used for processing data and controlling system functions.
- A control can send 4 different instructions to the robot: Forward, Left, Right, and Backward**.** These commands will allow manual control of the robot's direction and movement.

1.3. Methods and procedures

To develop the vehicle with all its features and peripherals, we will follow these steps:

- Exhaustive research on FPGA and SoC to select one compatible with the educational proposes.
- Design the structure of all the functions of the vehicle
- Select all the components needed to accomplish all functions.
- Implement the control logic for peripherals using VHDL to manage sensors, motors display and input signals for the control.
- Configure AXI interfaces for communication between the Programmable Logic (PL) and Processing System (PS).
- Create an algorithm to enable the vehicle's autonomous movement on the PS.
- Implement control for manual operation, allowing commands like Forward, Left, Right, and Backward.
- Create a PCB that integrates all hardware components and fits with the board for easy assembly.

1.4. Work plan

The final work plan has been modified from the initial proposition. This is because some of the tasks took longer than expected, specifically the Programable Logic Description and its testing. In the same way the AXI configuration was not as easy as expected.

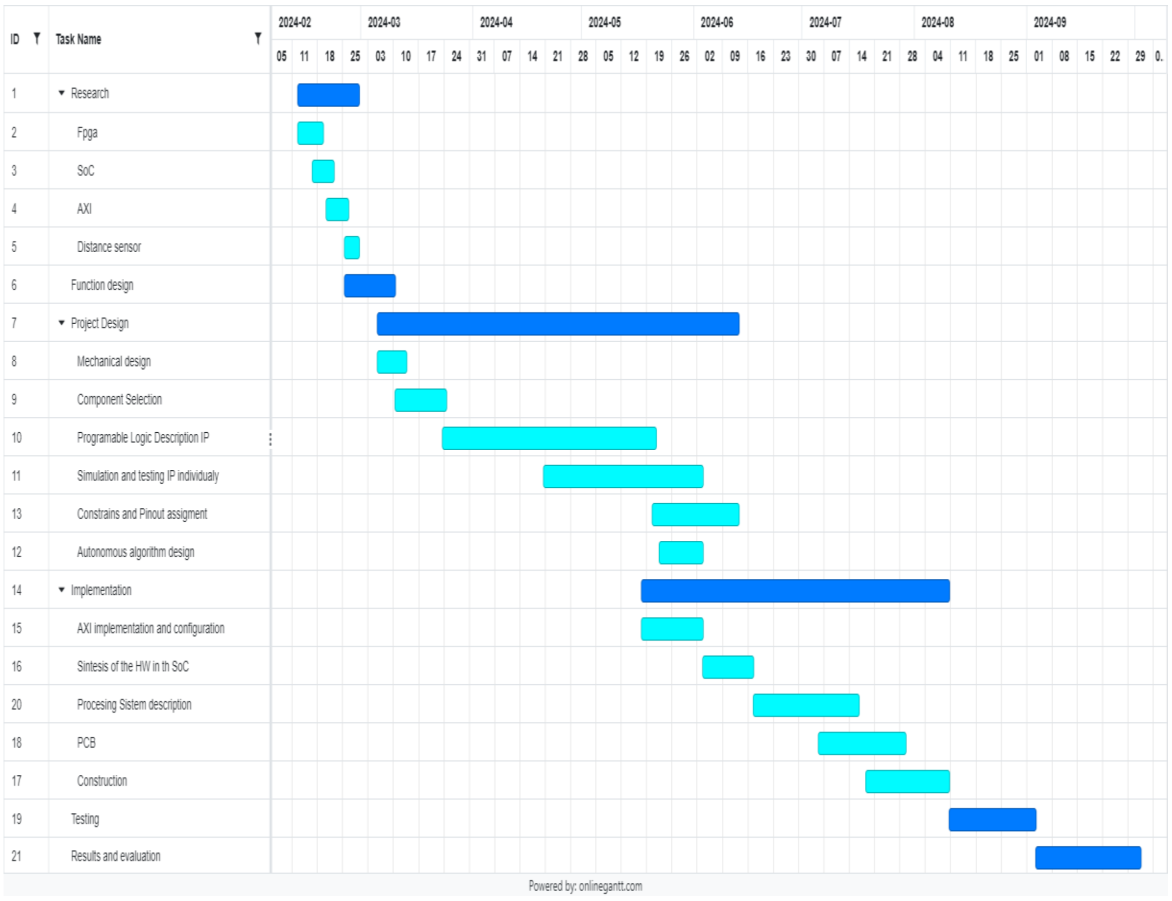


Figure 1 Diagram of Gantt

The tasks are explained in the following work packages:

WP1: Research of information of the technology wanted to use.

T1. FPGA

T2. SoC

T3. AXI

T4. Distance sensors

WP2: Design al the function that the vehicle has to accomplish

WP3: Design all the parts of the vehicle to execute the functions.

- T1. Mechanical design
- T2. Components Selection
- T3. Programable Logic description
- T4. Simulation and testing IP individually
- T5. Autonomous algorithm design
- T6. Constrains and pinout assignment

WP4: Implementation of all the parts designed earlier to put together the vehicle

- T1. AXI implementation and configuration
- T2. Synthesis of the HW in the SoC
- T3. Processing System description
- T4. Constrution

WP5: Test all the functions of the vehicle

WP6: Put together all the results and evaluations.

CHAPTER 2

State of the art of the technology used or applied in this thesis

In this final degree project, it is essential to understand FPGAs, microcontrollers and how they interact through AXI ports inside System-on-Chips (SoCs) since it is the main technology that is intended to show to the students. In this chapter we will go over the most important concepts

Additionally, we will cover the principles of distance sensor which play a significant role in this project capturing the information so the motors can act in consequence.

2.1. FPGAs

A Field-Programmable Gate Array (FPGA) is a type of programmable logic device that consists of simple logic blocks organized in a matrix structure, we can see an example in Figure 2. The three primary elements of an FPGA are configurable logic blocks (CLBs), registers, and programmable interconnection lines.

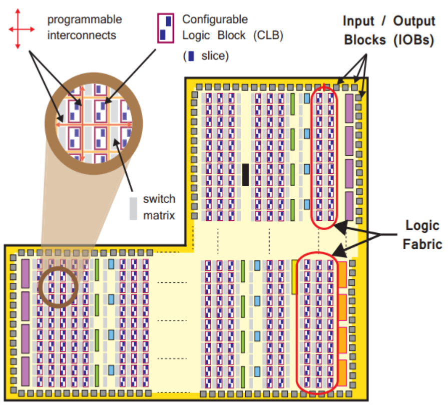

programmable interconnects
Configurable Logic Block (CLB)
(▮ slice)
Input / Output Blocks (IOBs)
switch matrix
Logic Fabric

Figure 2 FPGA architecture

To implement a circuit on an FPGA, the process begins with creating a digital design, providing a conceptual overview of the intended functionality. This design is then described using a hardware description language (HDL) and graphical schematics. In this project, VHDL has been utilized to describe the circuit. After the design phase, it's crucial to perform functional simulations to verify that the system operates as expected. Eny block design personalized is called Intellectual Property (IP).

The next step involves synthesizing the circuit. During synthesis, the HDL code is translated into a physical representation of the circuit, mapping the design onto the FPGA's resources. This includes selecting specific logic blocks and defining their interconnections. Once synthesis is complete, timing simulations are conducted to ensure the circuit meets performance requirements. Finally, the design is transferred to the FPGA via a bitstream which is a file that encodes all the synthesized information. After this step, the FPGA will function according to the described circuit, effectively becoming the custom hardware designed by the user.

2.2. Microprocessor

A microprocessor is an integrated circuit that contains all the necessary elements that are needed to operate a computer, which is why it is usually referred to as the "brain" of a computer. It usually contains one or more central processing units (CPUs) which mainly contain registers, an arithmetic logic unit (ALU) and a control unit (UC). It is responsible for executing the different instructions needed to run a computer program and it does so utilize simple arithmetic (addition, subtraction, multiplication, division) and logic (AND, OR, NOT) operations, as well as accessing memory.

2.3. System on a Chip (SoC)

A System on a Chip is an integrated circuit that integrates multiple electronic systems on a single silicon chip rather than several different physical chips being required. The limitations of ASIC SoCs are on their incompatibility with fast time-to-market, flexibility, and upgrade-ability. There is a clear need for a more flexible solution, and this is what motivates the System-on Programmable-Chip, a specific type of SoC implemented on a programmable, reconfigurable device.

Som SoCs have two interconnected differential parts, the Programable Logic (PL) section that it is ideal for implementing high-speed logic, arithmetic and data flow subsystems highlighting its flexibility to create custom peripherals and the Processing System (PS) that has a fixed architecture that supports software routines and/or operating systems, meaning that the overall functionality of any designed system can be appropriately partitioned between hardware and software. The interconnections are implemented via AXI interfaces linking the PS and PL. We can see an example of a SoC structure in the Figure 3.

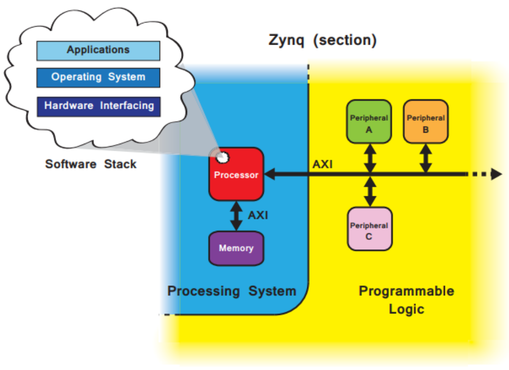


Figure 3 SoC structure

2.4. AXI protocol

AXI stands for Advanced eXtensible Interface, is an interface protocol defined by ARM as part of the AMBA (Advanced Microcontroller Bus Architecture) standard.

The current version is AXI4 and is the protocol used to communicate PS and PL. It is a protocol optimized for FPGA implementation through coordinated development with Xilinx.

The AXI protocol supports burst-based transactions, meaning that multiple data transfers can occur for a single request. Each transaction includes address and control information transmitted on the address channel.

The AXI protocol defines 5 channels:

- 2 are used for Read transactions shown in Figure 5
  - Read address: The transaction begins with the master sending the Address Read Channel to the slave, which sets the address and various control signals.
  - Read data: Following this, the slave transmits the data associated with the specified address back to the master
- 3 are used for Write transactions shown in Figure 4
  - Write address: The transaction starts with the master sending the Address Write Channel to the slave, which sets the address and necessary control signals.
  - Write data: Subsequently, the master sends the data corresponding to this address to the slave
  - Write response; Finally, the slave sends a response back to the master on the Write Response Channel, indicating whether the transfer was successful.

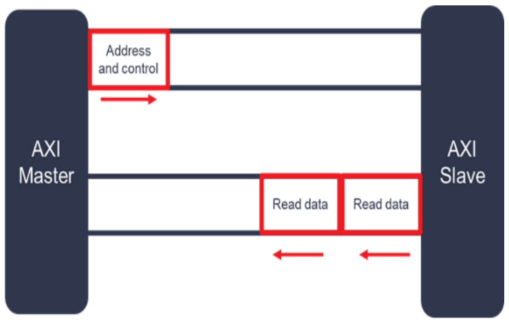


Figure 5 AXI Read

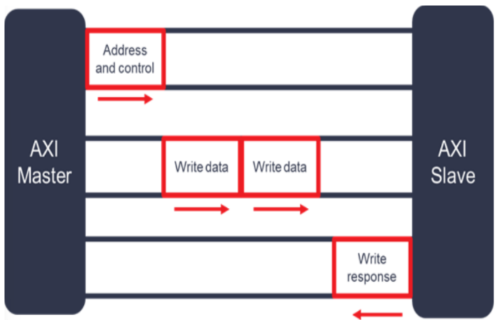


Figure 4 AXI Write

2.5. Distance sensors

A distance sensor is a device that measures the physical distance between itself and an object providing data that can be used in many different applications.

The working principle of distance sensors depends on the type of technology they use. Generally, a distance sensor emits a signal (such as light, sound, or electromagnetic waves) toward an object that travels at speed of v. The sensor then measures how long it takes for the signal to return after bouncing off the object. By calculating the time delay Δt and knowing the speed v at which the signal travels, the sensor can determine the distance to the object Δx . It has to be taken in consideration the fact that the wave travels the double of the distance of the sensor to the object. These calculations are reflected in the next equation. Figure 6 shows how the waves travel through the air.

$$\Delta x = v * \frac{\Delta t}{2}$$

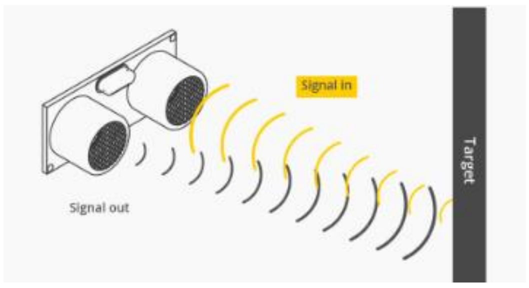


Figure 6 Distance sensor waves

CHAPTER 3

Methodology / project development

In this section we will focus on the procedures that have been followed to achieve the goal of building a Vehicle with an FPGA-Based Board and Positioning with Ultrasonic Transducers. We will discuss the decisions taken and we will see the details of the software code and the hardware design. We will also discuss the vehicle's construction, with a focus on the chassis and PCB design.

3.1. Function design

This project has been developed modularly, meaning it has different parts that each have clear and distinct functions. A function diagram can be seen in Figure 7. This describes all the internal functions that the vehicle has. The arrows indicate the direction of the connections. In Table 1 all the functional blocks are described.

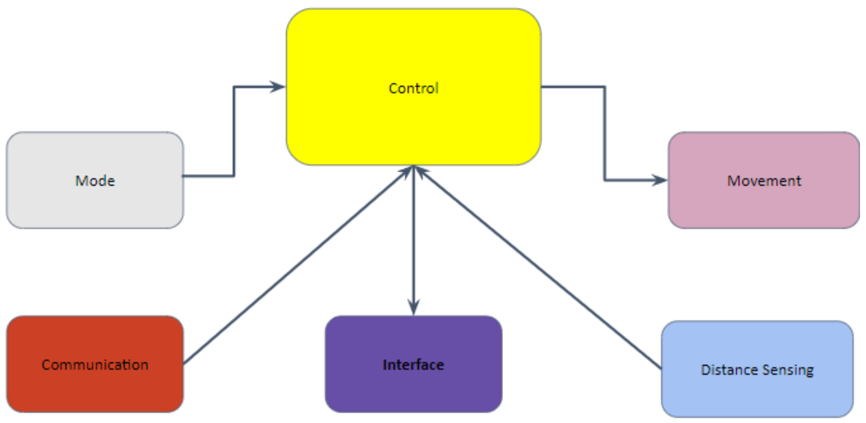


Figure 7 Function design

Block	Description
Control	Manages all the drivers written in VHDL , the software algorithm to avoid obstacles and all the signals necessary to run the peripherals.
Mode	Determines if the robot runs autonomously or in the manual mode.
Distance Sensing	Captures the distance to the surrounding objects
Communication	Manages the movement commands on the manual mode
Movement	All what is related to make the vehicle physically move
Interface	Communicate to the user the distance calculated

Table 1 Internal functions

3.2. Design proposition

After defining all the functions that we want the vehicle to do we have to define how and what components will perform them. The interconnections between all the parts that will compose the vehicle have to be also considered. In this section we will go over all the components selected and alternatives that have been considered to build this project.

3.2.1. Control

It is time to decide what development board is going to be used. Since the goal is to create a useful tool for subjects of digital design and high-performance systems, we will consider boards that are already been used in their laboratories.

On one hand we have the option to work with a DE-10 Standard SoC from Intel. It carries a Ciclon V and works with the development tool Quartus. On the other hand, there is the PYNQ-Z2 from Xilinx that carries a Zynq-7000 SoC and uses Vivado and Vitis as a development tool.

DE10-Standard is better suited for high-performance, industrial, and professional applications, while the PYNQ-Z2 is ideal for educational purposes, prototyping, and embedded systems development. The PYNQ-Z2 is smaller than the DE-10 so it will be easier to fit in the chassis.

It is also needed to say that from a personal perspective there is an especial interest in learning about the Xilinx environment leading to use the PYNQ-Z2 shown in Figure 8.

Focusing now on the PYNQ-Z2 the SoC Zynq-7000 has a PS formed around a dual-core ARM Cortex-A9 processor, with traditional PL FPGA Xilinx Artix-7 as well. The development board has also integrated memory, a variety of peripherals, and high-speed communications interfaces. PS and PL are interconnected through the AXI protocol explained in the state of art.

Vivado is the hardware design and verification environment used. Vitis extends its capabilities by enabling software development for embedded systems making easier to program the ARM Cortex-A9. In this project the version Vivado 2022.1 is used.

In the Figure 8 we can see the top part of the board with some of the peripherals and connectors that includes.

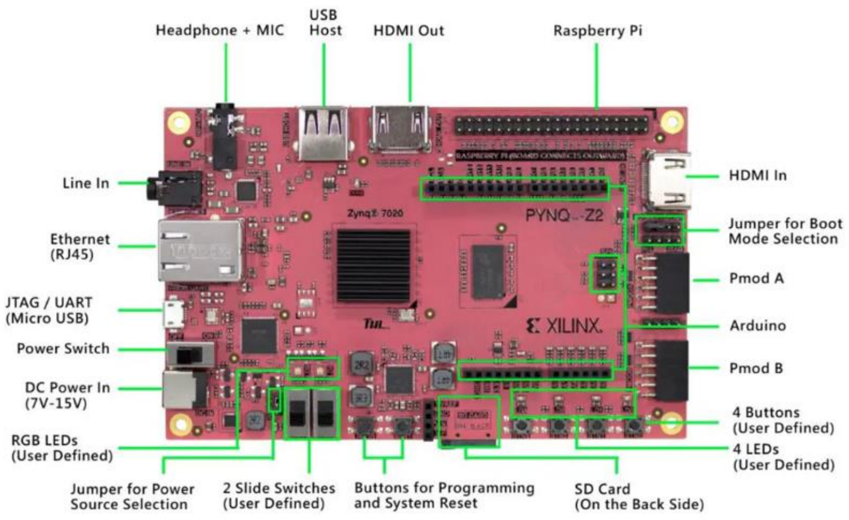


Figure 8 PYNQ-Z2

The main features to bear in mind are:

- 650MHz ARM® Cortex®-A9 dual-core processor
- Programmable logic
  - 13,300 logic slices, each with four 6-input LUTs and 8 flip flops
  - 630 KB block RAM
- Memory and storage
  - 512MB DDR3 with 16-bit bus @ 1050Mbps
  - 16MB Quad-SPI Flash with factory programmed 48-bit globally
- Power
  - USB or 7V-15V external power regulator
- Switches, Push-buttons and LEDs
  - 2 slide switches
- Expansion Connectors
  - 2xPmod ports
  - Arduino Shield compatible connector
  - 24 Total FPGA I/O

3.2.1.1. Processing System

The microprocessor can be seen as the brain of the vehicle, it is in charge to run the application and manage all the signals and communications to properly run all the functions except the interface. Uses the AXI ports to receive the necessary data and then act in consequence.

The first action that takes is to check the switch of mood to run ether the manual mood or the autonomous mood. In case the manual mood is ON the code will read the AXI ports corresponding to de commands received from the radio control. Otherwise, will run the algorithm to avoid obstacles autonomously using the distance sensors. In the Figure 9 we can see a flowchart of the software runed in the microprocessor. The code used can be seen in the Appendix [II].

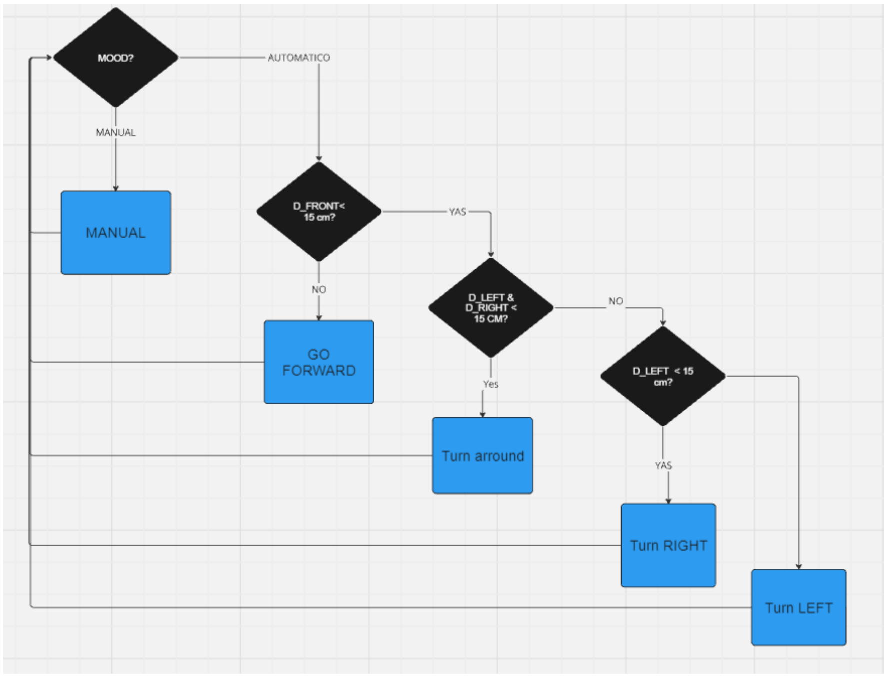


Figure 9 Flowchart

3.2.1.2. Programable Logic

In the PL the following controllers are defined.

- PWM generator & direction control
- Distance sensor control
- LCD control

Each of them is explained in their corresponded function section and the code is shown in the appendix [I].

3.2.2. Chassis

To begin with the construction of the vehicle we will have to define the physical structure of it. Since designing the chassis is not the main goal, we decided to look for a pre-built chassis. There are many of them, different sizes and shapes, different kinds and number of wills, different materials, etc. Taking in mind that it has to be easily reproduced at college laboratories the size has to be pretty manageable therefore compact, nevertheless has to be big enough to fit all the other components.

It has been chosen as a simple kit shown in the Figure 10 due its simplicity and price. It has three wills which only the two on the sides have motors, the third one positioned in the front does not have a motor but has rotational freedom. All of it is subjected together on a simple polyacrylate surface with multiple screw holes.

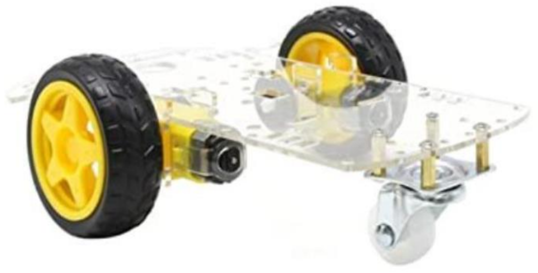


Figure 10 Chassis

3.2.3. Mode

The vehicle has two modes to run. It is capable of running autonomously using the distance sensors to position itself and follow an algorithm described in the processor to avoid obstacles. Otherwise, there is a manual mode which the user will control de motors with a controller. The pynq-z2 has multiple switches incorporated that we will use to determine the mode that we want to run the vehicle.

3.2.4. Movement

In this section we will discuss which motors are going to be used and what hardware is needed to use them. The control of the motors is also described.

3.2.4.1. Motors

Coming as a kit with the chassis we will use the motors DC TT 6V/200RPM that can run from 3 V to 6 V. We will power them with 5V due the PYNQ-Z2 has a 5V output. These motors are equipped with a 1:48 gearbox, which increases the motor's output torque and reduces its speed. The motor, after the gearbox, is capable of 200 RPM at 5 V with our 6.5 cm diameter, the theoretical maximum speed is calculated as follows in Equation 2

$$2\pi \frac{6.8}{2} cm = 20.42cm \qquad \textit{Equation 1}$$

$$200rpm \times \frac{20.42cm}{rev} \times \frac{1}{60} s^{-1} = 0.681m\, s^{-1} \qquad \textit{Equation 2}$$

However, this speed is theoretical and assumes no load on the motor

The maximum torque it has also has to be in consideration. These motors have a torque of 0.8 kg cm. Giving that the center of gravity of the vehicle is approximately 5 cm away from the wheel axis, the load capacity can be calculated as follows in Equation 4

$$Torque = Force \times Distance \quad \textit{Equation 3}$$

$$Force = \frac{1\,kg\,cm}{2cm} = 0.5\,kg \quad \textit{Equation 4}$$

This means that each motor can support a load of 0.5 kg at this distance. Since we have two motors, we can carry up to 1 kg which is higher than the approximate 0.5 kg that weights our vehicle.

Given these speed and torque capacities, it can be determined that these motors, shown in the Figure 11, are a good fit for this application.

Figure 11 Motos DC TT 6V/200RPM

3.2.4.2. L293D

A Half-H bridge driver is a crucial component for efficiently controlling both the direction and speed of motors. In this project, the L293D IC ,shown in Figure 12, has been chosen for this purpose due to its reliability and effectiveness. In Table 2 there is a list of L293D signals and pin functions. This device can drive up to two different DC motors

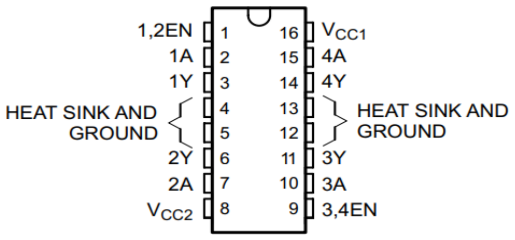


Figure 12 L293D

Pin	Direction	Description
1.2 EN 3.4 EN	INPUT	Enable driver channels. We will use to regulate the speed of the motors with a PWM signal. The speed will fluctuate depending on the duty cycle.
1A,2A 3A,4A	INPUT	With these pins we will control the direction of the motors
1Y,2Y,3Y,4Y	OUTPUT	It outputs the PWM signal to control the motors.

1A / 3A	2A /4A	Direction
0	0	No Movement
0	1	Backwards
1	0	Frontward
1	1	No Movement

Table 2 L293D Pin description

3.2.4.3. PWM generator & direction control IP

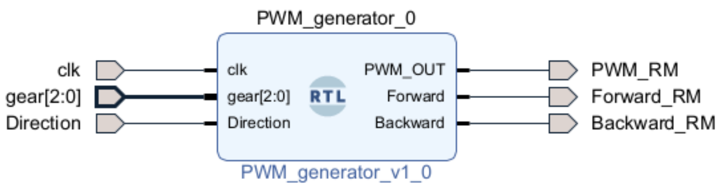


Figure 13 PWM_generator IP

This block, shown in Figure 13 , generates a PWM signal with a frequency of 1KHz which duty circle depends on the gear input. The gear input has 3 bits to determine the 5 different velocities of the motors. The input directions determine the direction of rotation of the motors setting the outputs Forward or Backward to 1.

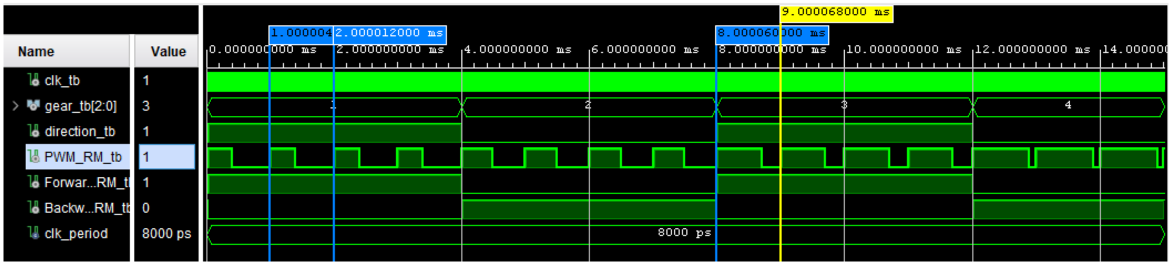


Figure 14 PWM_generator IP simulation

In the Figure 14 we can see a simulation of the block. The gear and direction inputs that have been set in the testbench are visible on the second and third lines. The following tree signals are the outputs, it is easy to see that the Forward signal is equal to the input Direction and the Backward signal is always opposed. The PWM signal spotted on the fourth signal always has 1 KHz and its duty cycle increases as the gear input increases. The code of this IP is shown in appendix [1].

3.2.5. Distance Sensing

Naw we will talk about the sensors used. The distance sensors are responsible for capturing physical distance data, which allows the control to make the actuators act accordingly.

3.2.5.1. Sensor Alterantives

There are several types of distance sensors, each using different technologies to measure distance. The most common types include:

Ultrasonic Sensors: Those use high-frequency sound waves to measure distance. The ultrasonic wave emitted travels through the air at the speed of the sound. This type of sensors is widely used in robotics for obstacle detection, in automotive systems and in industrial settings. We can see an example in Figure 15.

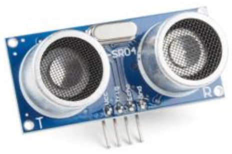


Figure 15 HC-SR04 Distance sensor

Infrared (IR) Sensors: These sensors work as ultrasonic sensors but instead of using sound waves, they use laser beams that travel at speed of light. This type of sensor is used in surveying, mapping. and industrial automation. We can see an example in Figure 16.

Figure 16 IR sensor

Capacitive Sensors: Capacitive Sensors measure changes in capacitance caused by the proximity of an object. As an object gets closer to the sensor, the capacitance changes, allowing the sensor to determine the distance. This type of sensor is used in touchscreens, proximity detection and liquid level measurement. We can see an example of if Figure 17.

Figure 17 Capacitive sensor

The ultrasonic distance sensor HC-SR04, shown in Figure 15, has been chosen for this project due its versatility, accurate measurements, price and easy implementation. It has a range from 2 cm to 400 cm with a resolution of 0.3 cm. It sends a burst of 40 KHz every 20ms. In the Table 3 a pin description is shown and the Figure 18 shows the form of the expected signals.

Pin	Direction	Description
VCC	SUPPLY	Power supply. It needs 5 V to work properly.
Trigger	INPUT	Signal to enable the ultrasonic wave output. Requires a 10 us pulse every 20 ms.
Echo	OUTPUT	Once the burst is sent the ECHO pin will automatically go HIGH. This pin will remain HIGH until the the burst hits the sensor again
GND	GND	GND

Table 3 HC-SR04 Pin description

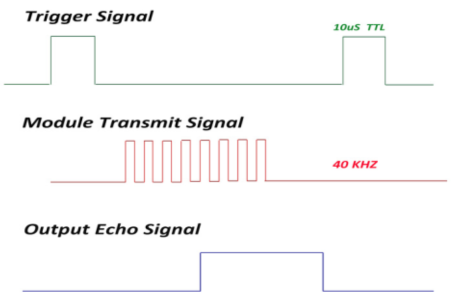


Figure 18 HC-SR04 waveform

3.2.5.2. Voltage divider

Due the pynq-z2 can only hold 3.3 V on its inputs and the HC-SR04 outputs 5 V we need a voltage divider to reduce the output sensor voltage. It needs two resistances put in series, ideally on resistance of 283 ohm and 550 ohm. The output of the sensor is connected to the 283 ohm resistor, the GND to the other end of the 550 ohm resistor. The input of the PYNQ-2 is connected where the two resistors are connected.

3.2.5.3. Distance sensor control IP

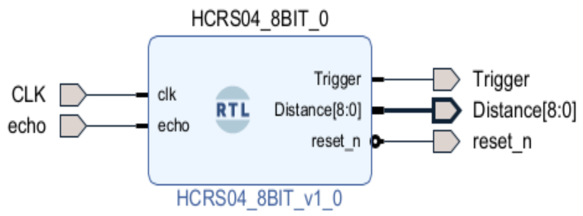


Figure 19 HC-SR04 IP

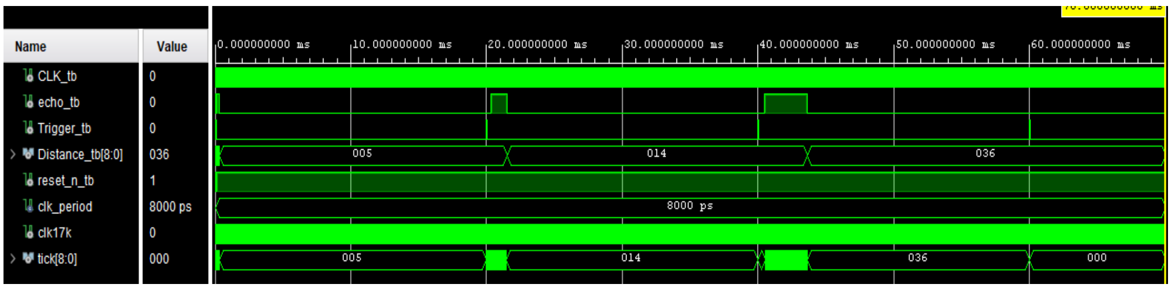


Figure 20 HCRS04 Simulation

This IP shown in *Figure 19* is in charge of generating the Trigger signal, which is a pulse of 10 us every 20 ms. This signal can be seen in *Figure 20* on the third signal. The echo signal above is simulated and represents a specific distance. A 17 KHz CLK is generated due to its period of 58.823 us, which is the time that the waves traveling at the speed of sound take to travel 2 cm. It is important to bear in mind that the wave travels the doble distance that the actual distance from the object measure dew it has to go and return. Therefore, when counting pulses of the clk15 signal, we will be directly counting the distance in cm. We can see this signal at the bottom of the simulation clearly counting only when the echo signal is high which is the time that the wave burst is traveling. Eventually we have the signal distance out seen on the middle of the simulation which is represented by 9 bits. *The code of this IP is shown in appendix [1].*

3.2.6. Interface

Since the sensors will be reading distance constantly it is useful to show the data to the user. There are different options available to represent characters. One option is a 7-segment display however we would need too many to represent all the data. The best option is an LCD display specifically a 20x4 due it has enough lines to represent multiple distances, it is shown in Figure 21.

3.2.6.1. LCD

LCD displays are commonly used in embedded systems, have a set of pins used for communication, power and control. On Table 4 the pins of a 20x4 LCD with are explained.

PIN	Description
Vss	Ground
Vdd	power supply which it has to be connected to 5 V
Vo	contrast adjustment
RS	It is the register select, determines whether the data is interpreted as a command or data. If it is set to 0 is interpreted as an instruction if it is set to 1 is interpreted as a data register.
RW	selects between reading from or writing. If it is set to 0 it is writing if it is set to 1 it is reading.
E	is used to latch data. A falling edge on this pin tells the LCD to process the data on the data pins (D0-D7).
D0-D7	Data pins used to send the data (8-bit parallel).
LED+	powers the LED backlight
LED-	Connect this to ground to complete the backlight circuit

Table 4 LCD pin description

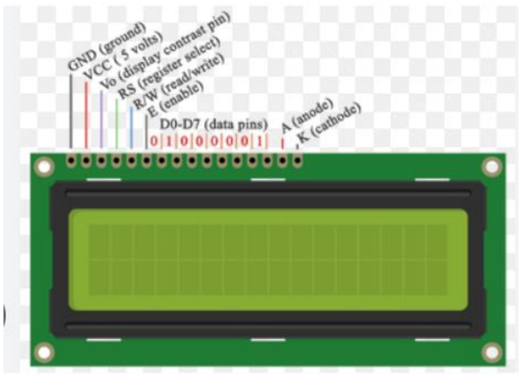


Figure 21 LCD

3.2.6.2. LCD control IP

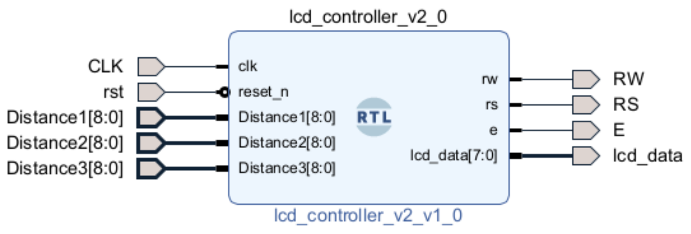


Figure 22 LCD controller IP

This IP shown in Figure 22 sets up the LCD and translates the 9-bit distance in cm to the 8 data bits that the HD44780 driver needs to represent the characters.

When initializing an LCD, a series of fixed steps and timings have to be followed in order to do it correctly. They are described in the manufacturer's datasheet. In this case, the LCD Control Unit follows this routine:

1- After power on, wait 50 ms
2- For 10 µs activate LCD_EN and configure 2 lines display
3- Wait for 50 µs
4- For 10 µs activate LCD_EN and configure display on, cursor off, no blink
5- Wait for 50 µs with LCD_EN deactivated
6- For 10 µs activate LCD_EN and clear display
7- Wait 2 ms so the display finished clearing
8- For 10 µs activate LCD_EN and configure increment mode, shift off
9- Wait for 60 µs with LCD_EN deactivated

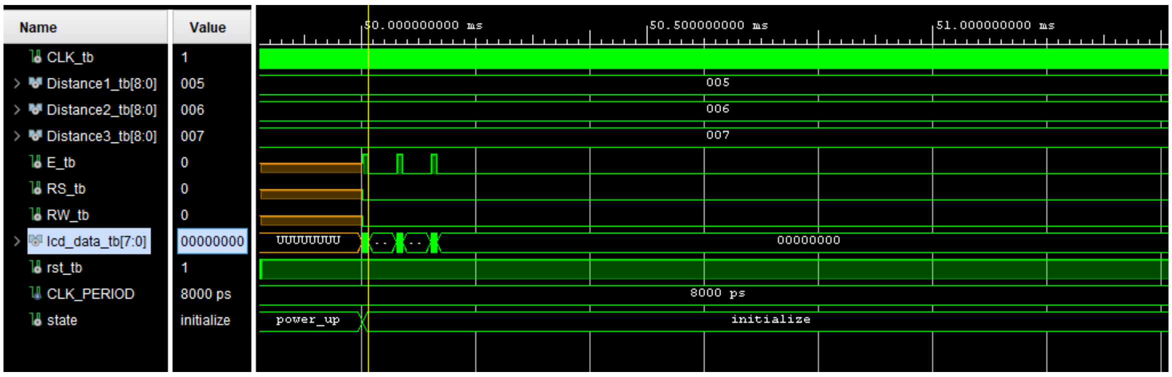


Figure 23 Simulation intialization LCD

In Figure 23 it is shown the process of initialization of the LCD, we can se de state signal on the bottom how changes from power_up state to initialization state and wen this one begins the lcd_data starts sending the binary codes to configure the LCD. The E signal only enables the output lcd_data when there is something to send.

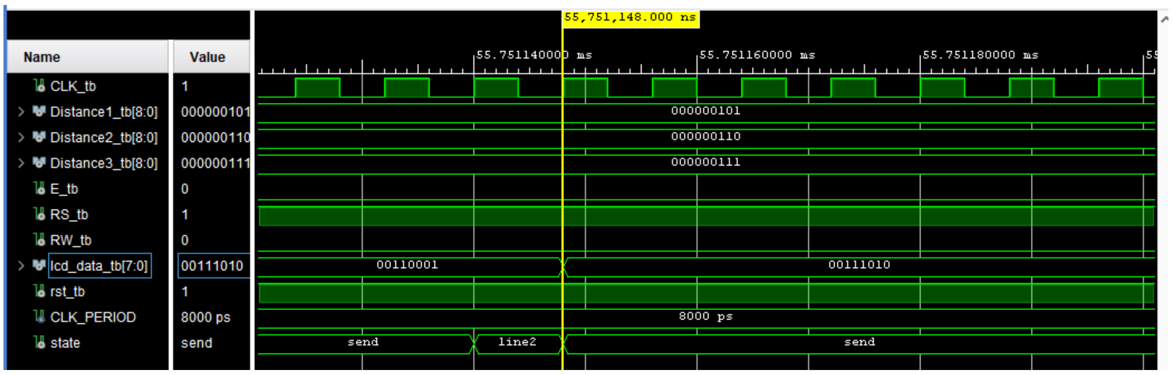


Figure 24 Simulation LCD send data

In Figure 24 , we can see how the state value changes to determine which line will be written. In this case, we observe line 2. Next, the data is sent through lcd_data

3.2.7. Communication

In the manual mode the vehicle is controlled by the user through a radio controller. The radio controller (TX) has only 4 buttons corresponding to the 4 directions that the vehicle can take (forward, backward, left and right). Attached to the vehicle there is a receptor (RX) that is connected to the PYNQ-Z2 which reads the signals transmitted and acts in consequence.

3.2.7.1. TX

To transmit the commands from the radio controller we use the MC145026 to encode the input signals into a serial data stream. The serial data inputs to the 433MHz module that is in charge to transmit the data to the RX attached to the vehicle with RF. The 433 MHz has to be powered with 5 V, it has also a GND pin and an antenna extender.

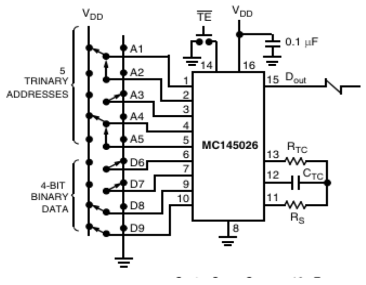


Figure 25 MC145026

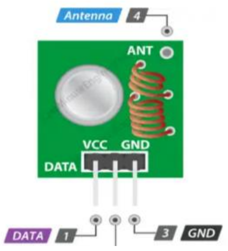


Figure 26 Transmitter 433 Mhz module

3.2.7.2. RX

On the receiving end a 433 MHz module receiver captures the RF and outputs the serial data into the decoder, such as the MC145027 used to decode the received data into parallel. This data inputs the PYNQ-Z2. In figure Figure 28 there is the circuit recommendation and used , in Figure 27 the 433 MHz module is shown.

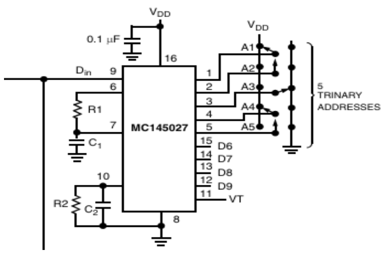


Figure 28 MC145027

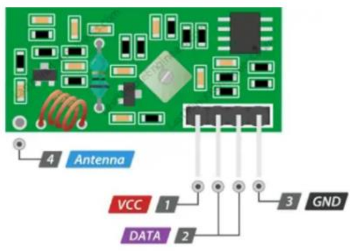


Figure 27 Receiver 433 MHz module

3.3. Mechanics

Once we have chosen the components, we will proceed to put all of the project together. Beside the TX part of the communications all the components are built up on the chassis, which is the main structural component. Four columns of 3.5 cm screwed to the polyacrylate base of the chassis holds the PYNQ-Z2 elevated so the battery can go between the base and the board.

The LCD is placed at the back of the vehicle, held only by the cables of its pins connected to the PMOD ports on the board. Moreover, a PCB is designed to hold all the ICs and components needed. The L293D, MC145027, and voltage divider are soldered to the PCB, along with all the resistors and capacitors needed to operate the ICs properly. It also has screw connectors for the battery and the motors. There are three connectors for the three HC-SR04 sensors. One sensor points towards the front, while the other two are angled slightly towards the front, pointing to the sides. Furthermore, a connector for the 433 MHz receiver module is also soldered.

The PCB connects to the PYNQ-Z2 through the Arduino pins so it holds as a board shield.

Finally, a small PCB out of the chassis holds the radio control, where four buttons, the MC145026, and the 433 MHz transmitter are soldered.

3.4. PCB

The PCB has been designed using Kicad7 (https://www.kicad.org/) and is used to hold the peripheric is quite simple, it only has two capes so it can be easily reproduced on collage laboratories. On Figure 29 the layout is shown where the blew traces are the bottom plane and the red traces are the top plane. On the left sight the three distance sensors are connected to their voltages dividers. On the center we have the L293D IC and on its right the MC145027 IC. On the right top and bottom there are the Arduino pins so the shield can be easily inserted to the PYNQ-Z2. The schematic can be checked on the Appendix [II].

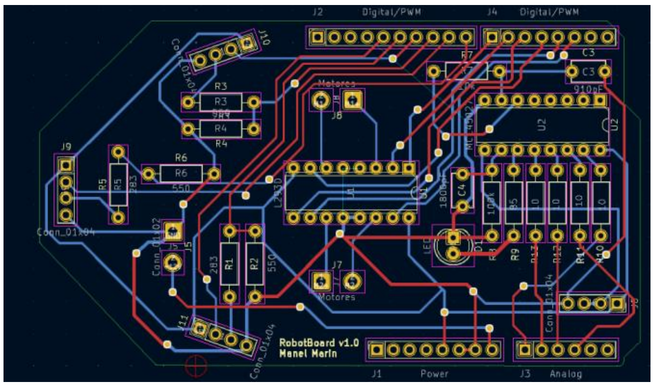


Figure 29 PCB

3.5. Implementation

Eventually once we have all the components selected and individually running, we put them all together, in Figure 30 is shown the final design. In this same figure the blocks are color distributed by the function they are implementing. The color concordats with Figure 7 where all the functions are described. The internal connections of each module are symbolized by the arrows and what kind of connection and protocol is.

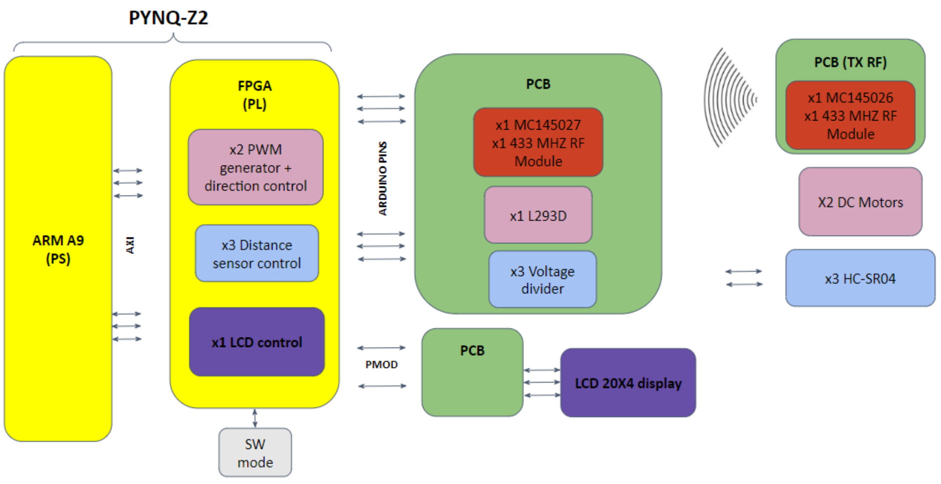


Figure 30 Final design

Next, we will explain how Zynq-7000 SoC, represented in yellow *Figure 30*, is synthetized and how it is connected to the other components, *Figure 31* shows the bloc implementation. From now on we will be talking about the *Figure 31*. It is needed to clarify that all the ports used are connected to the PL.

The PS is in the yellow square outputs the 125 MHz CLK signal for all the peripherals, it is also connected to the AXI managing blocs which are in the orange rectangle. The mode switch is on the grey rectangle, since the switches on the PYNQ-Z2 are connected to the PL we need an AXI_GPIO bloc to send the input to the PS , this IP is given by the Xillinx we only need to configure it as an input and the bites used. The 4 bit input of the RX control, which are on the red square, are connected to the Arduino ports mining that it also need an AXI_GPIO bloc as the mode switch. The three distance sensor controllers are on the blue rectangle, the echo inputs and the trigger outputs are connected to the Arduino ports as well but since they are connected straight to the PL they do not need an AXI_GPIO bloc. The distance sensor output of 8 bits of each controller goes to an AXI_GPIO bloc to eventually be used in the PS, likewise they go to the LCD controller. The LCD controller is not connected to the PS only receives data from the distance sensor IP and outputs the data signals through the PMOD ports.

The last bloc to talk about is the motors controller situated on the green rectangle, their outputs go straight to the Arduino connectors. This IP incorporates the AXI connection, when we generate the IP we specify to Vivado that AXI is required and it automatically generates the AXI connections in VHDL. It is needed to manually write the connections between the inputs of the motor controller and the AXI registers so the data can be exchanged. In this case the gear bits and the direction bit are connected to 4 bits of the first AXI register, being the direction the most significant bit. From the PS will be needed to change the bits individually.

All the port connections and assignments are defined in the constrains file which is attached on the Appendix[VI].

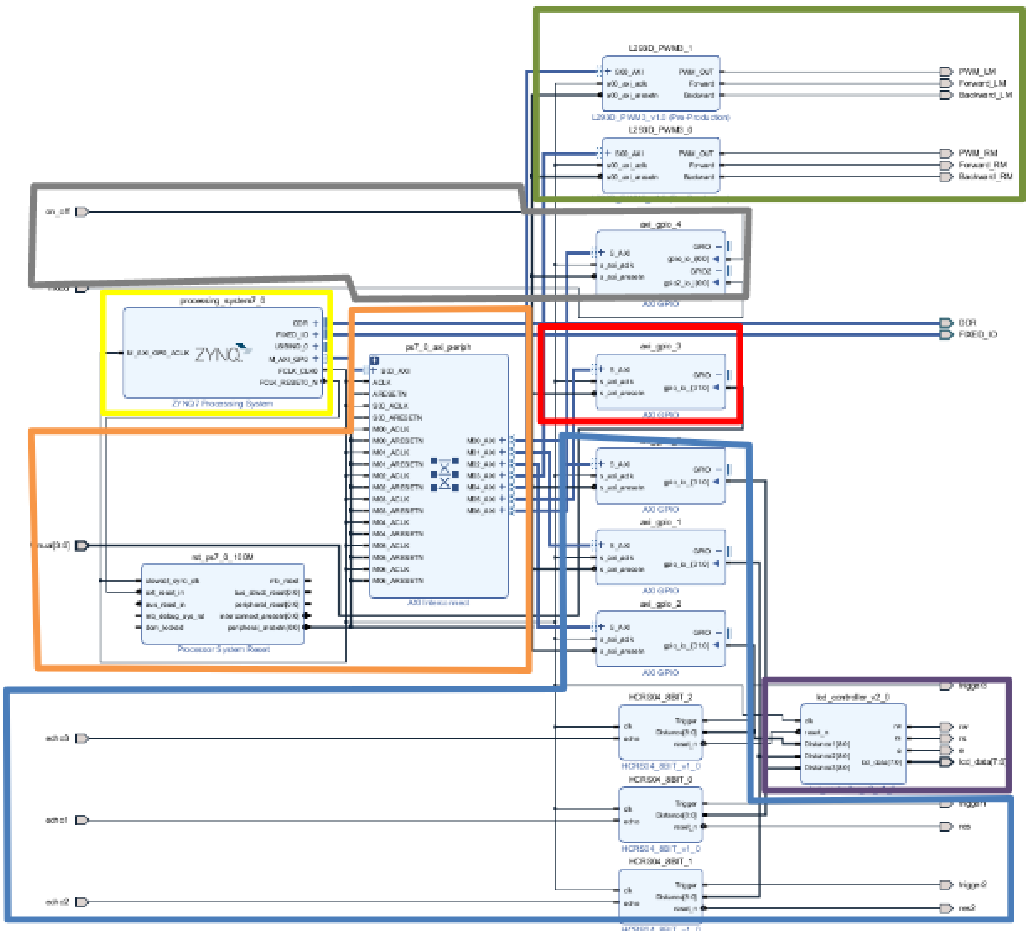


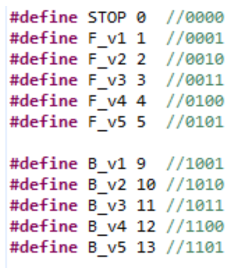
Figure 31 SoC implementation

After implementing all the Hardware with Vivdo is time to proceed with the software implementation with Vitis. From Vivado is exported the hardware to Vitis and automatically libraries to be able to use the hardware implemented are created. On table [] the most important libraries are described.

Library	**Description**
Xgpio.h	It allows you to control GPIO pins, read input states, and drive outputs.
Xparameters.h	This file contains definitions for hardware parameters, including addresses and configurations for peripherals, generated during the hardware design process in Vivado
Xscugic.h	It allows you to manage and configure interrupts for your application.

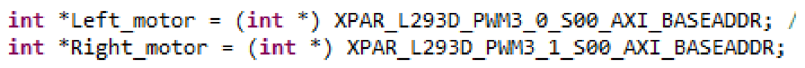
Table 5 Libraries hardware definition

First of all we define all the hardware parameters for the AXI_GPIO pins that we obtain from the libraries. Also we define the number in integer that the gear and direction are in binary for every combination of velocity and direction available, the code is shown in *Figure 32*.Next we create a pointer to the Motor IPs registers where the gear and directions are steed shown in *Figure 33*. For the manual mode we also define the numbers that represent in binary for each instruction of movement shown in *Figure 34*.



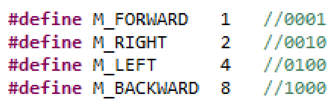
```
#define STOP 0  //0000
#define F_v1 1  //0001
#define F_v2 2  //0010
#define F_v3 3  //0011
#define F_v4 4  //0100
#define F_v5 5  //0101

#define B_v1 9  //1001
#define B_v2 10 //1010
#define B_v3 11 //1011
#define B_v4 12 //1100
#define B_v5 13 //1101
```

Figure 32 Software definition of Gear and Direction



```
int *Left_motor = (int *) XPAR_L293D_PWM3_0_S00_AXI_BASEADDR; /
int *Right_motor = (int *) XPAR_L293D_PWM3_1_S00_AXI_BASEADDR;
```

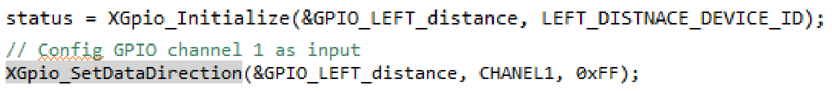
Figure 33 Pointer to the Motors IP



```
#define M_FORWARD   1   //0001
#define M_RIGHT     2   //0010
#define M_LEFT      4   //0100
#define M_BACKWARD  8   //1000
```

Figure 34 Controller commands Software definition

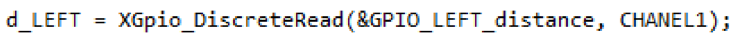
Next step on the code the GPIOS are initialized and configured shown in *Figure 35*.



```
status = XGpio_Initialize(&GPIO_LEFT_distance, LEFT_DISTNACE_DEVICE_ID);
// Config GPIO channel 1 as input
XGpio_SetDataDirection(&GPIO_LEFT_distance, CHANEL1, 0xFF);
```

Figure 35 GPIO Software initialization

Eventually we start the algorithm described on 3.2.1.1 Processing System. To read from the GPIO we use the function shown in *Figure 36*. To set a velocity and direction to a motor IP we just have to give value to the pointer of the motor IP.



```
d_LEFT = XGpio_DiscreteRead(&GPIO_LEFT_distance, CHANEL1);
```

Figure 36 GPIO Read function

All the code in C is attached to the Appendix's [].

3.6. Bat

In this section we will go over the power source. Bearing in mind the fact that this project has to be reproduced on a collage laboratory we will take in consideration the size, the duration and the easiness to use them. The board is powered with 12 V and it is able to output the 5 V needed for the other peripherals. In total the vehicle has a consumption of 800 mA approximately. On Table 6 we can see a breakdown of all the consumptions.

Component	**Consumption (mA)**
PYNQ-Z2	160
L293D	10
MC145027	0.4
HC-SR04	10
LCD	25

Table 6 Power consumption

Som batteries alternatives are LiPo due they are rechargeable and there are many options with different properties. These batteries are built with connections of multiple cells of 3.7 V so we would need 3 cells meaning that a balanced charger would be needed.

Another option are the Alkaline AA batteries which are not rechargeable. Each of them has 1.5 V and approximately 2500 mAh so we would need 8 of them.

We will assume that most of the development process there will be a power supply to do some test and only on the final stages a battery will be required. It will be assumed that around 3 h the vehicle will be running at its full capabilities on a single course of the subjects, therefore we will need 2400 mAh.

Although the alkaline AA batteries are not rechargeable, they are easy to use an only once per course would need to be changed. True to admit that if multiple courses are being runed at the same time should be considered the LiPo batteries.

3.7. Construction

The last step consists of building the final vehicle. First of all, we build the chassis screwing the motors, the columns and on top of those the PYNQ-Z2. We connect the LCD at the back of the chassis to the PMOD ports of the PYNW-Z2.We set the battery under the board and fix it with straps. After soldering the PCB, we connect it to the Arduino pins. Before proceeding to the computer, we screw the motors and the battery cables to the screwable pins of the PCB. The radio control TR is soldered on a separate PCB. Now on the computer we just have to upload to the board the software described on Vitis that runs on the hardware described on Vivado. Finaly everything works accordingly and the vehicle then is finally done running autonomously or by the radio control. Som pictures of the final robot can be seen on

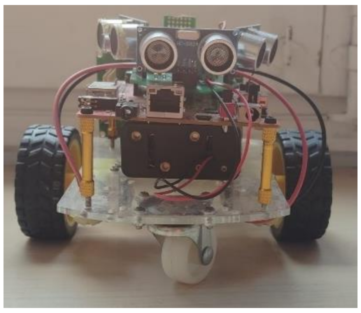


Figure 38 Vehicle front

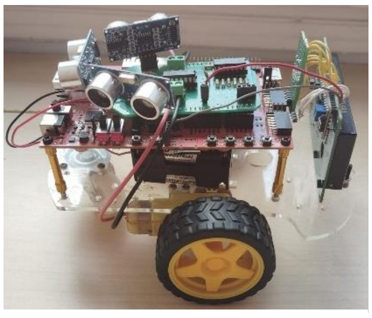


Figure 37 Vehicle lateral

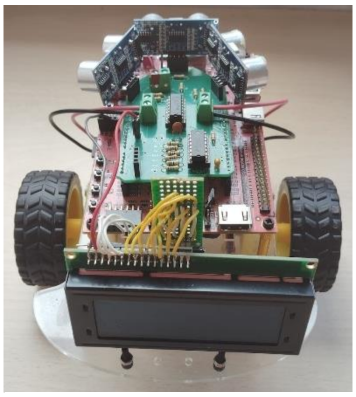


Figure 40 Back vehicle

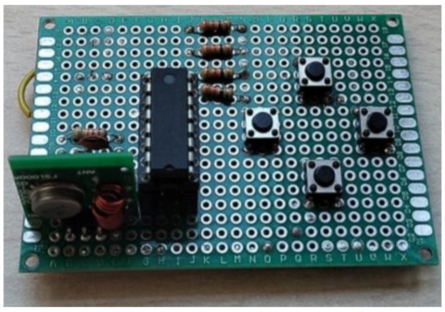


Figure 39 RT controller

CHAPTER 4

Results

After finally bullied the vehicle, this section is dedicated to go over the tests and the results to validate the accomplishment of the project, not only of the final result but also of each part of the vehicle.

4.1 Experiments and Tests

The experiments have been designed to test the different modules along the construction of the vehicle to eventually do a final test that put all the modules together. These tests check all the functions described on the Figure 7 alongside their performance.

The first module that has been evaluated is the distance sensing. After implementing the IP, a short VHDL program was developed to test distance measurement by illuminating the LEDs on the PYNQ-Z2, based on different distance ranges. Also the voltage divider was built on a breadboard. As a result of the tests, we observed how the 4 LEDs turned on and off based on the distance of an object as it was moved closer or farther away. We can see in Figure 41 only the first LED due the object was close to the sensor.

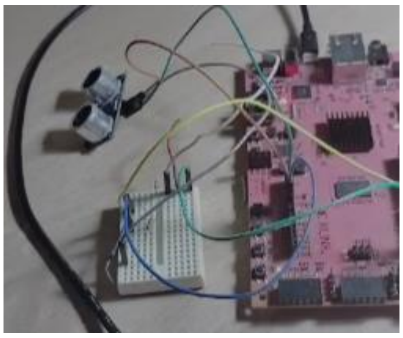


Figure 41 Distance sensing test

Next experiment was to test the Interface and see if the LCD worked properly, to do so the distance module already tested was used to represent the distance straight to the LCD. This test only uploaded the IP using the Vivado tool. In figure *Figure 42* can be seen that the LCD display sows the distance in cm. The LCD printed the distance perfectly and it changed the distance every time

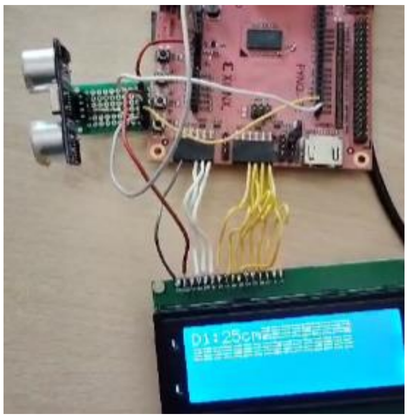


Figure 42 Interface test

The following test carried out was the Movement module. The buttons of the board were connected to the inputs of the L293D gear and direction so the velocity and direction of the motors was cheeked. The L293D IC was connected to a breadboard and the PWM to an oscilloscope, in *Figure 43* the set up is shown. As a result, the motors moved depending on the buttons pressed likewise the duty cycle of the PWM changed on the oscilloscope. The duty cycles followed the ones calculated for the different velocities controlled by the input gear.

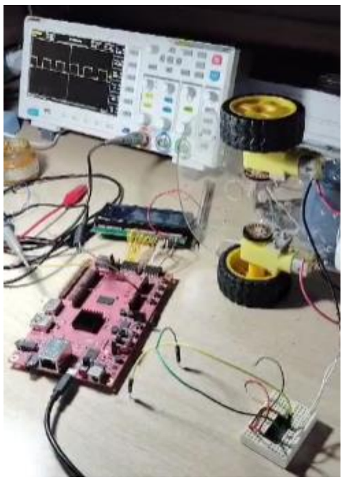


Figure 43 Movement test

Next module to test is the Communication. We soldered the TX on a test board and on the other end the RX was connected to ha breadboard as it is shown in Figure 44. LEDs were connected to the data outputs of the MC145027 to see if the corresponded signal was received. On a first stage the 433 MHz modules were not used and the output of the MC145026 was connected with a single cable to the input of the MC145027. In this case the data was properly sent and received. The second test the 433 MHz modules were connected and the sender and the receiver were not physical connected. In this case we had some problems, we were not able to receive any data. It is not known why the 433 MHz modules did not work. An extra test was performed with two Arduino UNO, an easy code to send a simple text from one Arduino to another with the 433 MHz modules. It did not work either. The supposition is that the modules purchased were not working and dude lag of time the wireless implementation has not been achieved. However, the manual mod stills implemented with a single cable between the vehicle and the controller.

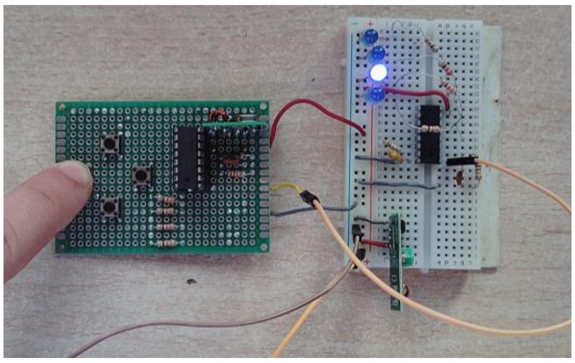


Figure 44 Communication test

After testing all the modules individually, it is time to put all of them together and build the testing vehicle shown in Figure 45. We soldered all the components in a testing board to see how all the modules interact with each other. To test that the movement module acts by consequence of what is read by the sensors we had to test the software application as well. An intermediate test was realized to check the AXI communication. A short software program was uploaded to the board to read the distance on Vitis Serial Terminal which meant that the Distance Sensing IP was correctly sending the data to the PS. The test vehicle was eventually used to verify that its movement responded based on the measured distance. The same way we tested that the motors responded based on the commands from the controller.

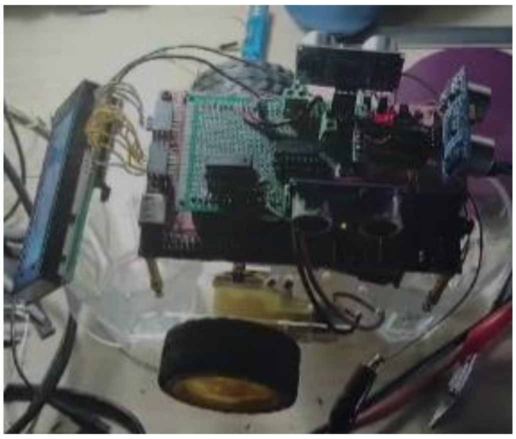


Figure 45 Testing vehicle

The last test is the autonomous function, at this point we already have the final version of the vehicle with its PCB. A small circuit shown in Figure 46 is built to check if it avoids the obstacles.

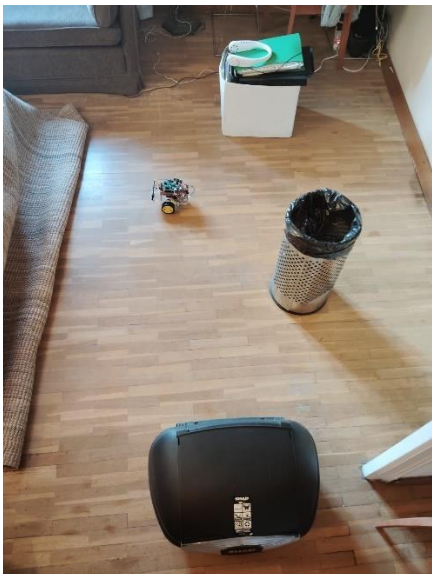


Figure 46 Autonomy test

After all this tests and experiments the vehicle performed all its functions as described beside the fact that the manual mode was not wireless.

4.3 Limitations

There exist few limitations and upgrading that can be applied to this project.

First of all is already been explained that the controller needs a wire to communicate with the vehicle so the manual control is not wireless as it was designed for. It is believed that the 433 MHz modules were not working since multiple tests were done with them and none of them worked. It is proposed to replace them with the same model and if it stills not working try other models of radio control like NRF24L01 although a microcontroller would be needed on the TX and other protocols would be needed to implement making it more complex.

In manual mode, only one command can be sent at a time. If two buttons on the controller are pressed simultaneously, the vehicle won't respond because the corresponding binary code is not toked in consideration in the software code.

The motors emit a small sound due the PWM frequency is on the audible range so it would be good practice to elevate the frequency to 20 kHz which is the limit of the audible range.

The LCD is hanging only with the cables and is not properly subjected to the chassis. It has accomplished its functions however is not the most practical and any cable could disconnect any time. It would be good to design a small PCB that makes the connection between the PYNQ-Z2 and the LCD safer.

CHAPTER

5

Sustainability Analysis and Ethical Implications

This chapter will explain the different aspects of the project's development, detailing how this work impacts the environment, economy, and society.

5.1. Cost of development

This project involves both hardware and software development, taking into account the mechanical and electronic fabrication. Therefore, the environmental impact of this project relates to the consumption of electronic devices such as the PC, power supplies, and soldering stations. All construction and development have been done at home which mean that there has not been any waste of personal transportation. Additionally, consideration must be given to all the orders for the different components.

The project has been carried out on a PC with a power consumption of 75W. Considering that the TFG spans a total of 520 hours and that the average carbon footprint in Spain is 72g CO2/kWh, the CO2 consumption from the PC is as follows:

$$\text{Co2PC} = 75\text{ W} * \frac{1kW}{1000W} * 520h * \frac{72\ gCo2}{kWh} = 2808\text{ gCo2}$$

The vehicle has a consumption of approximately 9.5 W and during all the development and test we approximate that we have used the vehicle 150 hours. It is taken in consideration that not all the components have been used at the same time all the hours which mean that not all the time the vehicle has consumed the same.

$$\text{Co2vehicle} = 9.5\text{W} * \frac{1kW}{1000W} * 150h * \frac{72\ gCo2}{kWh} = 102.6\ gCo2$$

Nearly all the components have been ordered from China, and the transportation of these packages has been estimated. We assume that the transportation is done via air freight, and the total weight of the components is approximately 2 kg, which means the CO2 emissions are around 3000 g CO2.

The total CO2 emissions for carrying out this project is 5910 gCO2.

To begin with the economic cost, we will calculate the electrical consumption.

$$\text{Electric consumption } = 9.5\text{W} * \frac{1kW}{1000W} * 150h + 75\text{W} * \frac{1kW}{1000W} * 520h = 40.43\ kW/h$$

Taking in consideration that average price of the energy this year is 0.0664 €/kWh the total cost of the electricity is 2.69€ .

In Table 7, there is a list of all the resources used in this project. The program used in the development of the project has been the Xillinx tools Vivado and Vitis, which is a free.

Product	**Price**	**Units**
PYNQ-Z2	216 €	1
Chassis +x2 motors	20 €	1
HC-SR04	2€	3
LCD 20X4	20 €	1
L293D	0.5€	1
MC145026	0.5€	1
MC145027	0.5€	1
Module receptor and emitter 433 MHz	2€	1
Resistances, capacitors, buttons	2€	1
PCB	2€	2
TOTAL	**265.5€**	

Table 7 Material budget

In *Table 8*,there is the number of hours the engineer has invested in this project, multiplied by the cost of those hours.

Hours invested	**Price per hour**	**Total**
26 weeks – 20h/week	40 € /h	15600€

Table 8 Hours invested budget

In Table 9, there is the final budget for this project. In addition to the fields calculated in Table and Table, other costs have been included, as well as taxes.

BUDGET		
Materials		**265.5€**
Hurs invested		15600€
Electricity		2.69€
SUBTOTAL 1		**15865.5€**
Indirect costs (10%)	10% ·15865.5€ =	1586.55€
Material amortization (5%)	5% · 265.5 =	13.27€
SUBTOTAL 2		**17465.32€**
IVA (21%)	21% · 17465.32€ =	3667.71€
TOTAL		**21133.04€**

Table 9 Final project budget

It is important to note that the budget is unrealistic and that if the project had to be repeated again from scratch, the number of hours would be much lower due to the knowledge and experience acquired during its making.

5.2. Execution of the project

As mentioned earlier, this project has been developed in software and hardware. It only makes sense talking about longevity of the hardware. For this robot, electronic components that can be recycled should be used, in addition to a development board optimized for low energy consumption, such as the PYNQ-Z2 board utilized in the vehicle that serves as the basis for the project.

Regarding to viability, everything developed in this project can function with other distance sensors, other SoC that integrates PS and PL, other kind of displays and other mechanics of movements. As this work has been approached, the requirements of the project are to facilitate the learning of subjects like digital design to the students.

5.3. Risks

As with any technological project, there are risks that must be evaluated in order to prevent them if possible.

From an environmental perspective, the risks include the impact of producing electronic components. The manufacturing of sensors and circuit boards can generate waste and pollution if not managed properly. Additionally, the disposal of these components at the end of their lifecycle poses an environmental risk. On the other hand, the environmental limitations focus on the eco-friendly design of the robot. Implementing sustainable production practices can be challenging due to the available resources and current technologies. One aspect to consider is the use of alkaline batteries. For extended operation of the vehicle, rechargeable LiPo batteries should be used. This requires having a set of chargers and procedures in place to maintain the batteries in proper condition.

From an economic perspective, the risks are tied to the future costs of upgrades and maintenance. While the distance sensor and the development board are a cost-effective solution, technological advancements and the need for future improvements may require additional investments. Likewise, the costs associated with the maintenance and replacement of the robot's components can be unpredictable and may fluctuate over time. and integrating components, which could restrict the use of more advanced technologies or higher-quality materials.

From a social perspective, one of the risks is student disinterest. Since the technology is quite new and the tools can be difficult to understand and master, students may lose interest.

CHAPTER

6

Conclusions and Future Work

7.1 Conclusions

In this project, the development of a small, autonomous vehicle using an FPGA-based board was successfully achieved. Fulfilling the goal of creating a modular and comprehensive educational resource for university courses demonstrating their potential in modern embedded systems. Using VHDL for hardware design and the AXI interface for communication between the Programmable Logic (PL) and Processing System (PS), a system was created to handle tasks such as motor control, sensor management, and user interaction via an LCD and a controller.

The project achieved its primary goals of designing a functional robot capable of both autonomous obstacle avoidance and manual control. The real-time processing capabilities of the FPGA, combined with the computational power of the SoC, enabled efficient execution of the obstacle avoidance algorithm and ensured reliable sensor data processing. However, while the manual mode was successfully implemented, the intended wireless communication between the controller and the robot could not be achieved. As a result, manual control was established using a wired connection, ensuring functionality but limiting the intended wireless flexibility.

Despite this limitation, the project provides valuable insights into the practical applications of FPGA and SoC technologies. It also serves as a useful educational tool, offering students a hands-on understanding of hardware-software integration and real-time, high-performance embedded systems. The project contributes to the university's aim of enhancing the learning experience in digital design and embedded systems subjects, particularly in illustrating the complexities of communication protocols in embedded designs.

7.2 Future Directions

The results of this project provide a strong foundation for various potential improvements. One key area for improvement is the implementation of wireless communication between the controller and the robot. While the current setup utilizes a wired connection exploring robust wireless technologies, such as Bluetooth or Wi.

Additionally, future work could focus on refining the obstacle avoidance algorithm by integrating advanced sensor technologies. This would enable the robot to adapt to more complex environments and improve its navigation capabilities.

The modular nature of the system presents opportunities for expansion with additional peripherals, such as robotic arms, GPS modules, or environmental sensors. These additions would not only expand the robot's functionality but also offer students hands-on experience with diverse applications of embedded systems, encouraging a deeper understanding of interdisciplinary integration.

Finally, as an educational tool, future iterations of this project could benefit from developing comprehensive tutorials, documentation, and interactive platforms that engage students more effectively. Incorporating web-based resources could provide students with flexible learning opportunities, making the integration of FPGA and SoC technologies even more accessible.

Bibliography

[1] R. K. "Using GPIO, USB, and HDMI on PYNQ-Z2 Board," *rkblog*, 2024. [Online]. Available: https://rkblog.dev/posts/python/using-gpio-usb-and-hdmi-pynq-z2-board/. [Accessed: Oct. 5, 2024].

[2] Element14, "Path to Programmable III: Training Blog 05 - Inter-Communication Between PS and PL of Zynq SoC," *Element14 Community*, 2024. [Online]. Available: https://community.element14.com/challenges-projects/design-challenges/pathprogrammable3/b/blog/posts/path-to-programmable-iii-training-blog-05-inter-communication-between-ps-and-pl-of-zynq-soc. [Accessed: Oct. 5, 2024].

[3] PYNQ, "PYNQ-Z2 User Manual v1.0," 2024. [Online]. Available: https://dpoauwgwqsy2x.cloudfront.net/Download/pynqz2_user_manual_v1_0.pdf. [Accessed: Oct. 5, 2024].

[4] Element14, "PYNQ-Z2 Workshop - AXI GPIO," *Element14 Community*, 2024. [Online]. Available: https://community.element14.com/members-area/personalblogs/b/blog/posts/pynq-z2-workshop---axi-gpio. [Accessed: Oct. 5, 2024].

[5] Vishay, "LCD016N002BCFHE-T Product Data Sheet," 2024. [Online]. Available: https://www.vishay.com/docs/37484/lcd016n002bcfhet.pdf. [Accessed: Oct. 5, 2024].

[6] Instructables, "Light-Seeking Mobile Robot (BASYS 3) VHDL," 2024. [Online]. Available: https://www.instructables.com/Light-Seeking-Mobile-Robot-BASYS-3-VHDL/. [Accessed: Oct. 5, 2024].

[7] FPGA4Student, "PWM Generator in VHDL," 2017. [Online]. Available: https://www.fpga4student.com/2017/06/pwm-generator-in-vhdl.html. [Accessed: Oct. 5, 2024].

[8] T. S. "L293D Based Arduino Motor Shield," 2024. [Online]. Available: https://5.imimg.com/data5/PX/UK/MY-1833510/l293d-based-arduino-motor-shield.pdf. [Accessed: Oct. 5, 2024].

[9] K. S. "Light Seeking Mobile Robot," *YouTube*, 2024. [Online]. Available: https://www.youtube.com/watch?v=r7FD31icQek&t=213s. [Accessed: Oct. 5, 2024].

[10] NXP, "MC145026 Data Sheet," 2024. [Online]. Available: https://www.nxp.com/docs/en/data-sheet/MC145026.pdf. [Accessed: Oct. 5, 2024].

[11] E. Proyectos, "Control Remoto 433MHz," *Proyectos Electrónica Creativa*, 2014. [Online]. Available: https://proyectoselectronicacreativa.blogspot.com/2014/08/control-remoto-433mhz.html. [Accessed: Oct. 5, 2024].

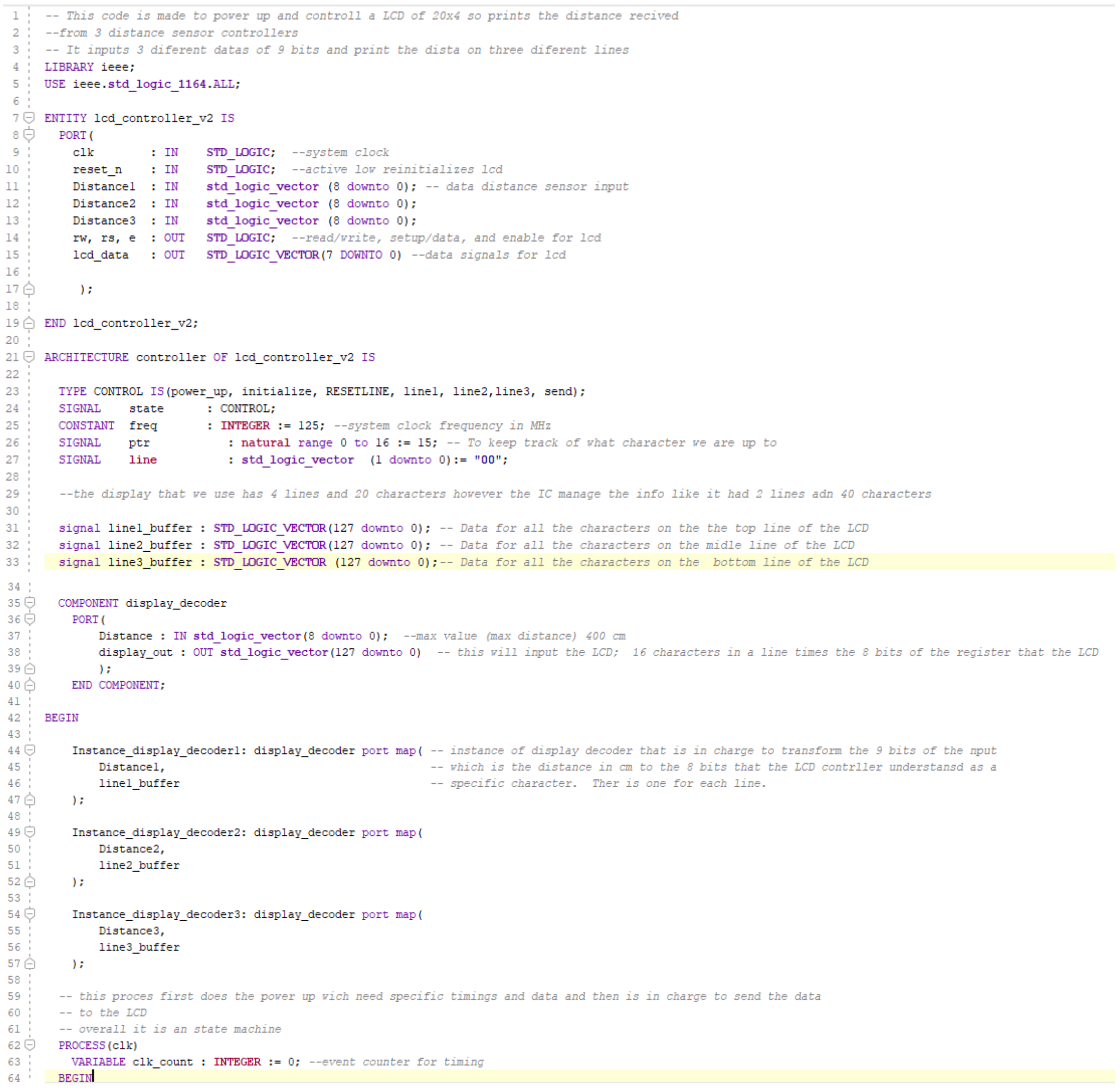
APPENDIX A

An appendix

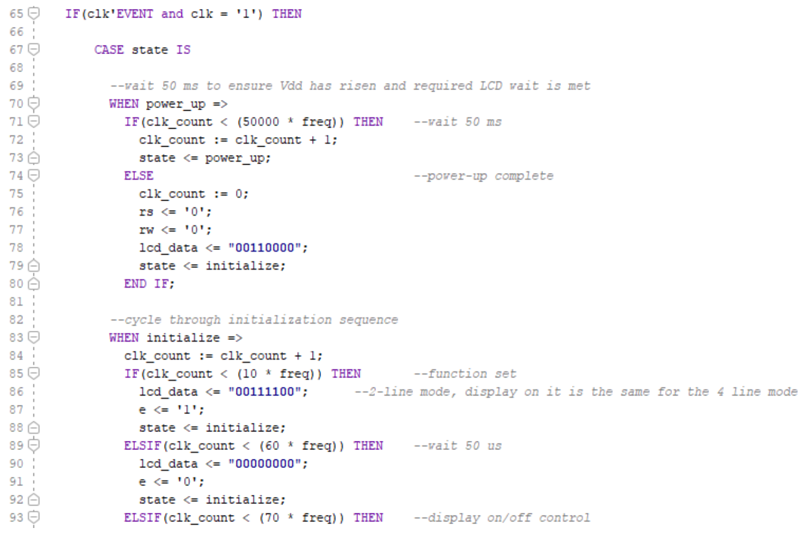
I. PL codes

• LCD_CONTROLLER IP

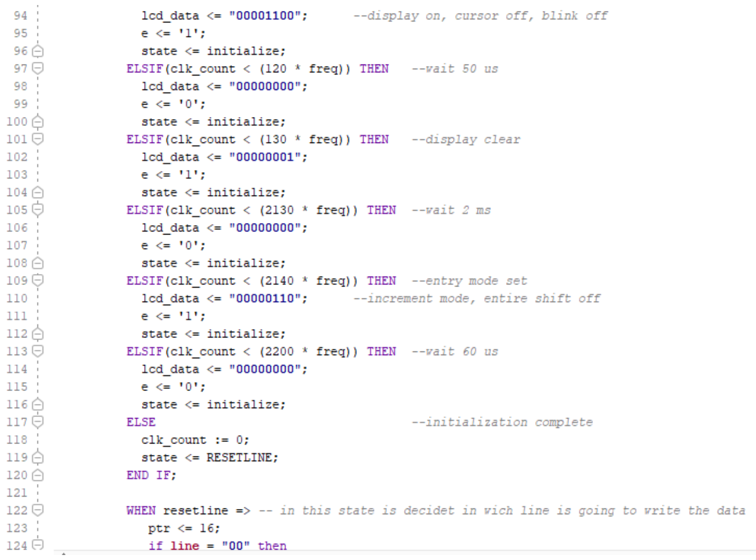
This is the principal code which does call another bloc called display decoder that is shown after.



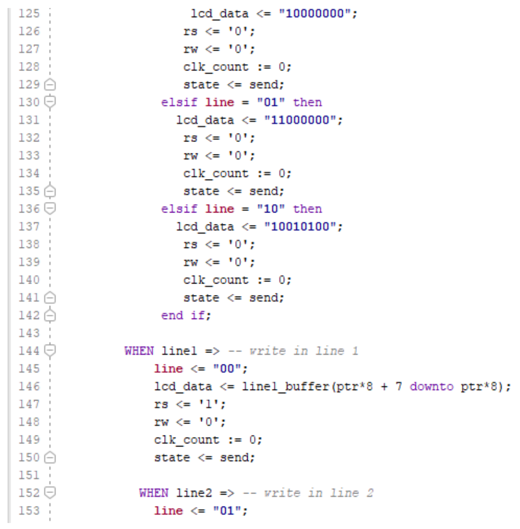
```vhdl
-- This code is made to power up and controll a LCD of 20x4 so prints the distance recived
--from 3 distance sensor controllers
-- It inputs 3 diferent datas of 9 bits and print the dista on three diferent lines
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY lcd_controller_v2 IS
  PORT(
    clk        : IN    STD_LOGIC;  --system clock
    reset_n    : IN    STD_LOGIC;  --active low reinitializes lcd
    Distance1  : IN    std_logic_vector (8 downto 0); -- data distance sensor input
    Distance2  : IN    std_logic_vector (8 downto 0);
    Distance3  : IN    std_logic_vector (8 downto 0);
    rw, rs, e  : OUT   STD_LOGIC;  --read/write, setup/data, and enable for lcd
    lcd_data   : OUT   STD_LOGIC_VECTOR(7 DOWNTO 0) --data signals for lcd

     );

END lcd_controller_v2;

ARCHITECTURE controller OF lcd_controller_v2 IS

  TYPE CONTROL IS(power_up, initialize, RESETLINE, line1, line2,line3, send);
  SIGNAL    state      : CONTROL;
  CONSTANT  freq       : INTEGER := 125; --system clock frequency in MHz
  SIGNAL    ptr           : natural range 0 to 16 := 15; -- To keep track of what character we are up to
  SIGNAL    line          : std_logic_vector  (1 downto 0):= "00";

  --the display that we use has 4 lines and 20 characters however the IC manage the info like it had 2 lines adn 40 characters

  signal line1_buffer : STD_LOGIC_VECTOR(127 downto 0); -- Data for all the characters on the the top line of the LCD
  signal line2_buffer : STD_LOGIC_VECTOR(127 downto 0); -- Data for all the characters on the midle line of the LCD
  signal line3_buffer : STD_LOGIC_VECTOR (127 downto 0);-- Data for all the characters on the  bottom line of the LCD

  COMPONENT display_decoder
    PORT(
        Distance : IN std_logic_vector(8 downto 0);  --max value (max distance) 400 cm
        display_out : OUT std_logic_vector(127 downto 0)  -- this will input the LCD;  16 characters in a line times the 8 bits of the register that the LCD
        );
    END COMPONENT;

BEGIN

    Instance_display_decoder1: display_decoder port map( -- instance of display decoder that is in charge to transform the 9 bits of the nput
        Distance1,                                       -- which is the distance in cm to the 8 bits that the LCD contrller understansd as a
        line1_buffer                                     -- specific character.  Ther is one for each line.
    );

    Instance_display_decoder2: display_decoder port map(
        Distance2,
        line2_buffer
    );

    Instance_display_decoder3: display_decoder port map(
        Distance3,
        line3_buffer
    );

  -- this proces first does the power up wich need specific timings and data and then is in charge to send the data
  -- to the LCD
  -- overall it is an state machine
  PROCESS(clk)
    VARIABLE clk_count : INTEGER := 0; --event counter for timing
  BEGIN
```



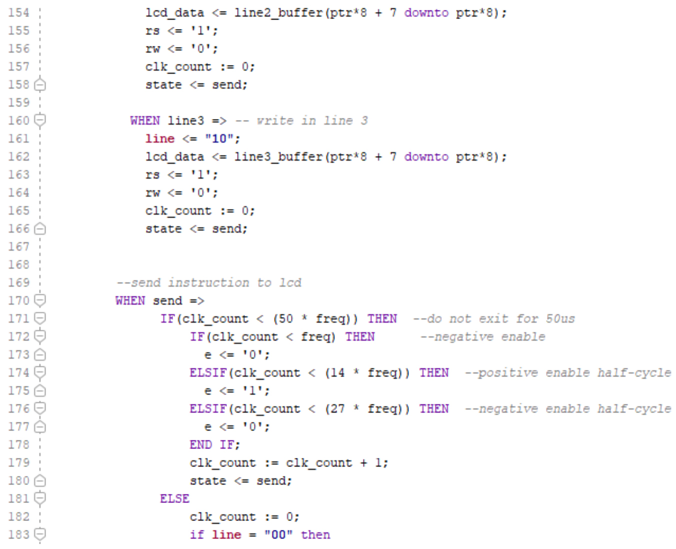
```
IF(clk'EVENT and clk = '1') THEN

    CASE state IS

      --wait 50 ms to ensure Vdd has risen and required LCD wait is met
      WHEN power_up =>
        IF(clk_count < (50000 * freq)) THEN    --wait 50 ms
          clk_count := clk_count + 1;
          state <= power_up;
        ELSE                                   --power-up complete
          clk_count := 0;
          rs <= '0';
          rw <= '0';
          lcd_data <= "00110000";
          state <= initialize;
        END IF;

      --cycle through initialization sequence
      WHEN initialize =>
        clk_count := clk_count + 1;
        IF(clk_count < (10 * freq)) THEN       --function set
          lcd_data <= "00111100";      --2-line mode, display on it is the same for the 4 line mode
          e <= '1';
          state <= initialize;
        ELSIF(clk_count < (60 * freq)) THEN    --wait 50 us
          lcd_data <= "00000000";
          e <= '0';
          state <= initialize;
        ELSIF(clk_count < (70 * freq)) THEN    --display on/off control
```



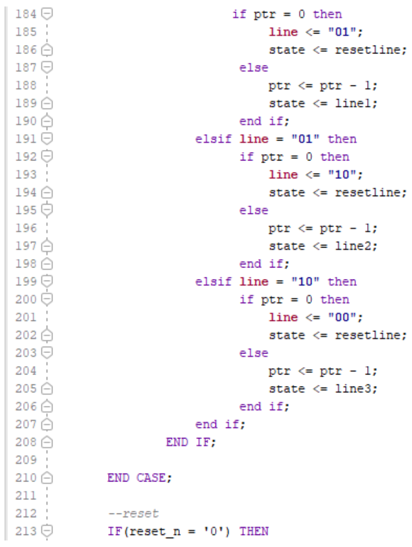
```
          lcd_data <= "00001100";      --display on, cursor off, blink off
          e <= '1';
          state <= initialize;
        ELSIF(clk_count < (120 * freq)) THEN   --wait 50 us
          lcd_data <= "00000000";
          e <= '0';
          state <= initialize;
        ELSIF(clk_count < (130 * freq)) THEN   --display clear
          lcd_data <= "00000001";
          e <= '1';
          state <= initialize;
        ELSIF(clk_count < (2130 * freq)) THEN  --wait 2 ms
          lcd_data <= "00000000";
          e <= '0';
          state <= initialize;
        ELSIF(clk_count < (2140 * freq)) THEN  --entry mode set
          lcd_data <= "00000110";      --increment mode, entire shift off
          e <= '1';
          state <= initialize;
        ELSIF(clk_count < (2200 * freq)) THEN  --wait 60 us
          lcd_data <= "00000000";
          e <= '0';
          state <= initialize;
        ELSE                                   --initialization complete
          clk_count := 0;
          state <= RESETLINE;
        END IF;

        WHEN resetline => -- in this state is decidet in wich line is going to write the data
           ptr <= 16;
           if line = "00" then
```



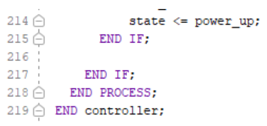
```
                 lcd_data <= "10000000";
                rs <= '0';
                rw <= '0';
                clk_count := 0;
                state <= send;
             elsif line = "01" then
               lcd_data <= "11000000";
                rs <= '0';
                rw <= '0';
                clk_count := 0;
                state <= send;
             elsif line = "10" then
               lcd_data <= "10010100";
                rs <= '0';
                rw <= '0';
                clk_count := 0;
                state <= send;
             end if;

        WHEN line1 => -- write in line 1
            line <= "00";
            lcd_data <= line1_buffer(ptr*8 + 7 downto ptr*8);
            rs <= '1';
            rw <= '0';
            clk_count := 0;
            state <= send;

          WHEN line2 => -- write in line 2
            line <= "01";
```



```vhdl
            lcd_data <= line2_buffer(ptr*8 + 7 downto ptr*8);
            rs <= '1';
            rw <= '0';
            clk_count := 0;
            state <= send;

          WHEN line3 => -- write in line 3
            line <= "10";
            lcd_data <= line3_buffer(ptr*8 + 7 downto ptr*8);
            rs <= '1';
            rw <= '0';
            clk_count := 0;
            state <= send;


        --send instruction to lcd
        WHEN send =>
              IF(clk_count < (50 * freq)) THEN  --do not exit for 50us
                  IF(clk_count < freq) THEN      --negative enable
                    e <= '0';
                  ELSIF(clk_count < (14 * freq)) THEN  --positive enable half-cycle
                    e <= '1';
                  ELSIF(clk_count < (27 * freq)) THEN  --negative enable half-cycle
                    e <= '0';
                  END IF;
                  clk_count := clk_count + 1;
                  state <= send;
              ELSE
                  clk_count := 0;
                  if line = "00" then
```

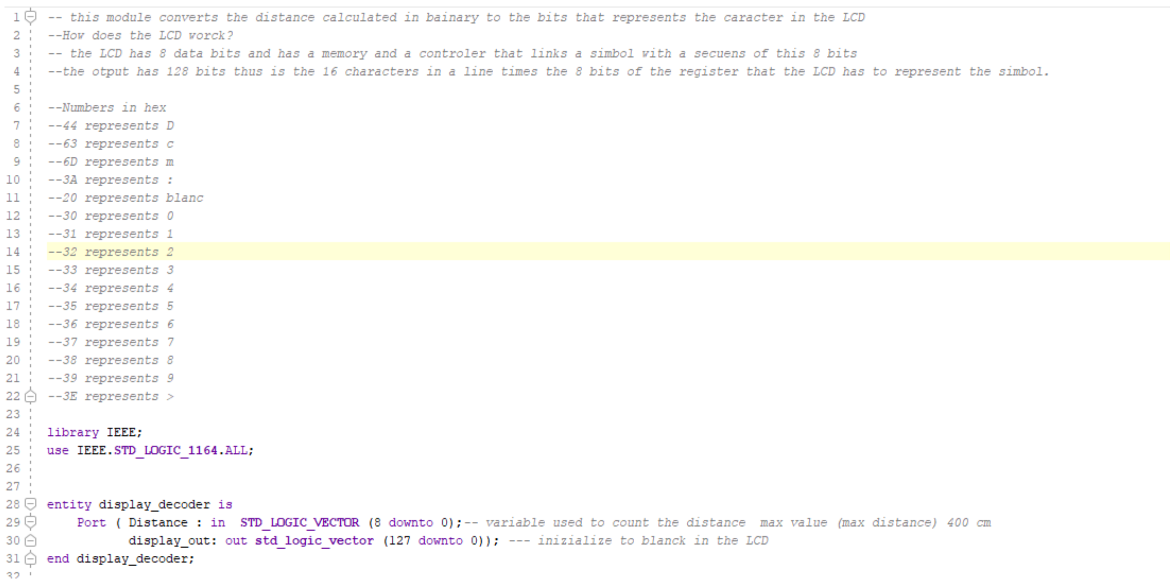


```vhdl
                       if ptr = 0 then
                            line <= "01";
                            state <= resetline;
                        else
                            ptr <= ptr - 1;
                            state <= line1;
                        end if;
                  elsif line = "01" then
                        if ptr = 0 then
                            line <= "10";
                            state <= resetline;
                        else
                            ptr <= ptr - 1;
                            state <= line2;
                        end if;
                  elsif line = "10" then
                        if ptr = 0 then
                            line <= "00";
                            state <= resetline;
                        else
                            ptr <= ptr - 1;
                            state <= line3;
                        end if;
                  end if;
              END IF;

      END CASE;

      --reset
      IF(reset_n = '0') THEN
```

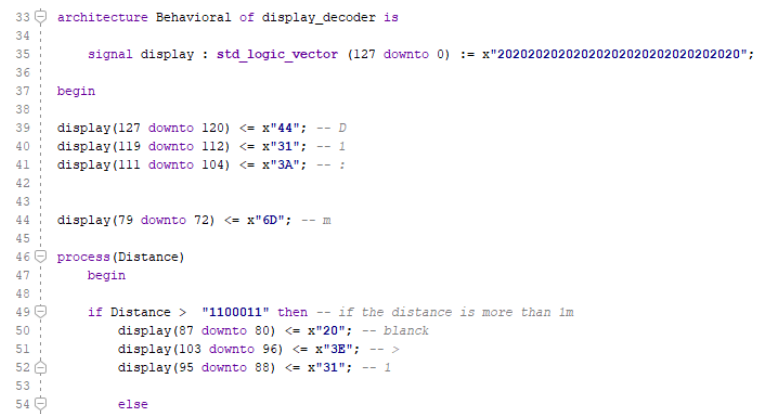


```vhdl
          state <= power_up;
      END IF;

    END IF;
  END PROCESS;
END controller;
```

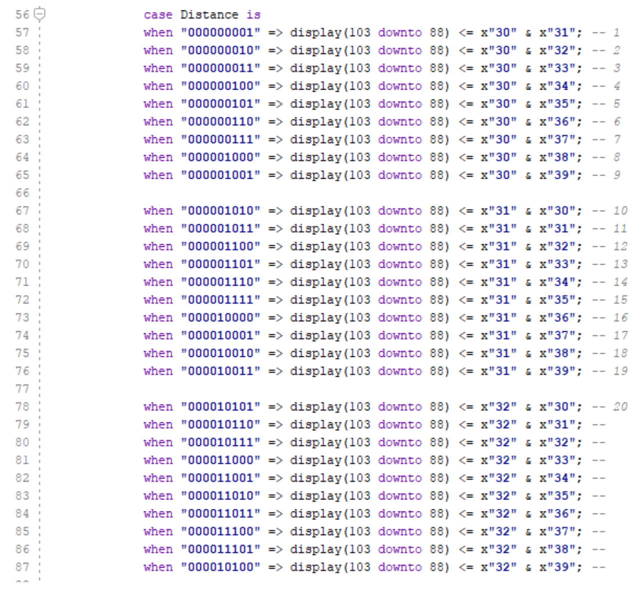
Next the display decoder is shown.



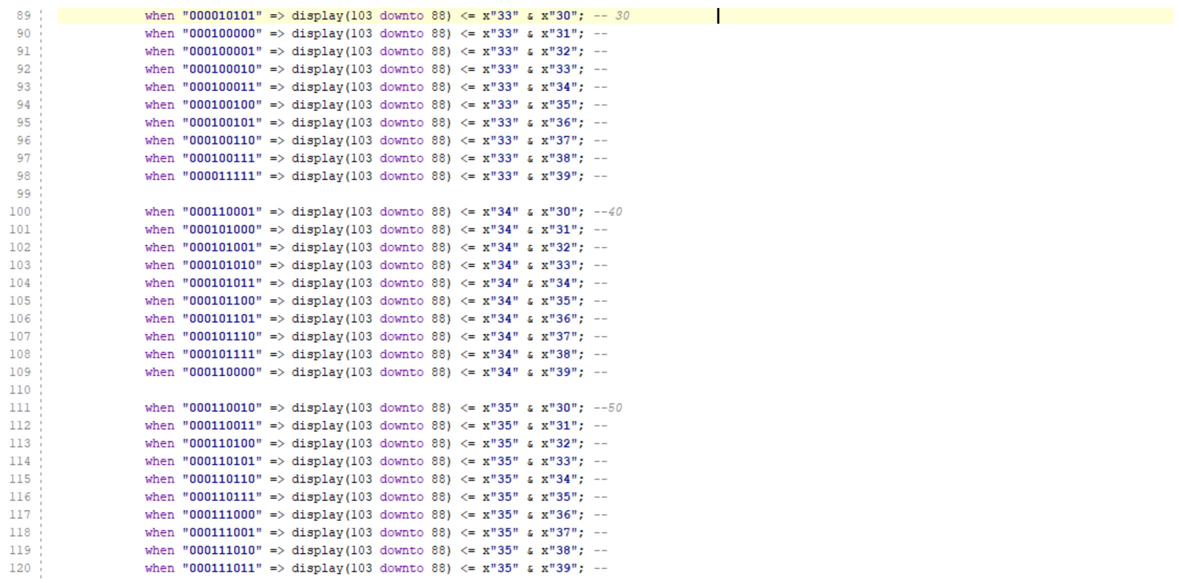
```vhdl
-- this module converts the distance calculated in bainary to the bits that represents the caracter in the LCD
--How does the LCD worck?
-- the LCD has 8 data bits and has a memory and a controler that links a simbol with a secuens of this 8 bits
--the otput has 128 bits thus is the 16 characters in a line times the 8 bits of the register that the LCD has to represent the simbol.

--Numbers in hex
--44 represents D
--63 represents c
--6D represents m
--3A represents :
--20 represents blanc
--30 represents 0
--31 represents 1
--32 represents 2
--33 represents 3
--34 represents 4
--35 represents 5
--36 represents 6
--37 represents 7
--38 represents 8
--39 represents 9
--3E represents >

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity display_decoder is
    Port ( Distance : in  STD_LOGIC_VECTOR (8 downto 0);-- variable used to count the distance  max value (max distance) 400 cm
           display_out: out std_logic_vector (127 downto 0)); --- inizialize to blanck in the LCD
end display_decoder;
```



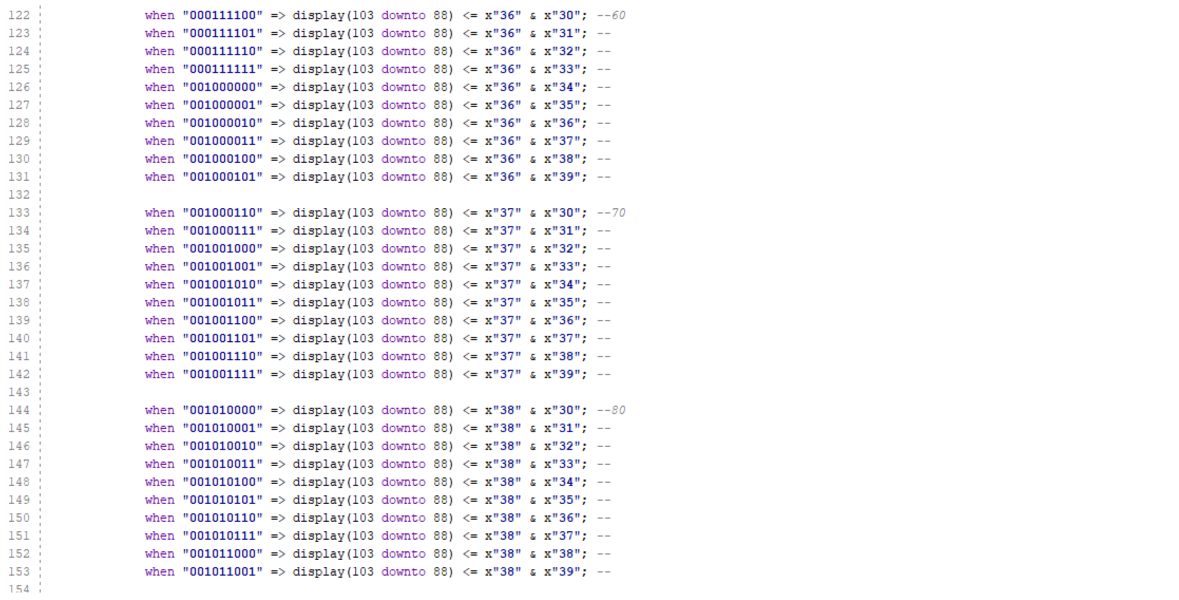
```vhdl
architecture Behavioral of display_decoder is

    signal display : std_logic_vector (127 downto 0) := x"20202020202020202020202020202020";

begin

display(127 downto 120) <= x"44"; -- D
display(119 downto 112) <= x"31"; -- 1
display(111 downto 104) <= x"3A"; -- :


display(79 downto 72) <= x"6D"; -- m

process(Distance)
    begin

    if Distance >  "1100011" then -- if the distance is more than 1m
        display(87 downto 80) <= x"20"; -- blanck
        display(103 downto 96) <= x"3E"; -- >
        display(95 downto 88) <= x"31"; -- 1

        else
```



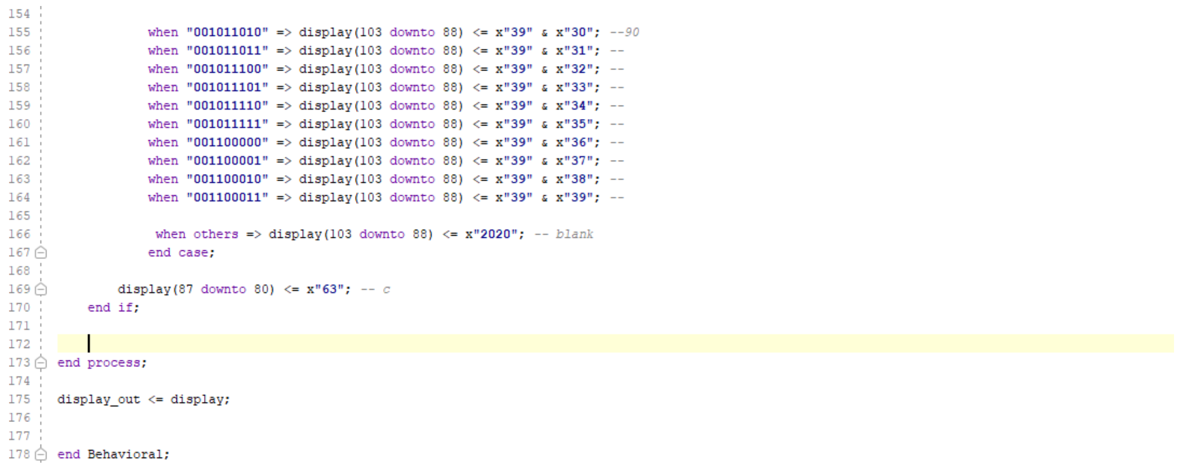
```vhdl
            case Distance is
            when "000000001" => display(103 downto 88) <= x"30" & x"31"; -- 1
            when "000000010" => display(103 downto 88) <= x"30" & x"32"; -- 2
            when "000000011" => display(103 downto 88) <= x"30" & x"33"; -- 3
            when "000000100" => display(103 downto 88) <= x"30" & x"34"; -- 4
            when "000000101" => display(103 downto 88) <= x"30" & x"35"; -- 5
            when "000000110" => display(103 downto 88) <= x"30" & x"36"; -- 6
            when "000000111" => display(103 downto 88) <= x"30" & x"37"; -- 7
            when "000001000" => display(103 downto 88) <= x"30" & x"38"; -- 8
            when "000001001" => display(103 downto 88) <= x"30" & x"39"; -- 9

            when "000001010" => display(103 downto 88) <= x"31" & x"30"; -- 10
            when "000001011" => display(103 downto 88) <= x"31" & x"31"; -- 11
            when "000001100" => display(103 downto 88) <= x"31" & x"32"; -- 12
            when "000001101" => display(103 downto 88) <= x"31" & x"33"; -- 13
            when "000001110" => display(103 downto 88) <= x"31" & x"34"; -- 14
            when "000001111" => display(103 downto 88) <= x"31" & x"35"; -- 15
            when "000010000" => display(103 downto 88) <= x"31" & x"36"; -- 16
            when "000010001" => display(103 downto 88) <= x"31" & x"37"; -- 17
            when "000010010" => display(103 downto 88) <= x"31" & x"38"; -- 18
            when "000010011" => display(103 downto 88) <= x"31" & x"39"; -- 19

            when "000010101" => display(103 downto 88) <= x"32" & x"30"; -- 20
            when "000010110" => display(103 downto 88) <= x"32" & x"31"; --
            when "000010111" => display(103 downto 88) <= x"32" & x"32"; --
            when "000011000" => display(103 downto 88) <= x"32" & x"33"; --
            when "000011001" => display(103 downto 88) <= x"32" & x"34"; --
            when "000011010" => display(103 downto 88) <= x"32" & x"35"; --
            when "000011011" => display(103 downto 88) <= x"32" & x"36"; --
            when "000011100" => display(103 downto 88) <= x"32" & x"37"; --
            when "000011101" => display(103 downto 88) <= x"32" & x"38"; --
            when "000010100" => display(103 downto 88) <= x"32" & x"39"; --
```



```vhdl
            when "000010101" => display(103 downto 88) <= x"33" & x"30"; -- 30
            when "000100000" => display(103 downto 88) <= x"33" & x"31"; --
            when "000100001" => display(103 downto 88) <= x"33" & x"32"; --
            when "000100010" => display(103 downto 88) <= x"33" & x"33"; --
            when "000100011" => display(103 downto 88) <= x"33" & x"34"; --
            when "000100100" => display(103 downto 88) <= x"33" & x"35"; --
            when "000100101" => display(103 downto 88) <= x"33" & x"36"; --
            when "000100110" => display(103 downto 88) <= x"33" & x"37"; --
            when "000100111" => display(103 downto 88) <= x"33" & x"38"; --
            when "000011111" => display(103 downto 88) <= x"33" & x"39"; --

            when "000110001" => display(103 downto 88) <= x"34" & x"30"; --40
            when "000101000" => display(103 downto 88) <= x"34" & x"31"; --
            when "000101001" => display(103 downto 88) <= x"34" & x"32"; --
            when "000101010" => display(103 downto 88) <= x"34" & x"33"; --
            when "000101011" => display(103 downto 88) <= x"34" & x"34"; --
            when "000101100" => display(103 downto 88) <= x"34" & x"35"; --
            when "000101101" => display(103 downto 88) <= x"34" & x"36"; --
            when "000101110" => display(103 downto 88) <= x"34" & x"37"; --
            when "000101111" => display(103 downto 88) <= x"34" & x"38"; --
            when "000110000" => display(103 downto 88) <= x"34" & x"39"; --

            when "000110010" => display(103 downto 88) <= x"35" & x"30"; --50
            when "000110011" => display(103 downto 88) <= x"35" & x"31"; --
            when "000110100" => display(103 downto 88) <= x"35" & x"32"; --
            when "000110101" => display(103 downto 88) <= x"35" & x"33"; --
            when "000110110" => display(103 downto 88) <= x"35" & x"34"; --
            when "000110111" => display(103 downto 88) <= x"35" & x"35"; --
            when "000111000" => display(103 downto 88) <= x"35" & x"36"; --
            when "000111001" => display(103 downto 88) <= x"35" & x"37"; --
            when "000111010" => display(103 downto 88) <= x"35" & x"38"; --
            when "000111011" => display(103 downto 88) <= x"35" & x"39"; --
```



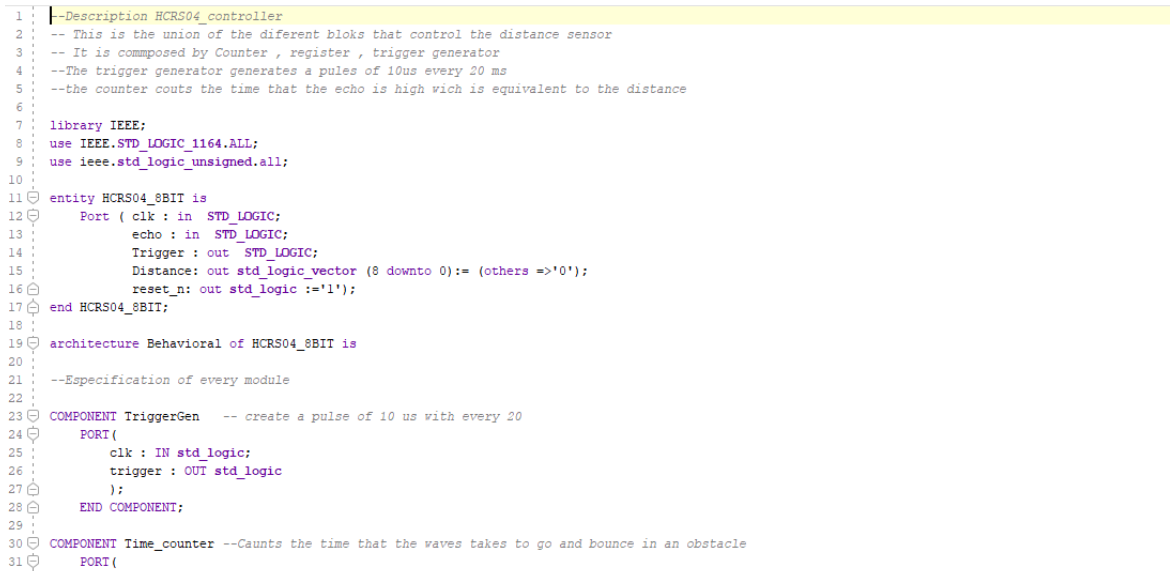
```vhdl
            when "000111100" => display(103 downto 88) <= x"36" & x"30"; --60
            when "000111101" => display(103 downto 88) <= x"36" & x"31"; --
            when "000111110" => display(103 downto 88) <= x"36" & x"32"; --
            when "000111111" => display(103 downto 88) <= x"36" & x"33"; --
            when "001000000" => display(103 downto 88) <= x"36" & x"34"; --
            when "001000001" => display(103 downto 88) <= x"36" & x"35"; --
            when "001000010" => display(103 downto 88) <= x"36" & x"36"; --
            when "001000011" => display(103 downto 88) <= x"36" & x"37"; --
            when "001000100" => display(103 downto 88) <= x"36" & x"38"; --
            when "001000101" => display(103 downto 88) <= x"36" & x"39"; --

            when "001000110" => display(103 downto 88) <= x"37" & x"30"; --70
            when "001000111" => display(103 downto 88) <= x"37" & x"31"; --
            when "001001000" => display(103 downto 88) <= x"37" & x"32"; --
            when "001001001" => display(103 downto 88) <= x"37" & x"33"; --
            when "001001010" => display(103 downto 88) <= x"37" & x"34"; --
            when "001001011" => display(103 downto 88) <= x"37" & x"35"; --
            when "001001100" => display(103 downto 88) <= x"37" & x"36"; --
            when "001001101" => display(103 downto 88) <= x"37" & x"37"; --
            when "001001110" => display(103 downto 88) <= x"37" & x"38"; --
            when "001001111" => display(103 downto 88) <= x"37" & x"39"; --

            when "001010000" => display(103 downto 88) <= x"38" & x"30"; --80
            when "001010001" => display(103 downto 88) <= x"38" & x"31"; --
            when "001010010" => display(103 downto 88) <= x"38" & x"32"; --
            when "001010011" => display(103 downto 88) <= x"38" & x"33"; --
            when "001010100" => display(103 downto 88) <= x"38" & x"34"; --
            when "001010101" => display(103 downto 88) <= x"38" & x"35"; --
            when "001010110" => display(103 downto 88) <= x"38" & x"36"; --
            when "001010111" => display(103 downto 88) <= x"38" & x"37"; --
            when "001011000" => display(103 downto 88) <= x"38" & x"38"; --
            when "001011001" => display(103 downto 88) <= x"38" & x"39"; --
```



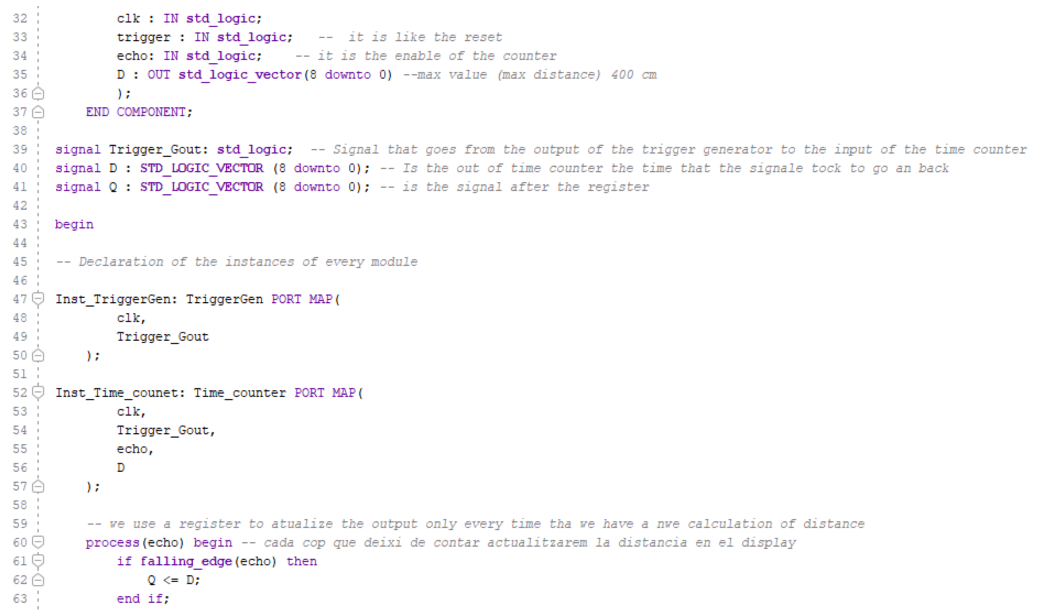
```vhdl
            when "001011010" => display(103 downto 88) <= x"39" & x"30"; --90
            when "001011011" => display(103 downto 88) <= x"39" & x"31"; --
            when "001011100" => display(103 downto 88) <= x"39" & x"32"; --
            when "001011101" => display(103 downto 88) <= x"39" & x"33"; --
            when "001011110" => display(103 downto 88) <= x"39" & x"34"; --
            when "001011111" => display(103 downto 88) <= x"39" & x"35"; --
            when "001100000" => display(103 downto 88) <= x"39" & x"36"; --
            when "001100001" => display(103 downto 88) <= x"39" & x"37"; --
            when "001100010" => display(103 downto 88) <= x"39" & x"38"; --
            when "001100011" => display(103 downto 88) <= x"39" & x"39"; --

             when others => display(103 downto 88) <= x"2020"; -- blank
            end case;

        display(87 downto 80) <= x"63"; -- c
    end if;


end process;

display_out <= display;


end Behavioral;
```

- **HCRS04_CONTROLLER IP**

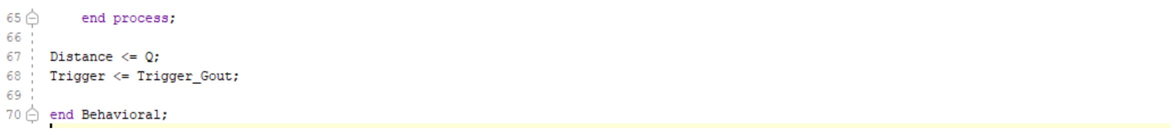
This IP is made with the combination of two other blocs. The first code is the main where it calls the other ones.



```
--Description HCRS04_controller
-- This is the union of the diferent bloks that control the distance sensor
-- It is commposed by Counter , register , trigger generator
--The trigger generator generates a pules of 10us every 20 ms
--the counter couts the time that the echo is high wich is equivalent to the distance

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;

entity HCRS04_8BIT is
    Port ( clk : in  STD_LOGIC;
           echo : in  STD_LOGIC;
           Trigger : out  STD_LOGIC;
           Distance: out std_logic_vector (8 downto 0):= (others =>'0');
           reset_n: out std_logic :='1');
end HCRS04_8BIT;

architecture Behavioral of HCRS04_8BIT is

--Especification of every module

COMPONENT TriggerGen   -- create a pulse of 10 us with every 20
    PORT(
        clk : IN std_logic;
        trigger : OUT std_logic
        );
    END COMPONENT;

COMPONENT Time_counter --Caunts the time that the waves takes to go and bounce in an obstacle
    PORT(
```

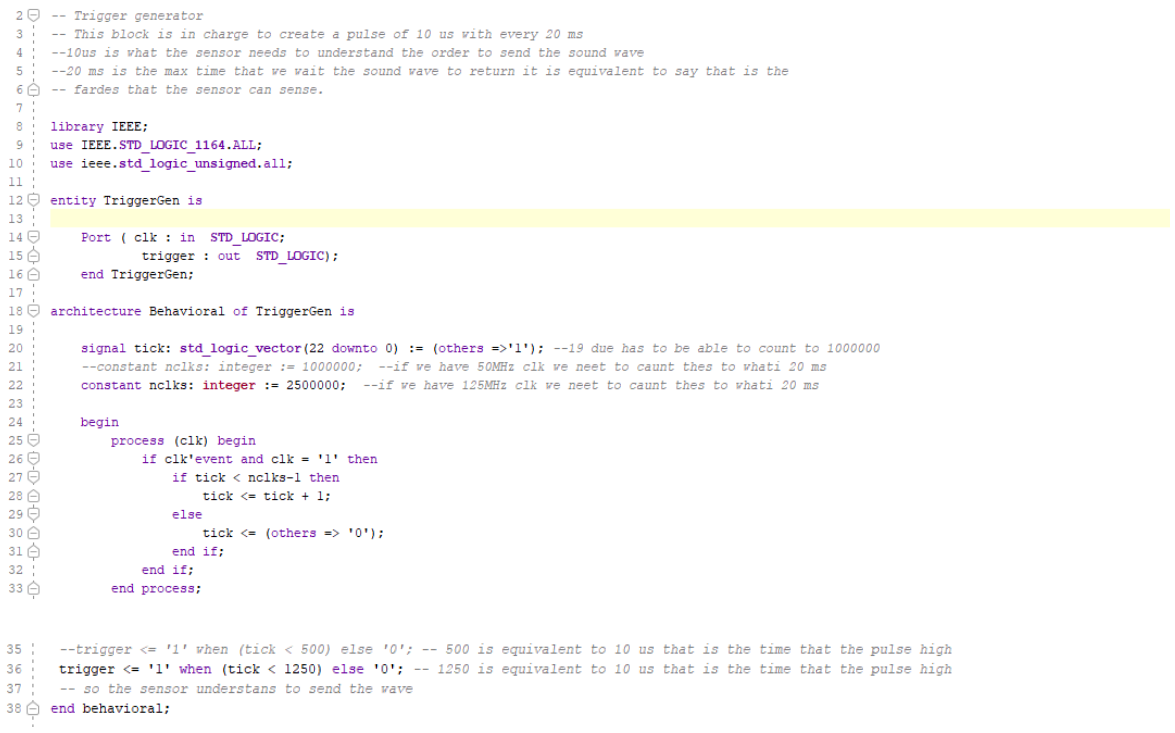


```
         clk : IN std_logic;
         trigger : IN std_logic;    --  it is like the reset
         echo: IN std_logic;    -- it is the enable of the counter
         D : OUT std_logic_vector(8 downto 0) --max value (max distance) 400 cm
         );
     END COMPONENT;

 signal Trigger_Gout: std_logic;  -- Signal that goes from the output of the trigger generator to the input of the time counter
 signal D : STD_LOGIC_VECTOR (8 downto 0); -- Is the out of time counter the time that the signale tock to go an back
 signal Q : STD_LOGIC_VECTOR (8 downto 0); -- is the signal after the register

 begin

 -- Declaration of the instances of every module

 Inst_TriggerGen: TriggerGen PORT MAP(
         clk,
         Trigger_Gout
     );

 Inst_Time_counet: Time_counter PORT MAP(
         clk,
         Trigger_Gout,
         echo,
         D
     );

     -- we use a register to atualize the output only every time tha we have a nwe calculation of distance
     process(echo) begin -- cada cop que deixi de contar actualitzarem la distancia en el display
         if falling_edge(echo) then
             Q <= D;
         end if;
```



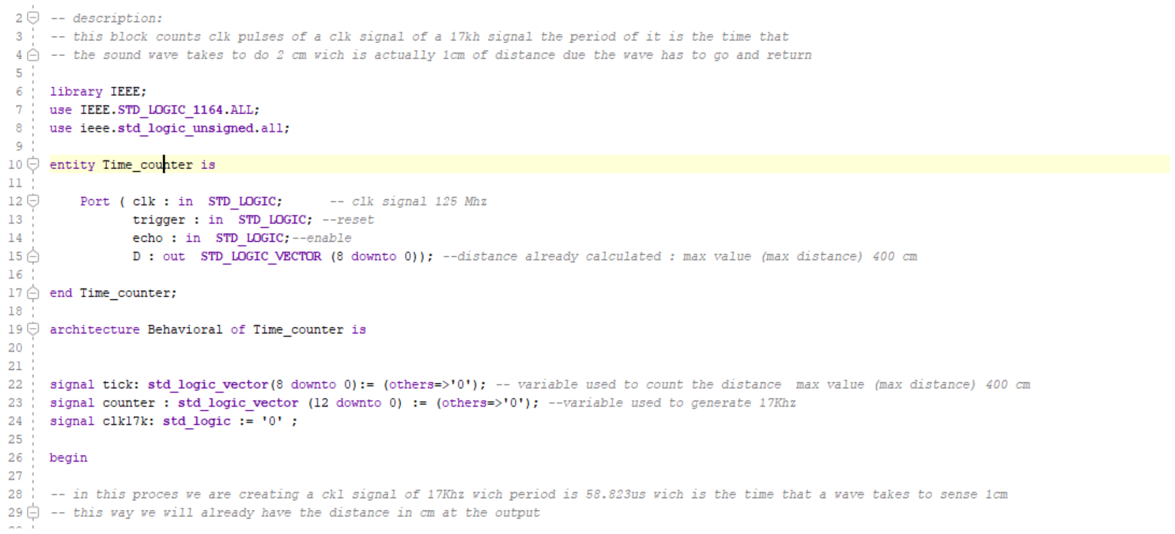
```
    end process;

Distance <= Q;
Trigger <= Trigger_Gout;

end Behavioral;
```

Next code is the trigger generator

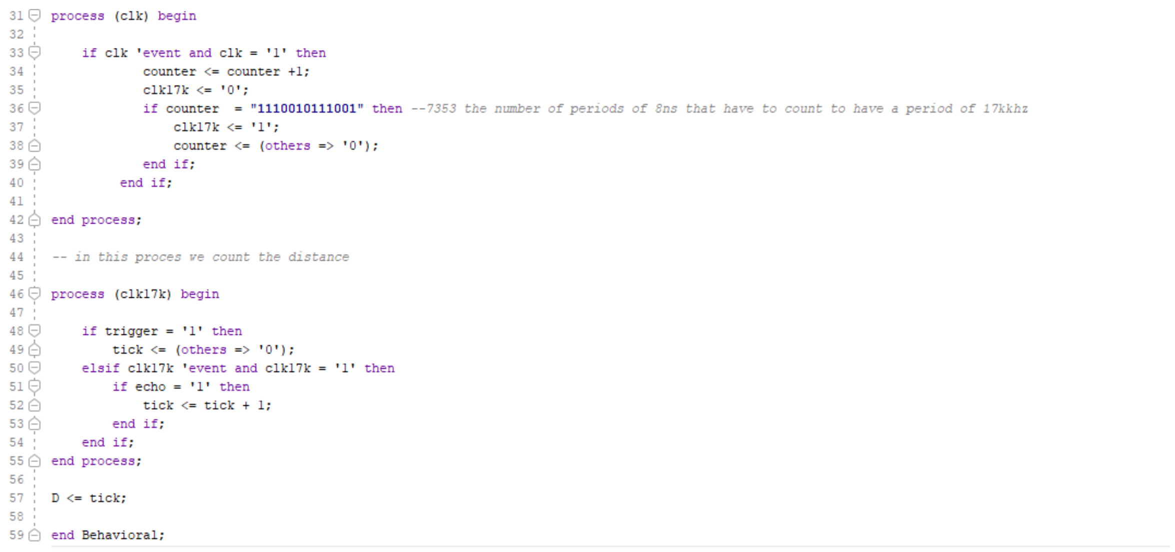


```vhdl
-- Trigger generator
-- This block is in charge to create a pulse of 10 us with every 20 ms
--10us is what the sensor needs to understand the order to send the sound wave
--20 ms is the max time that we wait the sound wave to return it is equivalent to say that is the
-- fardes that the sensor can sense.

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;

entity TriggerGen is

    Port ( clk : in  STD_LOGIC;
            trigger : out  STD_LOGIC);
    end TriggerGen;

architecture Behavioral of TriggerGen is

    signal tick: std_logic_vector(22 downto 0) := (others =>'1'); --19 due has to be able to count to 1000000
    --constant nclks: integer := 1000000;  --if we have 50MHz clk we neet to caunt thes to whati 20 ms
    constant nclks: integer := 2500000;  --if we have 125MHz clk we neet to caunt thes to whati 20 ms

    begin
        process (clk) begin
            if clk'event and clk = '1' then
                if tick < nclks-1 then
                    tick <= tick + 1;
                else
                    tick <= (others => '0');
                end if;
            end if;
        end process;


 --trigger <= '1' when (tick < 500) else '0'; -- 500 is equivalent to 10 us that is the time that the pulse high
 trigger <= '1' when (tick < 1250) else '0'; -- 1250 is equivalent to 10 us that is the time that the pulse high
 -- so the sensor understans to send the wave
end behavioral;
```

The following code is for the counter.



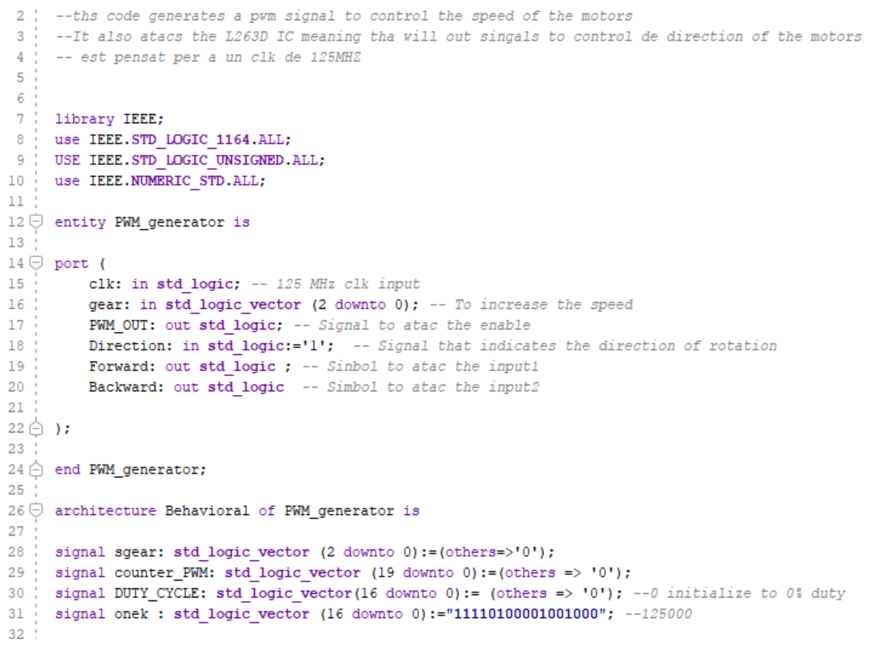
```vhdl
-- description:
-- this block counts clk pulses of a clk signal of a 17kh signal the period of it is the time that
-- the sound wave takes to do 2 cm wich is actually 1cm of distance due the wave has to go and return

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;

entity Time_counter is

    Port ( clk : in  STD_LOGIC;       -- clk signal 125 Mhz
           trigger : in  STD_LOGIC; --reset
           echo : in  STD_LOGIC;--enable
           D : out  STD_LOGIC_VECTOR (8 downto 0)); --distance already calculated : max value (max distance) 400 cm

end Time_counter;

architecture Behavioral of Time_counter is


signal tick: std_logic_vector(8 downto 0):= (others=>'0'); -- variable used to count the distance  max value (max distance) 400 cm
signal counter : std_logic_vector (12 downto 0) := (others=>'0'); --variable used to generate 17Khz
signal clk17k: std_logic := '0' ;

begin

-- in this proces we are creating a ckl signal of 17Khz wich period is 58.823us wich is the time that a wave takes to sense 1cm
-- this way we will already have the distance in cm at the output
```



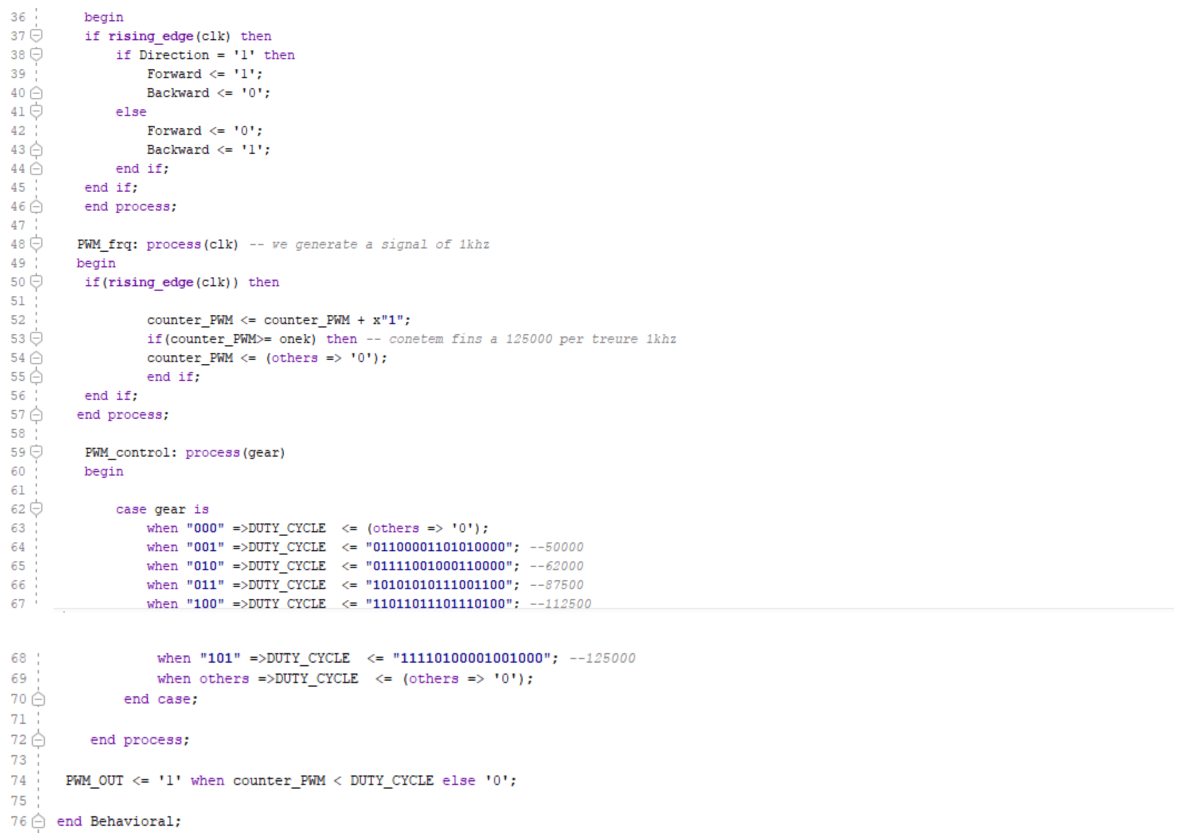
```
process (clk) begin

    if clk 'event and clk = '1' then
            counter <= counter +1;
            clk17k <= '0';
            if counter  = "1110010111001" then --7353 the number of periods of 8ns that have to count to have a period of 17kkhz
                clk17k <= '1';
                counter <= (others => '0');
            end if;
         end if;

end process;

-- in this proces we count the distance

process (clk17k) begin

    if trigger = '1' then
        tick <= (others => '0');
    elsif clk17k 'event and clk17k = '1' then
        if echo = '1' then
            tick <= tick + 1;
        end if;
    end if;
end process;

D <= tick;

end Behavioral;
```

- **L293D_CONTROLLER IP**

This IP is connected to the AXI register firs we will see the code of the motor control.



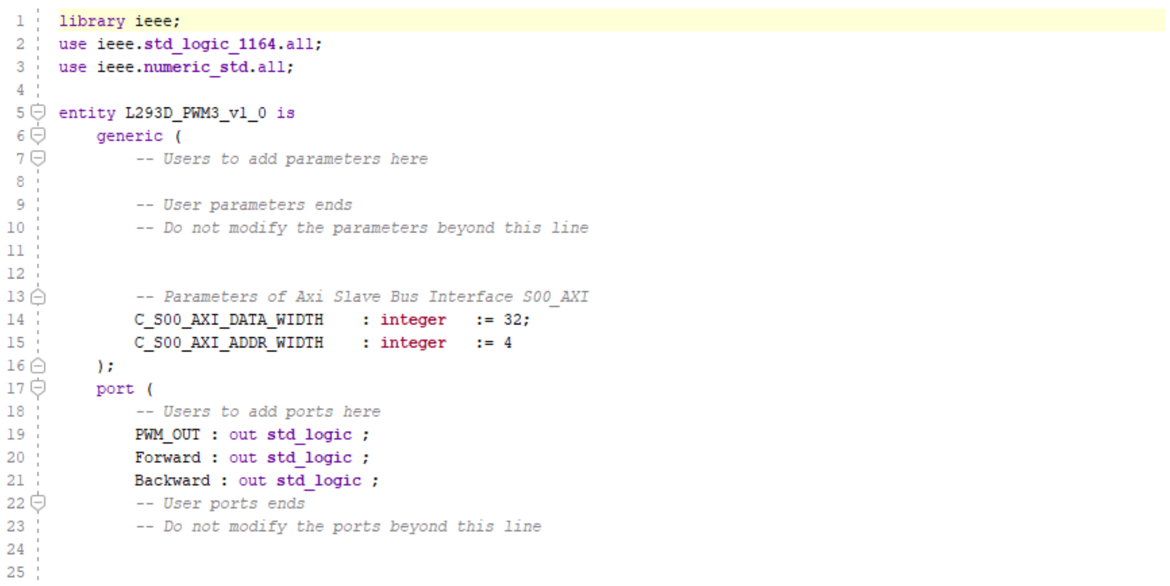
```
--ths code generates a pwm signal to control the speed of the motors
--It also atacs the L263D IC meaning tha will out singals to control de direction of the motors
-- est pensat per a un clk de 125MHZ


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;

entity PWM_generator is

port (
    clk: in std_logic; -- 125 MHz clk input
    gear: in std_logic_vector (2 downto 0); -- To increase the speed
    PWM_OUT: out std_logic; -- Signal to atac the enable
    Direction: in std_logic:='1';  -- Signal that indicates the direction of rotation
    Forward: out std_logic ; -- Sinbol to atac the input1
    Backward: out std_logic  -- Simbol to atac the input2

);

end PWM_generator;

architecture Behavioral of PWM_generator is

signal sgear: std_logic_vector (2 downto 0):=(others=>'0');
signal counter_PWM: std_logic_vector (19 downto 0):=(others => '0');
signal DUTY_CYCLE: std_logic_vector(16 downto 0):= (others => '0'); --0 initialize to 0% duty
signal onek : std_logic_vector (16 downto 0):="11110100001001000"; --125000

```



```
  begin
  if rising_edge(clk) then
      if Direction = '1' then
          Forward <= '1';
          Backward <= '0';
      else
          Forward <= '0';
          Backward <= '1';
      end if;
  end if;
  end process;

 PWM_frq: process(clk) -- we generate a signal of 1khz
 begin
  if(rising_edge(clk)) then

          counter_PWM <= counter_PWM + x"1";
          if(counter_PWM>= onek) then -- conetem fins a 125000 per treure 1khz
          counter_PWM <= (others => '0');
          end if;
  end if;
 end process;

  PWM_control: process(gear)
  begin

      case gear is
          when "000" =>DUTY_CYCLE  <= (others => '0');
          when "001" =>DUTY_CYCLE  <= "0110000110101000"; --50000
          when "010" =>DUTY_CYCLE  <= "0111100100011000"; --62000
          when "011" =>DUTY_CYCLE  <= "1010101011100110"; --87500
          when "100" =>DUTY_CYCLE  <= "1101101110111010"; --112500
          when "101" =>DUTY_CYCLE  <= "1111010000100100"; --125000
          when others =>DUTY_CYCLE  <= (others => '0');
      end case;

  end process;

 PWM_OUT <= '1' when counter_PWM < DUTY_CYCLE else '0';

end Behavioral;
```

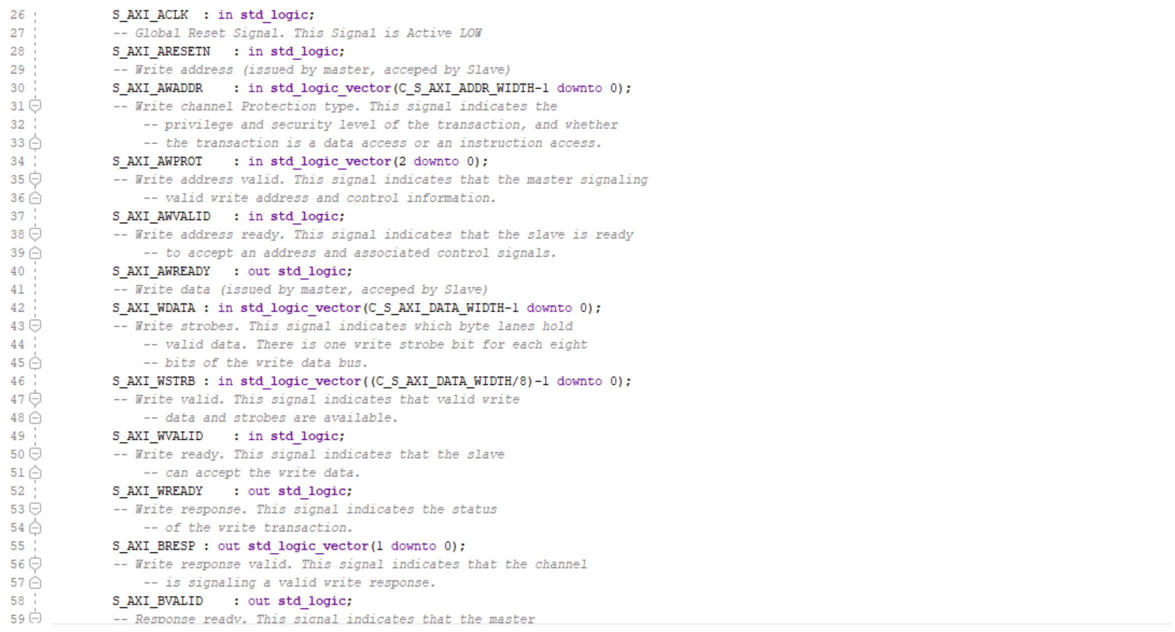
Now it is shown the parts of the code of the AXI implementation were the motor control interacts with the AXI protocol. There are two files , in the first one is were the AXI protocol is implemented.

First we declare the outputs of the motor control IP



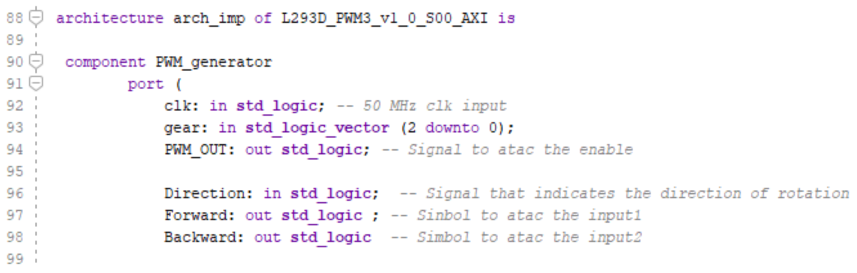
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity L293D_PWM3_v1_0 is
    generic (
        -- Users to add parameters here

        -- User parameters ends
        -- Do not modify the parameters beyond this line


        -- Parameters of Axi Slave Bus Interface S00_AXI
        C_S00_AXI_DATA_WIDTH    : integer   := 32;
        C_S00_AXI_ADDR_WIDTH    : integer   := 4
    );
    port (
        -- Users to add ports here
        PWM_OUT : out std_logic ;
        Forward : out std_logic ;
        Backward : out std_logic ;
        -- User ports ends
        -- Do not modify the ports beyond this line


```

Next we see how some of the input and output AXI signals are declared, this part is not written by the author it is automatically generated when the IP is generated with an AXI protocol.



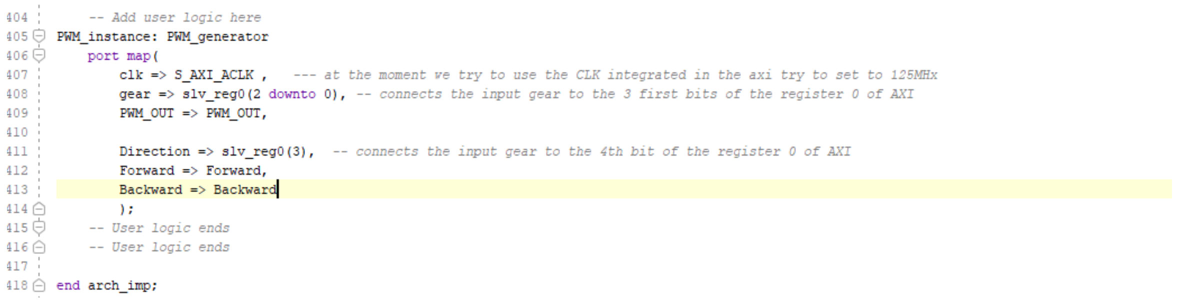
```vhdl
        S_AXI_ACLK  : in std_logic;
        -- Global Reset Signal. This Signal is Active LOW
        S_AXI_ARESETN   : in std_logic;
        -- Write address (issued by master, acceped by Slave)
        S_AXI_AWADDR    : in std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
        -- Write channel Protection type. This signal indicates the
            -- privilege and security level of the transaction, and whether
            -- the transaction is a data access or an instruction access.
        S_AXI_AWPROT    : in std_logic_vector(2 downto 0);
        -- Write address valid. This signal indicates that the master signaling
            -- valid write address and control information.
        S_AXI_AWVALID   : in std_logic;
        -- Write address ready. This signal indicates that the slave is ready
            -- to accept an address and associated control signals.
        S_AXI_AWREADY   : out std_logic;
        -- Write data (issued by master, acceped by Slave)
        S_AXI_WDATA : in std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
        -- Write strobes. This signal indicates which byte lanes hold
            -- valid data. There is one write strobe bit for each eight
            -- bits of the write data bus.
        S_AXI_WSTRB : in std_logic_vector((C_S_AXI_DATA_WIDTH/8)-1 downto 0);
        -- Write valid. This signal indicates that valid write
            -- data and strobes are available.
        S_AXI_WVALID    : in std_logic;
        -- Write ready. This signal indicates that the slave
            -- can accept the write data.
        S_AXI_WREADY    : out std_logic;
        -- Write response. This signal indicates the status
            -- of the write transaction.
        S_AXI_BRESP : out std_logic_vector(1 downto 0);
        -- Write response valid. This signal indicates that the channel
            -- is signaling a valid write response.
        S_AXI_BVALID    : out std_logic;
        -- Response ready. This signal indicates that the master
```

Now we declare the motor controller.



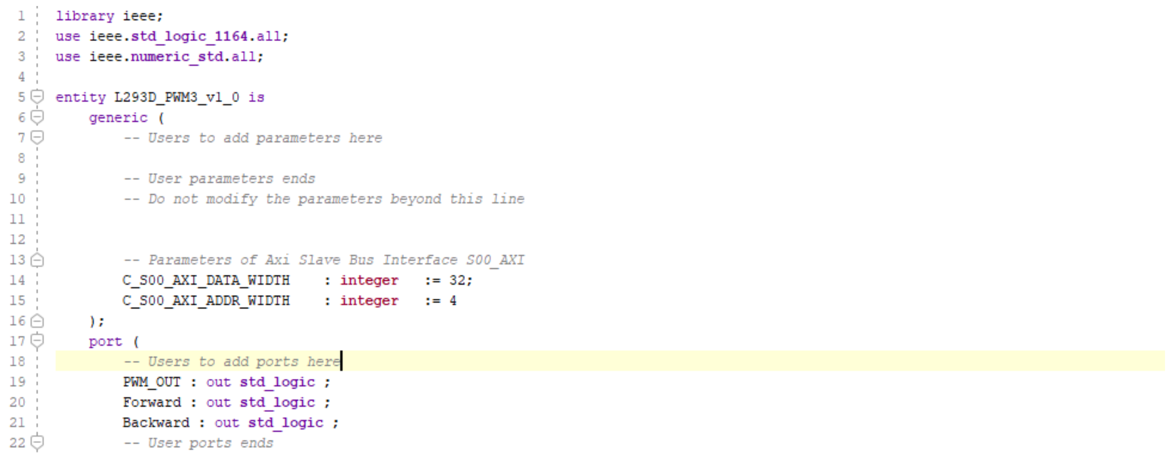
```vhdl
architecture arch_imp of L293D_PWM3_v1_0_S00_AXI is

 component PWM_generator
        port (
            clk: in std_logic; -- 50 MHz clk input
            gear: in std_logic_vector (2 downto 0);
            PWM_OUT: out std_logic; -- Signal to atac the enable

            Direction: in std_logic;  -- Signal that indicates the direction of rotation
            Forward: out std_logic ; -- Sinbol to atac the input1
            Backward: out std_logic  -- Simbol to atac the input2

```

After a long code of implementing the AXI protocol we eventually use the motor controller connecting the inputs to the registers of the AXI.

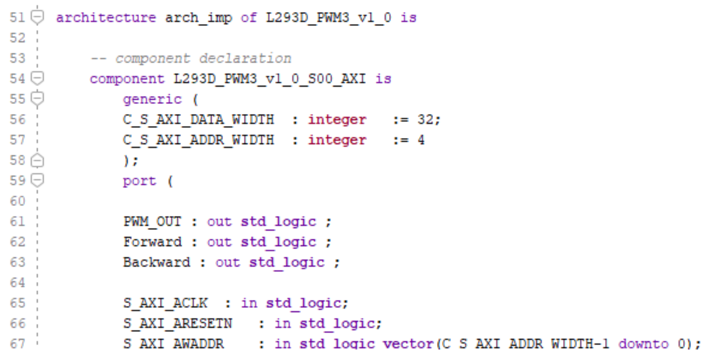


```vhdl
    -- Add user logic here
PWM_instance: PWM_generator
    port map(
        clk => S_AXI_ACLK ,   --- at the moment we try to use the CLK integrated in the axi try to set to 125MHx
        gear => slv_reg0(2 downto 0), -- connects the input gear to the 3 first bits of the register 0 of AXI
        PWM_OUT => PWM_OUT,

        Direction => slv_reg0(3),  -- connects the input gear to the 4th bit of the register 0 of AXI
        Forward => Forward,
        Backward => Backward
        );
    -- User logic ends
    -- User logic ends

end arch_imp;
```

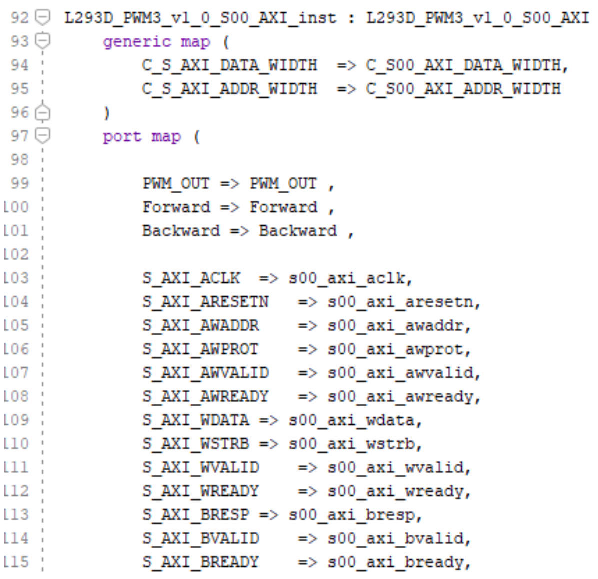
Now we will see the second fille that compose the AXI motor controller IP. This file is a wrapper of the AXI implementation and the motor controller.



```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity L293D_PWM3_v1_0 is
    generic (
        -- Users to add parameters here

        -- User parameters ends
        -- Do not modify the parameters beyond this line


        -- Parameters of Axi Slave Bus Interface S00_AXI
        C_S00_AXI_DATA_WIDTH    : integer   := 32;
        C_S00_AXI_ADDR_WIDTH    : integer   := 4
    );
    port (
        -- Users to add ports here
        PWM_OUT : out std_logic ;
        Forward : out std_logic ;
        Backward : out std_logic ;
        -- User ports ends
```

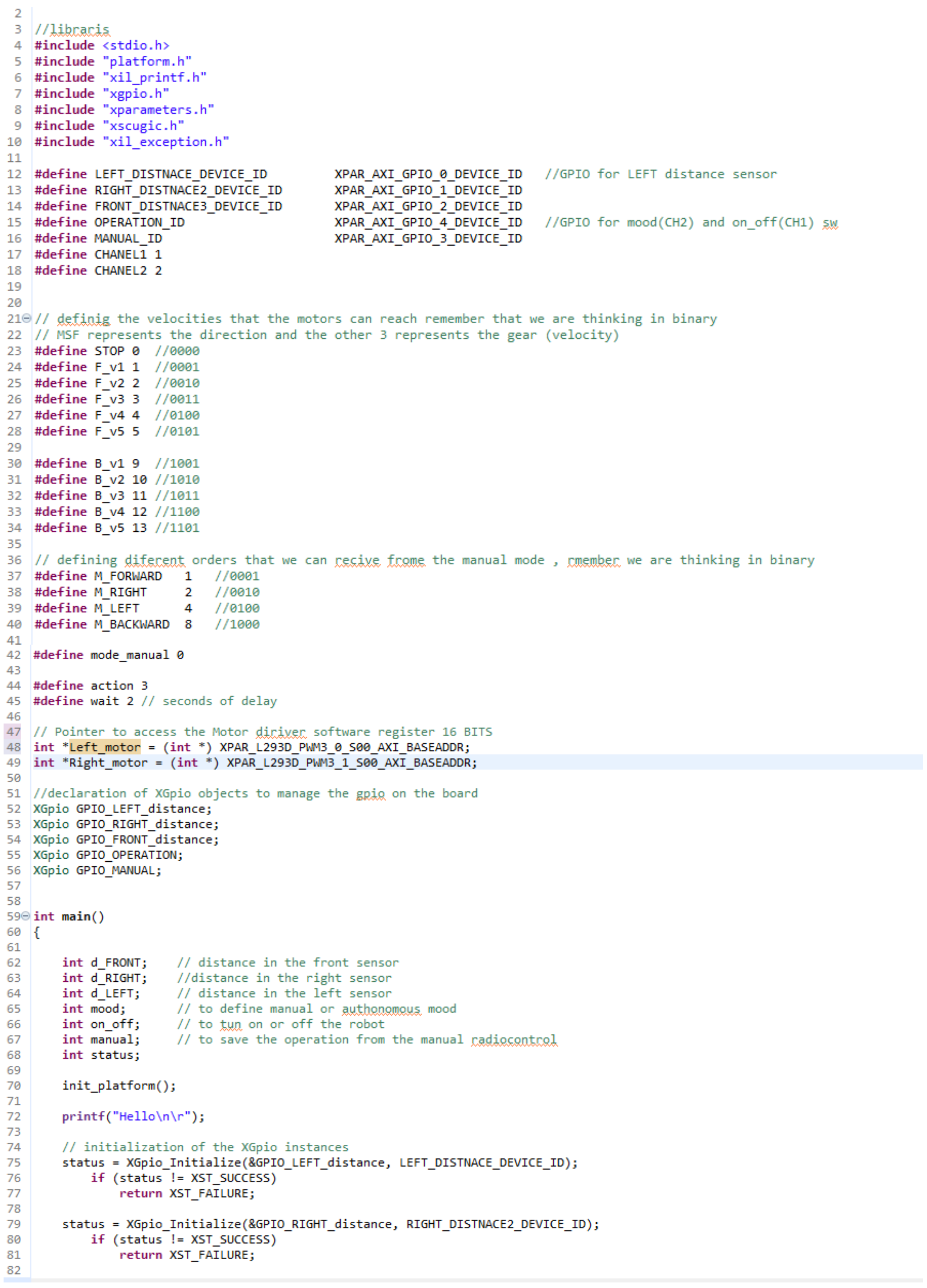


```vhdl
architecture arch_imp of L293D_PWM3_v1_0 is

    -- component declaration
    component L293D_PWM3_v1_0_S00_AXI is
        generic (
        C_S_AXI_DATA_WIDTH  : integer   := 32;
        C_S_AXI_ADDR_WIDTH  : integer   := 4
        );
        port (

        PWM_OUT : out std_logic ;
        Forward : out std_logic ;
        Backward : out std_logic ;

        S_AXI_ACLK  : in std_logic;
        S_AXI_ARESETN   : in std_logic;
        S_AXI_AWADDR    : in std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
```

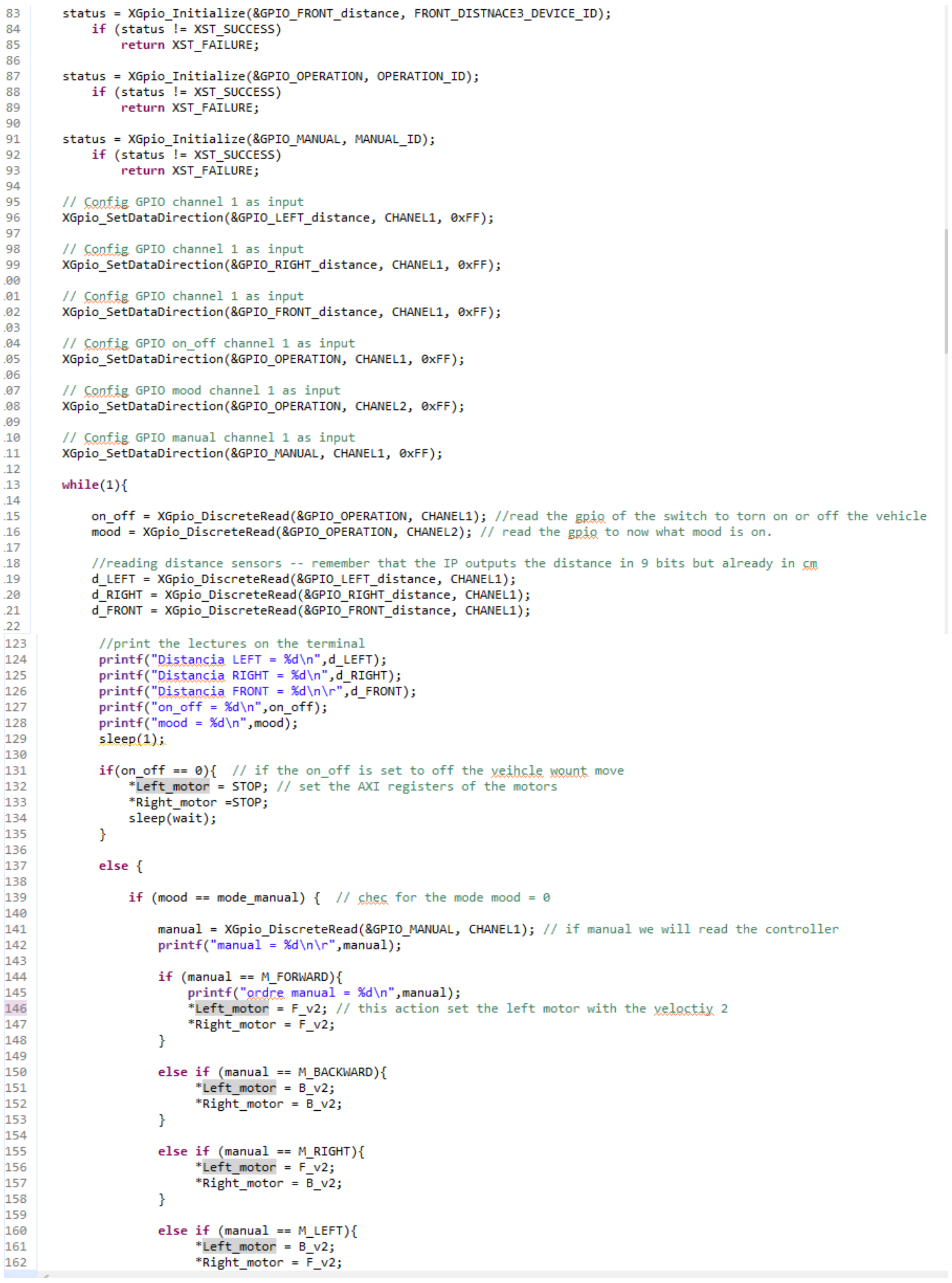


```vhdl
L293D_PWM3_v1_0_S00_AXI_inst : L293D_PWM3_v1_0_S00_AXI
    generic map (
        C_S_AXI_DATA_WIDTH  => C_S00_AXI_DATA_WIDTH,
        C_S_AXI_ADDR_WIDTH  => C_S00_AXI_ADDR_WIDTH
    )
    port map (

        PWM_OUT => PWM_OUT ,
        Forward => Forward ,
        Backward => Backward ,

        S_AXI_ACLK  => s00_axi_aclk,
        S_AXI_ARESETN   => s00_axi_aresetn,
        S_AXI_AWADDR    => s00_axi_awaddr,
        S_AXI_AWPROT    => s00_axi_awprot,
        S_AXI_AWVALID   => s00_axi_awvalid,
        S_AXI_AWREADY   => s00_axi_awready,
        S_AXI_WDATA => s00_axi_wdata,
        S_AXI_WSTRB => s00_axi_wstrb,
        S_AXI_WVALID    => s00_axi_wvalid,
        S_AXI_WREADY    => s00_axi_wready,
        S_AXI_BRESP => s00_axi_bresp,
        S_AXI_BVALID    => s00_axi_bvalid,
        S_AXI_BREADY    => s00_axi_bready,
```

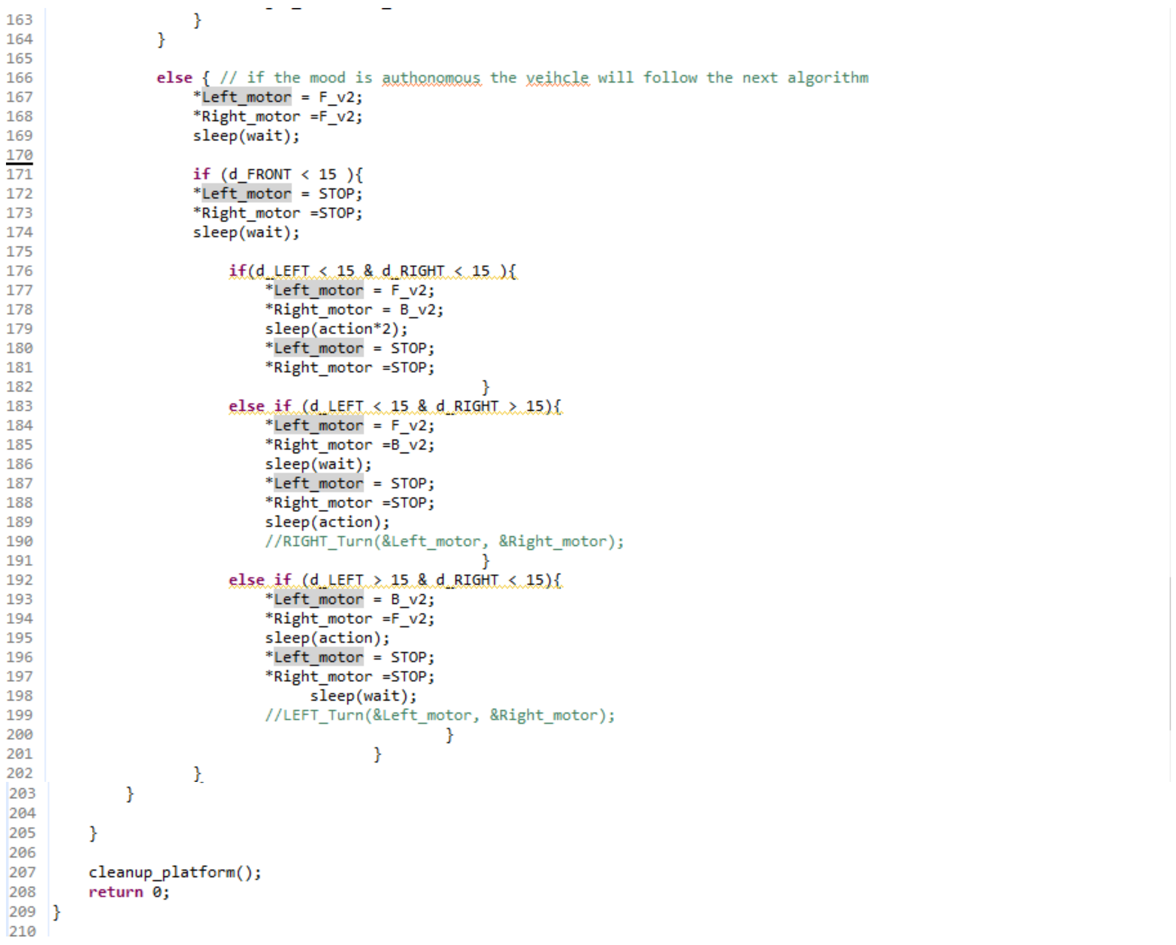
II. PS



```c

//libraris
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "xgpio.h"
#include "xparameters.h"
#include "xscugic.h"
#include "xil_exception.h"

#define LEFT_DISTNACE_DEVICE_ID          XPAR_AXI_GPIO_0_DEVICE_ID   //GPIO for LEFT distance sensor
#define RIGHT_DISTNACE2_DEVICE_ID        XPAR_AXI_GPIO_1_DEVICE_ID
#define FRONT_DISTNACE3_DEVICE_ID        XPAR_AXI_GPIO_2_DEVICE_ID
#define OPERATION_ID                     XPAR_AXI_GPIO_4_DEVICE_ID   //GPIO for mood(CH2) and on_off(CH1) sw
#define MANUAL_ID                        XPAR_AXI_GPIO_3_DEVICE_ID
#define CHANEL1 1
#define CHANEL2 2


// definig the velocities that the motors can reach remember that we are thinking in binary
// MSF represents the direction and the other 3 represents the gear (velocity)
#define STOP 0  //0000
#define F_v1 1  //0001
#define F_v2 2  //0010
#define F_v3 3  //0011
#define F_v4 4  //0100
#define F_v5 5  //0101

#define B_v1 9  //1001
#define B_v2 10 //1010
#define B_v3 11 //1011
#define B_v4 12 //1100
#define B_v5 13 //1101

// defining diferent orders that we can recive frome the manual mode , rmember we are thinking in binary
#define M_FORWARD   1   //0001
#define M_RIGHT     2   //0010
#define M_LEFT      4   //0100
#define M_BACKWARD  8   //1000

#define mode_manual 0

#define action 3
#define wait 2 // seconds of delay

// Pointer to access the Motor diriver software register 16 BITS
int *Left_motor = (int *) XPAR_L293D_PWM3_0_S00_AXI_BASEADDR;
int *Right_motor = (int *) XPAR_L293D_PWM3_1_S00_AXI_BASEADDR;

//declaration of XGpio objects to manage the gpio on the board
XGpio GPIO_LEFT_distance;
XGpio GPIO_RIGHT_distance;
XGpio GPIO_FRONT_distance;
XGpio GPIO_OPERATION;
XGpio GPIO_MANUAL;


int main()
{

    int d_FRONT;    // distance in the front sensor
    int d_RIGHT;    //distance in the right sensor
    int d_LEFT;     // distance in the left sensor
    int mood;       // to define manual or authonomous mood
    int on_off;     // to tun on or off the robot
    int manual;     // to save the operation from the manual radiocontrol
    int status;

    init_platform();

    printf("Hello\n\r");

    // initialization of the XGpio instances
    status = XGpio_Initialize(&GPIO_LEFT_distance, LEFT_DISTNACE_DEVICE_ID);
        if (status != XST_SUCCESS)
            return XST_FAILURE;

    status = XGpio_Initialize(&GPIO_RIGHT_distance, RIGHT_DISTNACE2_DEVICE_ID);
        if (status != XST_SUCCESS)
            return XST_FAILURE;

```

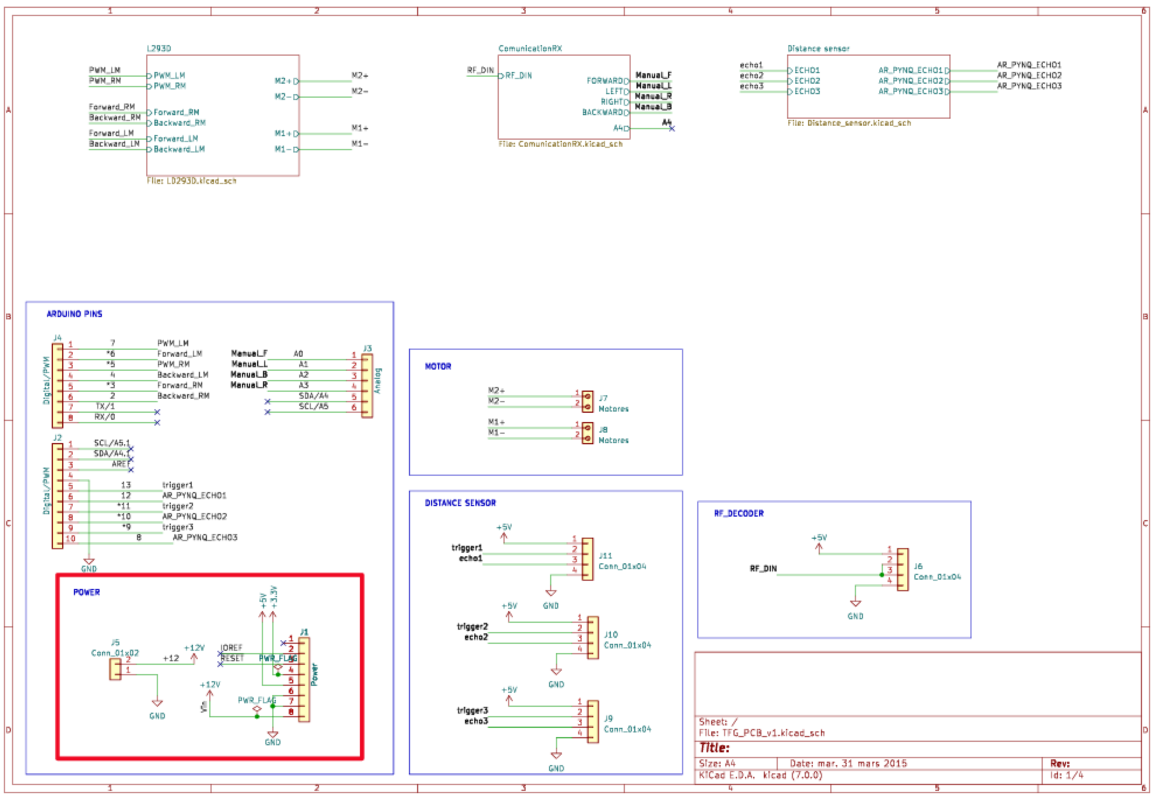


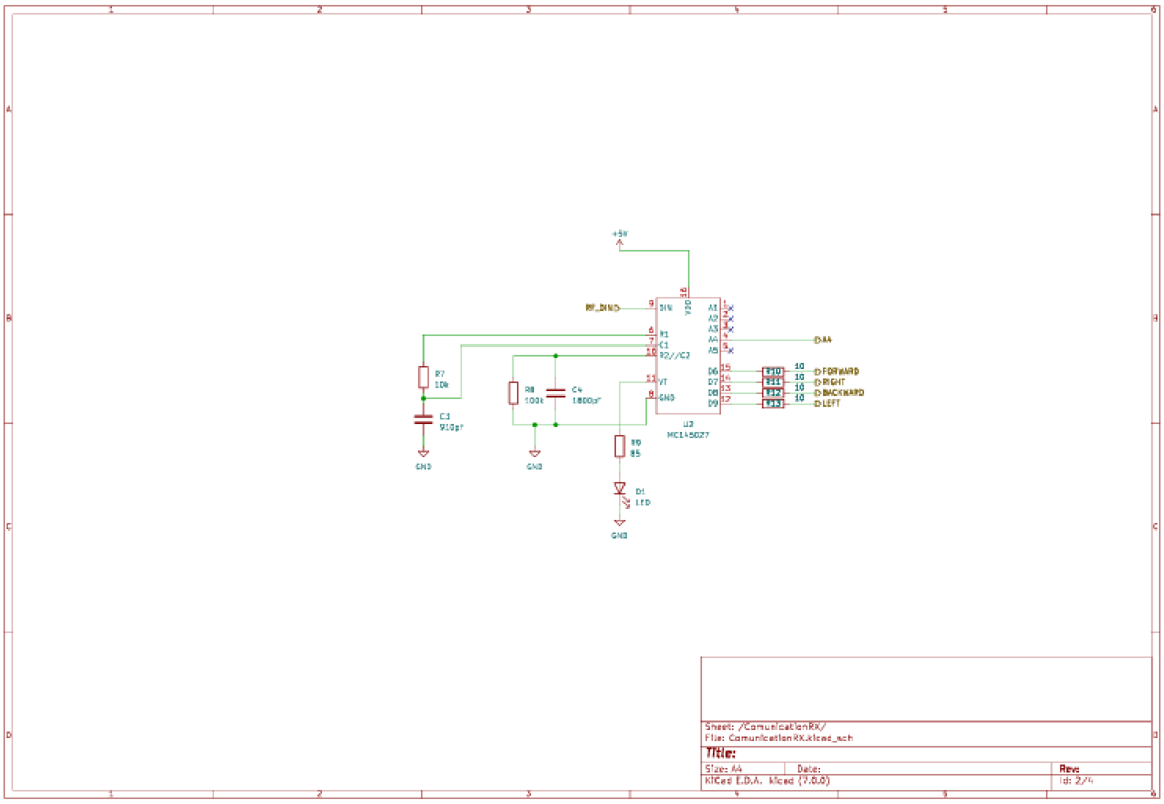
```c
    status = XGpio_Initialize(&GPIO_FRONT_distance, FRONT_DISTNACE3_DEVICE_ID);
        if (status != XST_SUCCESS)
            return XST_FAILURE;

    status = XGpio_Initialize(&GPIO_OPERATION, OPERATION_ID);
        if (status != XST_SUCCESS)
            return XST_FAILURE;

    status = XGpio_Initialize(&GPIO_MANUAL, MANUAL_ID);
        if (status != XST_SUCCESS)
            return XST_FAILURE;

    // Config GPIO channel 1 as input
    XGpio_SetDataDirection(&GPIO_LEFT_distance, CHANEL1, 0xFF);

    // Config GPIO channel 1 as input
    XGpio_SetDataDirection(&GPIO_RIGHT_distance, CHANEL1, 0xFF);

    // Config GPIO channel 1 as input
    XGpio_SetDataDirection(&GPIO_FRONT_distance, CHANEL1, 0xFF);

    // Config GPIO on_off channel 1 as input
    XGpio_SetDataDirection(&GPIO_OPERATION, CHANEL1, 0xFF);

    // Config GPIO mood channel 1 as input
    XGpio_SetDataDirection(&GPIO_OPERATION, CHANEL2, 0xFF);

    // Config GPIO manual channel 1 as input
    XGpio_SetDataDirection(&GPIO_MANUAL, CHANEL1, 0xFF);

    while(1){

        on_off = XGpio_DiscreteRead(&GPIO_OPERATION, CHANEL1); //read the gpio of the switch to torn on or off the vehicle
        mood = XGpio_DiscreteRead(&GPIO_OPERATION, CHANEL2); // read the gpio to now what mood is on.

        //reading distance sensors -- remember that the IP outputs the distance in 9 bits but already in cm
        d_LEFT = XGpio_DiscreteRead(&GPIO_LEFT_distance, CHANEL1);
        d_RIGHT = XGpio_DiscreteRead(&GPIO_RIGHT_distance, CHANEL1);
        d_FRONT = XGpio_DiscreteRead(&GPIO_FRONT_distance, CHANEL1);

        //print the lectures on the terminal
        printf("Distancia LEFT = %d\n",d_LEFT);
        printf("Distancia RIGHT = %d\n",d_RIGHT);
        printf("Distancia FRONT = %d\n\r",d_FRONT);
        printf("on_off = %d\n",on_off);
        printf("mood = %d\n",mood);
        sleep(1);

        if(on_off == 0){  // if the on_off is set to off the veihcle wount move
            *Left_motor = STOP; // set the AXI registers of the motors
            *Right_motor =STOP;
            sleep(wait);
        }

        else {

            if (mood == mode_manual) {  // chec for the mode mood = 0

                manual = XGpio_DiscreteRead(&GPIO_MANUAL, CHANEL1); // if manual we will read the controller
                printf("manual = %d\n\r",manual);

                if (manual == M_FORWARD){
                    printf("ordre manual = %d\n",manual);
                    *Left_motor = F_v2; // this action set the left motor with the veloctiy 2
                    *Right_motor = F_v2;
                }

                else if (manual == M_BACKWARD){
                     *Left_motor = B_v2;
                     *Right_motor = B_v2;
                }

                else if (manual == M_RIGHT){
                     *Left_motor = F_v2;
                     *Right_motor = B_v2;
                }

                else if (manual == M_LEFT){
                     *Left_motor = B_v2;
                     *Right_motor = F_v2;
```

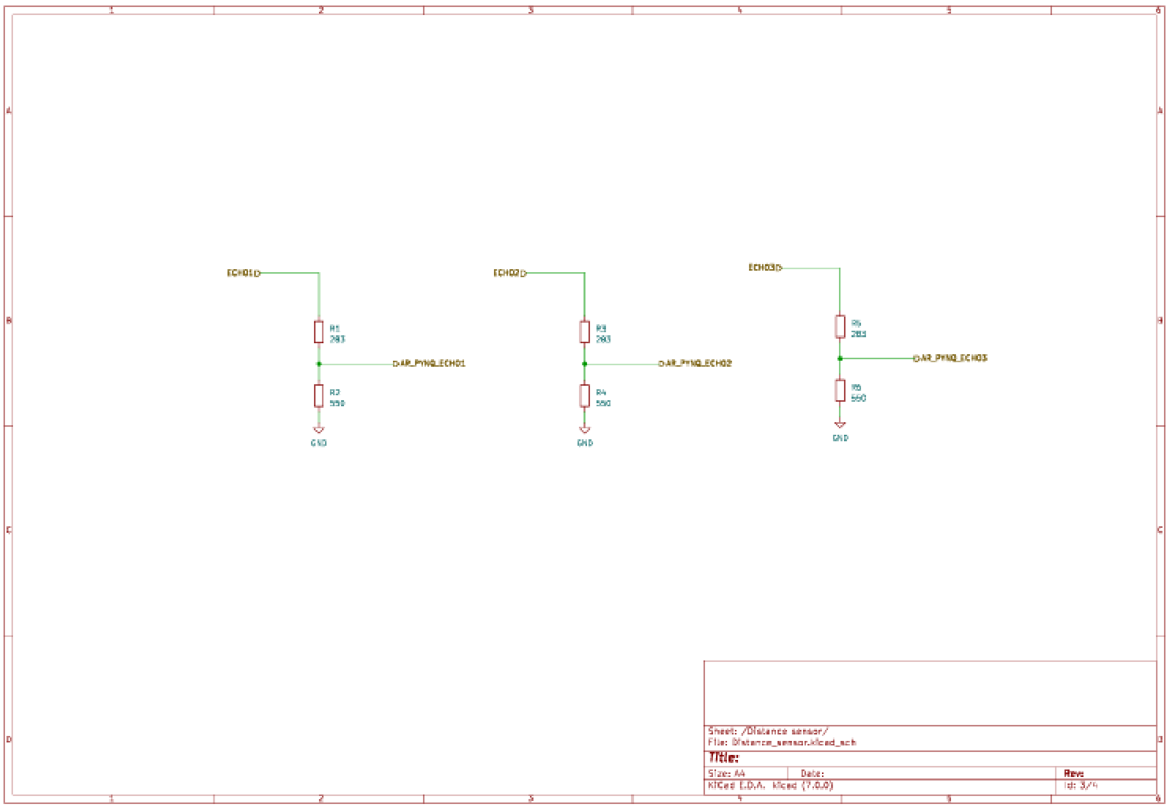


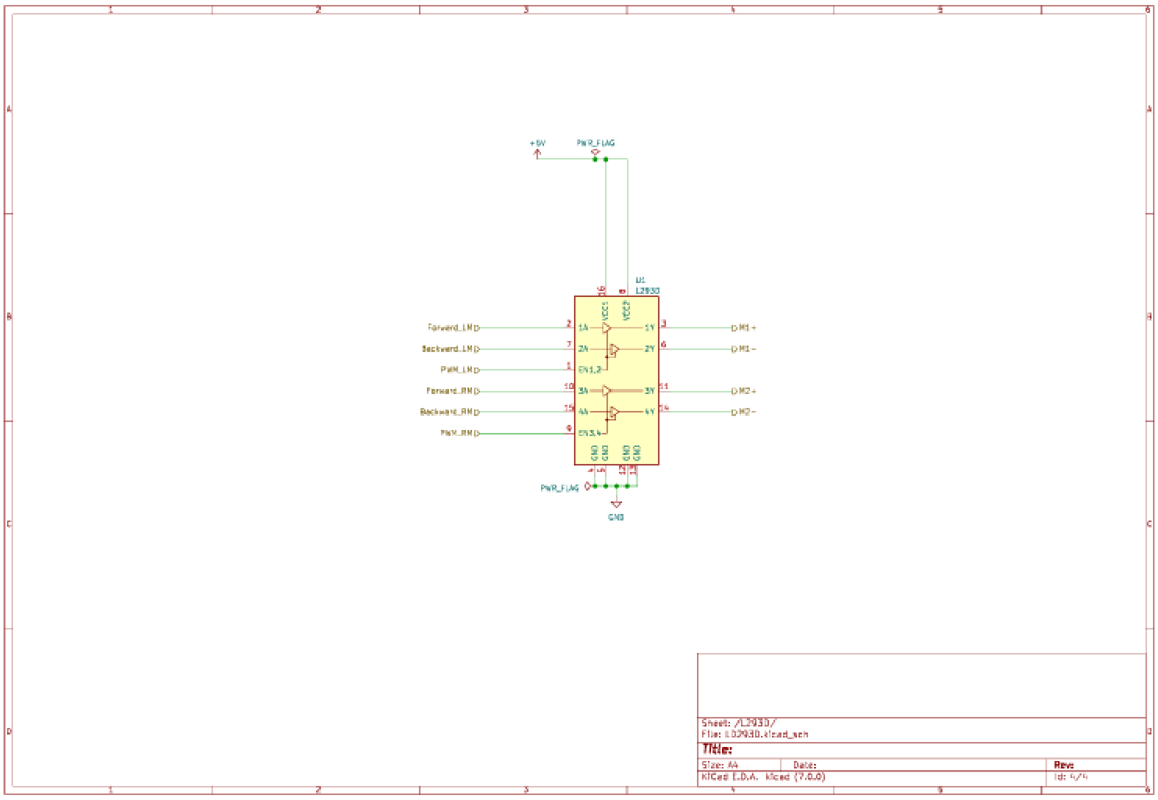
```c
                }
            }

            else { // if the mood is authonomous the veihcle will follow the next algorithm
                *Left_motor = F_v2;
                *Right_motor =F_v2;
                sleep(wait);

                if (d_FRONT < 15 ){
                *Left_motor = STOP;
                *Right_motor =STOP;
                sleep(wait);

                    if(d_LEFT < 15 & d_RIGHT < 15 ){
                        *Left_motor = F_v2;
                        *Right_motor = B_v2;
                        sleep(action*2);
                        *Left_motor = STOP;
                        *Right_motor =STOP;
                                                }
                    else if (d_LEFT < 15 & d_RIGHT > 15){
                        *Left_motor = F_v2;
                        *Right_motor =B_v2;
                        sleep(wait);
                        *Left_motor = STOP;
                        *Right_motor =STOP;
                        sleep(action);
                        //RIGHT_Turn(&Left_motor, &Right_motor);
                                                }
                    else if (d_LEFT > 15 & d_RIGHT < 15){
                        *Left_motor = B_v2;
                        *Right_motor =F_v2;
                        sleep(action);
                        *Left_motor = STOP;
                        *Right_motor =STOP;
                            sleep(wait);
                        //LEFT_Turn(&Left_motor, &Right_motor);
                                            }
                                    }
                }
        }

    }

    cleanup_platform();
    return 0;
}
```

III. Schematics

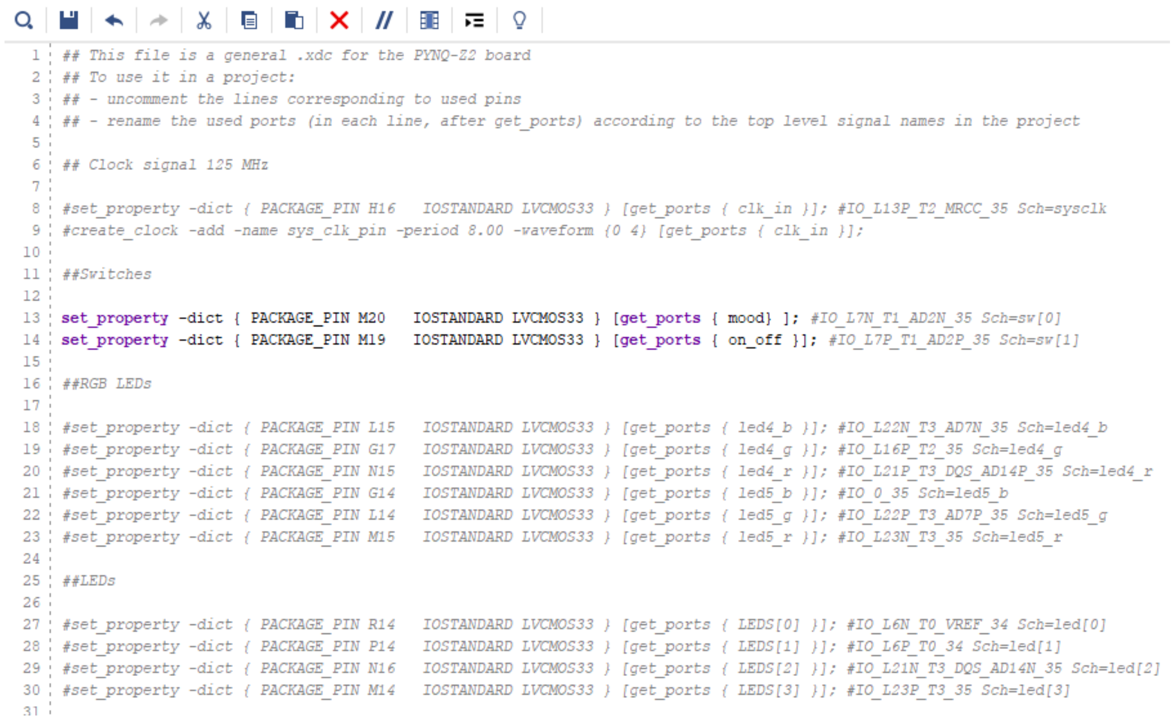




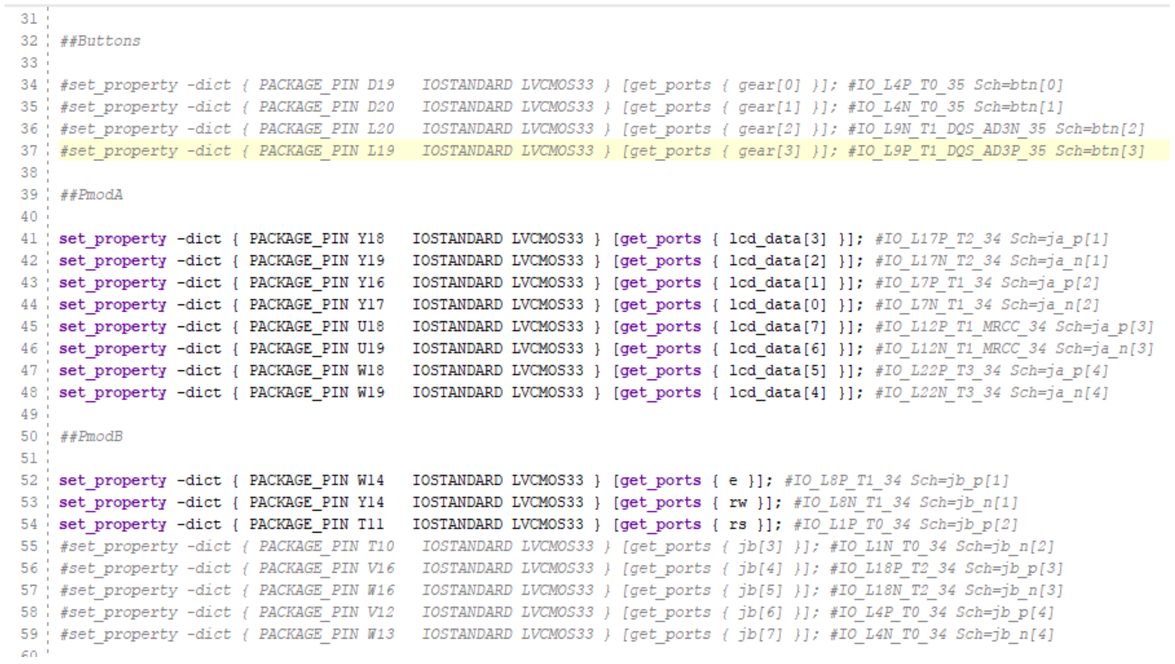

ECHO1
R1
283
AR_PYNQ_ECHO1
R2
550
GND
ECHO2
R3
283
AR_PYNQ_ECHO2
R4
550
GND
ECHO3
R5
283
AR_PYNQ_ECHO3
R6
550
GND
Sheet: /Distance sensor/
File: Distance_sensor.kicad_sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. kicad (7.0.0)
Id: 3/4


+5V
PWR_FLAG
U1
L293D
VCC1
VCC2
Forward_LM
Backward_LM
PWM_LM
Forward_RM
Backward_RM
PWM_RM
1A
2A
EN1,2
3A
4A
EN3,4
1Y
2Y
3Y
4Y
M1+
M1-
M2+
M2-
GND
PWR_FLAG
GND
Sheet: /L293D/
File: L293D.kicad_sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. kicad (7.0.0)
Id: 4/4

IV. Constrains



```
## This file is a general .xdc for the PYNQ-Z2 board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project

## Clock signal 125 MHz

#set_property -dict { PACKAGE_PIN H16   IOSTANDARD LVCMOS33 } [get_ports { clk_in }]; #IO_L13P_T2_MRCC_35 Sch=sysclk
#create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports { clk_in }];

##Switches

set_property -dict { PACKAGE_PIN M20   IOSTANDARD LVCMOS33 } [get_ports { mood} ]; #IO_L7N_T1_AD2N_35 Sch=sw[0]
set_property -dict { PACKAGE_PIN M19   IOSTANDARD LVCMOS33 } [get_ports { on_off }]; #IO_L7P_T1_AD2P_35 Sch=sw[1]

##RGB LEDs

#set_property -dict { PACKAGE_PIN L15   IOSTANDARD LVCMOS33 } [get_ports { led4_b }]; #IO_L22N_T3_AD7N_35 Sch=led4_b
#set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS33 } [get_ports { led4_g }]; #IO_L16P_T2_35 Sch=led4_g
#set_property -dict { PACKAGE_PIN N15   IOSTANDARD LVCMOS33 } [get_ports { led4_r }]; #IO_L21P_T3_DQS_AD14P_35 Sch=led4_r
#set_property -dict { PACKAGE_PIN G14   IOSTANDARD LVCMOS33 } [get_ports { led5_b }]; #IO_0_35 Sch=led5_b
#set_property -dict { PACKAGE_PIN L14   IOSTANDARD LVCMOS33 } [get_ports { led5_g }]; #IO_L22P_T3_AD7P_35 Sch=led5_g
#set_property -dict { PACKAGE_PIN M15   IOSTANDARD LVCMOS33 } [get_ports { led5_r }]; #IO_L23N_T3_35 Sch=led5_r

##LEDs

#set_property -dict { PACKAGE_PIN R14   IOSTANDARD LVCMOS33 } [get_ports { LEDS[0] }]; #IO_L6N_T0_VREF_34 Sch=led[0]
#set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports { LEDS[1] }]; #IO_L6P_T0_34 Sch=led[1]
#set_property -dict { PACKAGE_PIN N16   IOSTANDARD LVCMOS33 } [get_ports { LEDS[2] }]; #IO_L21N_T3_DQS_AD14N_35 Sch=led[2]
#set_property -dict { PACKAGE_PIN M14   IOSTANDARD LVCMOS33 } [get_ports { LEDS[3] }]; #IO_L23P_T3_35 Sch=led[3]
```



```

##Buttons

#set_property -dict { PACKAGE_PIN D19   IOSTANDARD LVCMOS33 } [get_ports { gear[0] }]; #IO_L4P_T0_35 Sch=btn[0]
#set_property -dict { PACKAGE_PIN D20   IOSTANDARD LVCMOS33 } [get_ports { gear[1] }]; #IO_L4N_T0_35 Sch=btn[1]
#set_property -dict { PACKAGE_PIN L20   IOSTANDARD LVCMOS33 } [get_ports { gear[2] }]; #IO_L9N_T1_DQS_AD3N_35 Sch=btn[2]
#set_property -dict { PACKAGE_PIN L19   IOSTANDARD LVCMOS33 } [get_ports { gear[3] }]; #IO_L9P_T1_DQS_AD3P_35 Sch=btn[3]

##PmodA

set_property -dict { PACKAGE_PIN Y18   IOSTANDARD LVCMOS33 } [get_ports { lcd_data[3] }]; #IO_L17P_T2_34 Sch=ja_p[1]
set_property -dict { PACKAGE_PIN Y19   IOSTANDARD LVCMOS33 } [get_ports { lcd_data[2] }]; #IO_L17N_T2_34 Sch=ja_n[1]
set_property -dict { PACKAGE_PIN Y16   IOSTANDARD LVCMOS33 } [get_ports { lcd_data[1] }]; #IO_L7P_T1_34 Sch=ja_p[2]
set_property -dict { PACKAGE_PIN Y17   IOSTANDARD LVCMOS33 } [get_ports { lcd_data[0] }]; #IO_L7N_T1_34 Sch=ja_n[2]
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { lcd_data[7] }]; #IO_L12P_T1_MRCC_34 Sch=ja_p[3]
set_property -dict { PACKAGE_PIN U19   IOSTANDARD LVCMOS33 } [get_ports { lcd_data[6] }]; #IO_L12N_T1_MRCC_34 Sch=ja_n[3]
set_property -dict { PACKAGE_PIN W18   IOSTANDARD LVCMOS33 } [get_ports { lcd_data[5] }]; #IO_L22P_T3_34 Sch=ja_p[4]
set_property -dict { PACKAGE_PIN W19   IOSTANDARD LVCMOS33 } [get_ports { lcd_data[4] }]; #IO_L22N_T3_34 Sch=ja_n[4]

##PmodB

set_property -dict { PACKAGE_PIN W14   IOSTANDARD LVCMOS33 } [get_ports { e }]; #IO_L8P_T1_34 Sch=jb_p[1]
set_property -dict { PACKAGE_PIN Y14   IOSTANDARD LVCMOS33 } [get_ports { rw }]; #IO_L8N_T1_34 Sch=jb_n[1]
set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 } [get_ports { rs }]; #IO_L1P_T0_34 Sch=jb_p[2]
#set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 } [get_ports { jb[3] }]; #IO_L1N_T0_34 Sch=jb_n[2]
#set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { jb[4] }]; #IO_L18P_T2_34 Sch=jb_p[3]
#set_property -dict { PACKAGE_PIN W16   IOSTANDARD LVCMOS33 } [get_ports { jb[5] }]; #IO_L18N_T2_34 Sch=jb_n[3]
#set_property -dict { PACKAGE_PIN V12   IOSTANDARD LVCMOS33 } [get_ports { jb[6] }]; #IO_L4P_T0_34 Sch=jb_p[4]
#set_property -dict { PACKAGE_PIN W13   IOSTANDARD LVCMOS33 } [get_ports { jb[7] }]; #IO_L4N_T0_34 Sch=jb_n[4]
```

```

##Arduino Digital I/O

#set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 } [get_ports {  }]; #IO_L5P_T0_34 Sch=ar[0]
#set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports {  }]; #IO_L2N_T0_34 Sch=ar[1]
set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33 } [get_ports { Backward_RM }]; #IO_L3P_T0_DQS_PUDC_B_34 Sch=ar[2]
set_property -dict { PACKAGE_PIN V13   IOSTANDARD LVCMOS33 } [get_ports { Forward_RM }]; #IO_L3N_T0_DQS_34 Sch=ar[3]
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports { Backward_LM }]; #IO_L10P_T1_34 Sch=ar[4]
set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 } [get_ports { PWM_RM }]; #IO_L5N_T0_34 Sch=ar[5]
set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports { Forward_LM }]; #IO_L19P_T3_34 Sch=ar[6]
set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports { PWM_LM }]; #IO_L9N_T1_DQS_34 Sch=ar[7]
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { echo3 }]; #IO_L21P_T3_DQS_34 Sch=ar[8]
set_property -dict { PACKAGE_PIN V18   IOSTANDARD LVCMOS33 } [get_ports { trigger3 }]; #IO_L21N_T3_DQS_34 Sch=ar[9]
set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports { echo2 }]; #IO_L9P_T1_DQS_34 Sch=ar[10]
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { trigger2 }]; #IO_L19N_T3_VREF_34 Sch=ar[11]
set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 } [get_ports { echo1 }]; #IO_L23N_T3_34 Sch=ar[12]
set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports { trigger1 }]; #IO_L23P_T3_34 Sch=ar[13]
#set_property -dict { PACKAGE_PIN Y13   IOSTANDARD LVCMOS33 } [get_ports { a }]; #IO_L20N_T3_13 Sch=a

##Arduino Digital I/O On Outer Analog Header
##NOTE: These pins should be used when using the analog header signals A0-A5 as digital I/O

set_property -dict { PACKAGE_PIN Y11   IOSTANDARD LVCMOS33 } [get_ports { Manual[0] }]; #IO_L18N_T2_13 Sch=a[0]
set_property -dict { PACKAGE_PIN Y12   IOSTANDARD LVCMOS33 } [get_ports { Manual[1] }]; #IO_L20P_T3_13 Sch=a[1]
set_property -dict { PACKAGE_PIN W11   IOSTANDARD LVCMOS33 } [get_ports { Manual[2] }]; #IO_L18P_T2_13 Sch=a[2]
set_property -dict { PACKAGE_PIN V11   IOSTANDARD LVCMOS33 } [get_ports { Manual[3] }]; #IO_L21P_T3_DQS_13 Sch=a[3]
set_property -dict { PACKAGE_PIN T5    IOSTANDARD LVCMOS33 } [get_ports { res }]; #IO_L19P_T3_13 Sch=a[4]
set_property -dict { PACKAGE_PIN U10   IOSTANDARD LVCMOS33 } [get_ports { res2 }]; #IO_L12N_T1_MRCC_13 Sch=a[5]

```