ESCUELA SUPERIOR DE INGENIERÍA

COMPUTER ENGINEERING

FPGA BASED EMBEDDED SYSTEM TO CONTROL AN ELECTRIC VEHICLE AND THE DRIVER ASSISTANCE SYSTEMS

Mario Márquez Luciano

June 5, 2014

ESCUELA SUPERIOR DE INGENIERÍA

COMPUTER ENGINEERING

FPGA BASED EMBEDDED SYSTEM TO CONTROL AN ELECTRIC VEHICLE AND THE DRIVER ASSISTANCE SYSTEMS

- Departament: Systems and Automation Engineering, Electronic and Electronic Technology
- Supervisor: Mª Ángeles Cifredo Chacón
- Author: Mario Márquez Luciano

Cádiz, June 5, 2014

Mario Márquez Luciano

Aknowledgments

I would like to aknowledge the continuous support of my family during my years of student. Also I would like to thank the people of VDE group for giving me the oportunity of working in the E-Performance Kart. Of course I have to say thank you to my supervisors at the Karlsruhe Institut of Technoly, Michael Dreschmann and Falco Bapp, and to my supervisor at the University of Cádiz, Maria de los Ángeles Cifredo Chacón. And last but not less important to all my lifelong friends and to those who shared with me an unforgettable Erasmus experience in Karlsruhe.

License

List of Abbreviations

ANSI	American National Standards Institute
API	Application Programming Interface
ASIC	Application-specific Integrated Circuit
AXI	Advanced eXtensible Interface
CAN	Controller Area Network
CU	Control Unit
DC	Death Current
DSP	Digital Signal Processor
EDK	Embedded Development Kit
FPGA	Field Programmable Gate Array
IPC	Industrial Personal Computer
IRT	Isochronous Real-Time
ISE	Integrated Software Environment
ITIV	Institut für Technik der Informationsverarbeitung
KIT	Karlsruher Institute für Technologie
LED	Light-emitting Diode
LUT	Look-up Table
PLB	Processor Local Bus
POSIX	Portable Operating System Interface
RAM	Random Acess Memory
RPM	Revolutions per minute
SDK	Software Development Kit
TCS	Traction Control System
VDE	Verband der Elektrotechnik
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Several competitions deal with the area “electrification” and many engineers are currently researching this topic. In this background, the electric automobile exhibition in Aschaffenburg, Germany, takes place at the same time as a contest to award the best idea and implementation of electrification. This competition is aimed at students and student groups who have made a project in their own work. It can be an existing vehicle to be converted to electric drive or a whole new can be created. They will be announced alongside with the latest technical ideas in area electrification and tested in a 50 meters acceleration sprint [1].

Over the course of the past two and a half years and in collaboration with various institutes of the electrical and mechanical engineering faculties, the VDE [1] converted a go-kart to have a full- electrical engine. The project focuses on sportive and dynamic driving characteristics. Thus, a racing kart was chosen as platform to implement various applications of high-profile sports racing, e.g. torque vectoring and a traction control system. The team was able to complete a sprint race victoriously in fall of 2012.

This project shows that electric mobility is dynamic and fun and that having fun and to preserve costs and resources are not mutually exclusive. Many disadvantages of the conventional combustion engine don’t play a role in the E-mobility (revs, turbo lag, oil changes) [1]. Because of this it is said that E-mobility is the future of the automotive industry and for testing purpouses racing is the best laboratory in order to transfer this advantages to street cars.

1.2 Motivation

Despite the kart has won the contest in his first participation in the Aschaffenburg exhibition and all the control systems of the kart work perfectly, the VDE group aims to improve the performance of their kart and because of this the present Master Thesis takes place. But not only improving the performance in terms of pure speed is important in this kind of project but also improving the overall efficiency of the system by developing smaller hardware, more

[1] German electrical engineering association

efficient software and making them more generic, manufacter independent and easy to use.
This Master Thesis could be seen as a perfect example of what hardware-software codesign is.

1.3 Starting Point

At the beginning of this Master Thesis, the VDE E-Kart was delivered as he had won the competition with the Siemens control unit, motors and converters, which means that it was necesary an initial analysis in order to gather requirements for the new system.

The vehicle's central data node, hosting the traction control system and remaining control implementations, is an industrial PC by SIEMENS (IPC 427C), running the real-time operating system RMOS3. To reduce weight while achieving equal or possibly higher processing speed, the IPC will be replaced by a custom FPGA board developed at ITIV institute for hosting the implementation of an embedded system including the software control part.
The board was in development in August 2013 and after exhaustive tests was delivered ready for use.

1.4 Problem definition

The main aim of this work is the design and implementation of an embedded system and his software part in order to replace the existing Siemens IPC unit, solving the problems described in the previous section and adding new functionalities to the system. All these new functionalities are developed to be the new basis control electronic for a future new version of the kart for long runs instead of short sprint races, which is going to be developed and constructed in the next years. The development of this Master Thesis has been divided in the following different tasks:

- Sensor data processing
- Acquisition and processing of all four wheel speeds for use in vehicle dynamics calculations: traction control and torque vectoring
- Acquisition, processing and output of further analog and digital signals, for instance throttle, steering, acceleration and dashboard switches
- Implementation of traction control system and torque vectoring using VHDL
  - Direct VHDL coding based on matlab models
- Lightweight, FPGA-adjusted implementation
  - Implementation of peripherals in VHDL to control the digital inputs and outputs and analog inputs
  - Standard hardware interface for the peripherals, which simplifies the process of adapting them to any type of architecture and buses (PLB, Wishbone, AXI)

  - Standard and plattform independent drivers for all the peripherals
  - Equal or better performance than the previous system
- Structuring and simplification of code
  - Use of a simpler and faster programming language (C instead of C++) to write low level drivers and control tasks
  - Change of the Programming paradigm from Object Oriented Programming to Procedural Programming
  - Abstraction layer for the peripherals's firmware in order to get plattform independence
  - Well documented source code using documentation tools (Doxygen)
- Connection of converters (power electronics devices for motor control) via CAN bus
- Visualization and documentation of results

1.5 Schedule

In this section the different phases of this thesis are explained in detail and also a schedule plan is given.

1.5.1 System analysis

In this stage, the actual system was analyzed, which means both Hardware and Software and all the requirements (hardware and software) for the future system were gathered. The next three stages compose the development of the system itself and were parallel executed.

1.5.2 Hardware development

In this stage, the embedded system including all custom the components were developed based on the gathered requirements of the previous stage. This stage is known as the Hardware/Software codesign.

1.5.3 Software development

The firmware of the developed peripherals and the control software were developed at this stage. The firmware development took place right after the hardware design worked properly in the simulator meanwhile the control software development had to wait until the hardware platform was tested and stable.

1.5.4 Verification process and tests

All the developed components were tested exhaustively using the avaible tools for that, in this case the hdl simulator and custom test benchs. This stage took place parallel with both hardware and software development stages and was necesary in order to debug and improve the developed components.

1.5.5 Gantt chart

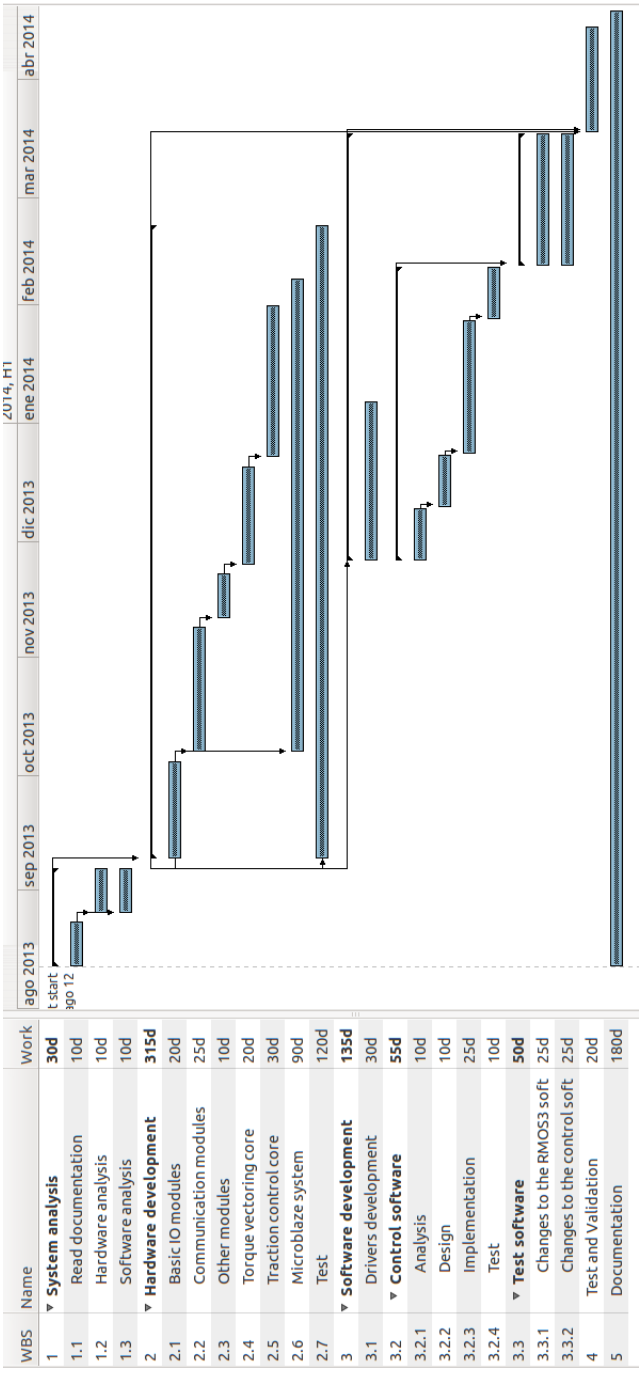

Figure 1.1: Gantt chart

Figure 1.1 shows the Gantt chart of this Thesis including all the previously explained phases. As can be seen, the total duration was 8 months, beginning in mid-August 2013 and ending in mid-April 2014.

Chapter 2

The VDE KART

In this chapter, the VDE-Kart and all of the technologies involved are described so as to have a general idea of the requirements for the future system developed in this Master Thesis, which is going to replace the actual system. All these new functionalities are developed to be the new basis control electronic for a future new version of the kart for long runs instead of short sprint races, which is going to be developed and constructed in the next years. This car will have similar features, but adapted to the requirements of long runs (autonomy, power, gear ratio), compared with the sprint version of the kart, which is described in this chapter.

2.1 Overview

The choice of the vehicle platform, the drive concept and parts of the vehicle's structure was the target of a previous Bachelor Thesis. Due to the chassis' low weight and its maneuverability, given by the smaller wheels, it was decided to choose a kart platform for the electrification project. In addition, due to the low center of gravity and the fact that the construction of a kart allows for greater wheel loads on the rear axle, a kart platform has a lot of advantages for sprint races [2].

It has two electric drivetrains, each of them consisting of an electric motor and the necessary converter. Each drivetrain unit (Motor + Converter) is governed by a dedicated Siemens control unit. This control unit receives the necesary control data from a Siemens Industrial PC (IPC). Overall, the kart weighs about 166kg (without driver) and is equipped with similar tyres as the Formula Student cars. The motors are controlled by a fast and robust industrial PC [2].

Figures 2.1 and 2.2 shows how the E-Kart is but in order to get a more detailed view of the different components of the kart, they are described individualy in the next sections.

2.2 Main components

2.2.1 Siemens IPC 427

The complete vehicle is controlled by a robust Siemens Industrial PC 427 (see Figure 2.3). The Siemens IPC 427C monitors and controls the kart. He has 24 digital inputs, 16 digital outputs,

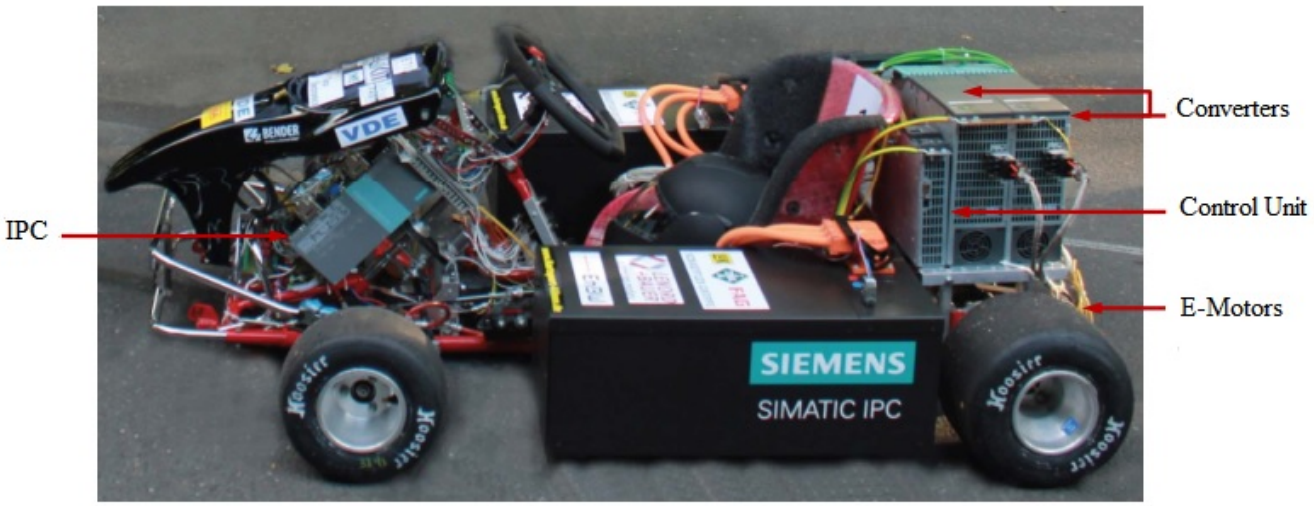

Figure 2.1: VDE E-Performance Kart

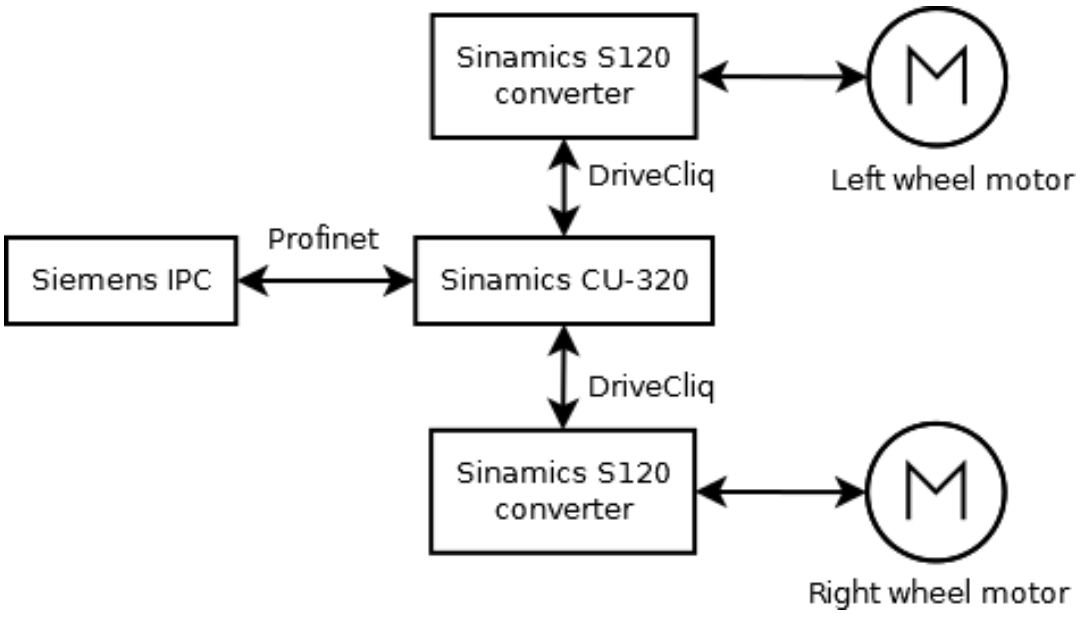

Figure 2.2: VDE E-Performance Kart System block diagram

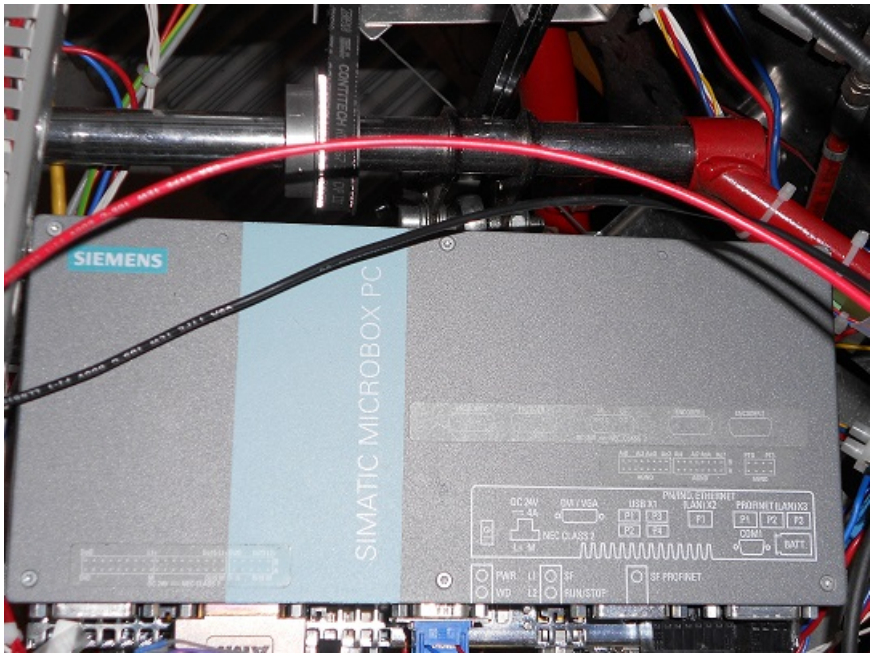

Figure 2.3: Siemens IPC 427

8 analog inputs and outputs. In addition the IPC has 4 ports for rotary encoders for wheel's speed meassuring, RS-232 serial port and Ethernet ports, which are used for Profinet protocol and programming the IPC via ftp server.

The Siemens IPC is designed for real time tasks and in fact it has installed the real time operating system RMOS v3.50. It is suited to update all the signals every 50 μs so the performance is guaranteed.
In addition, every 250 microseconds, new values such as measurements of road tests are stored in the RAM memory of the IPC, which can be recovered later via the serial interface. Between the IPC and the control of the converters works the Profinet IRT protocol [2].

2.2.2 Motors

The kart is powered by two water-cooled permanent-magnet synchronous motors manufactured by Siemens. The Siemens motors 1FE1-042 deliver , (as can be seen in Figure 2.4), a nominal power output of 14.4 kW and a rated torque of 11 Nm, which can be up to 20 Nm in overload mode (operating mode S6-40%). The motors can reach a maximum of 18000 RPM.
The overload capability is used in the race as a boost function and is controlled in the software [2].

Figure 2.4 shows the setup of the motors in the kart. As it can be seen, each of one is conected to an independent axle, which gives through a gearbox the power to each wheel. The gearbox is designed to achieve 140 Km/h at 18000 RPM, the maximum engine speed.

For more information about de electric motors, a detailed datasheet is provided in the Appendix.

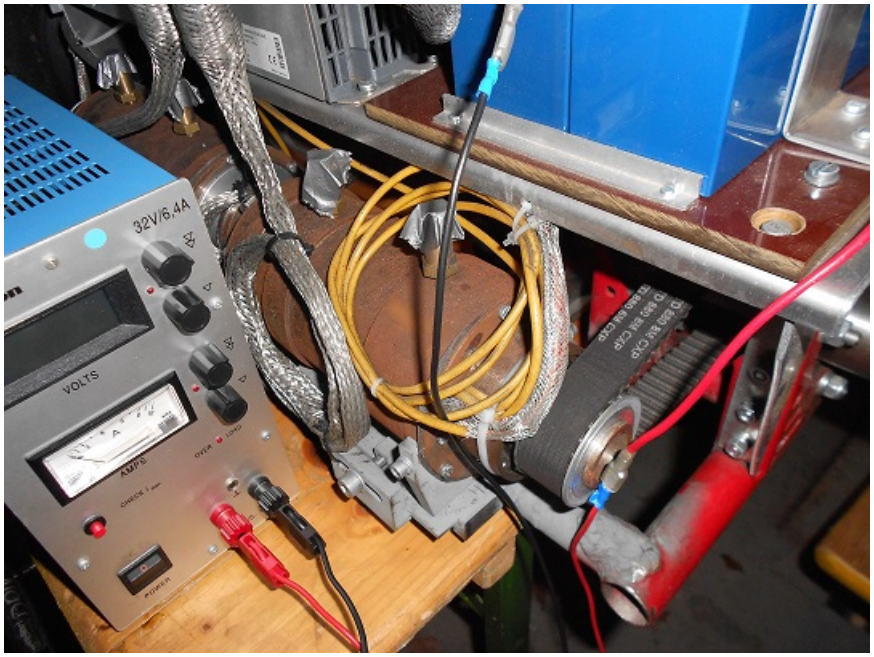

Figure 2.4: Siemens electric Motors

2.2.3 Converters

There are two SINAMICS Siemens S120 frequency converters installed in the vehicle, which are controlled by one Siemens Control Unit. Each of one controls one motor independently and the communication between the converters and the control unit is made via the Siemens "DriveCliq" protocol, which are not open published [2].

Figure 2.5 shows the converters' placement, which is marked in red colour.

2.2.4 Control units

The Siemens Control Unit C320 is used to control the converters. The exchange of information between the CU and IPC occurs every 500 μs. This exchange of information is made through the profinet IRT protocol and by this way, the controller can modify the values of the accelerator pedal and wheel speeds fast enough to ensure a good control of the motors [2].

Figure 2.5 shows the control unit's placement, which is marked in yellow colour.

2.2.5 Wheel speed sensors

Is a prerequisite for the optimal work of the Traction Control System to keep the readout time of the sensor in the range below 1 ms. Further processing of the signals should be in the range below 5 to 10 ms and this condition was fulfilled successfully via using an inductive sensor of the company "Lenord+Bauer". The only limitation which has been found is that the sensor is only accurate from speeds above 10 km/h. Therefore, all the calculations concerning wheel

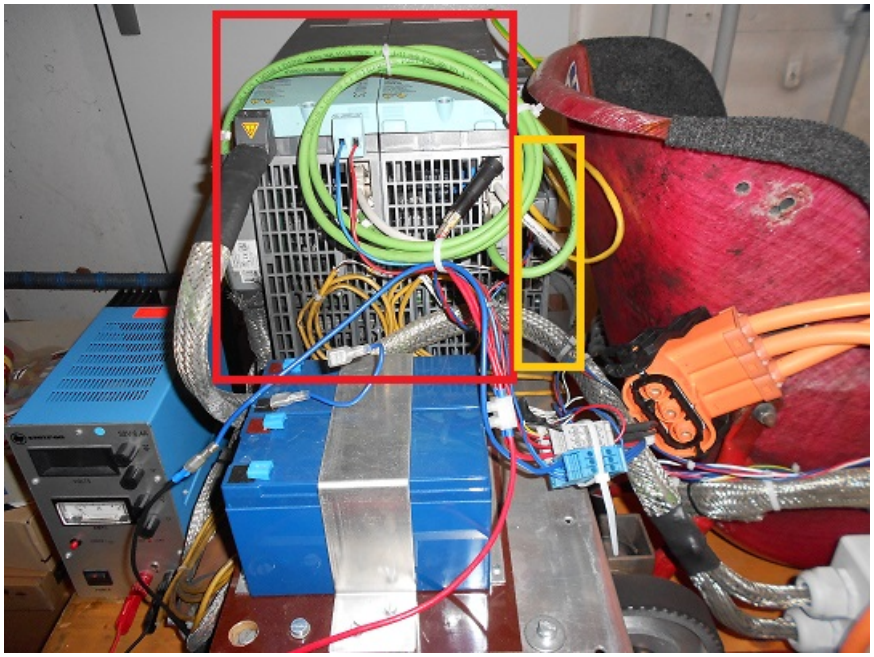

Figure 2.5: Siemens converters and control unit

speeds must be executed only with wheel speeds above 10 km/h [2].

Figure 2.6 shows the Speed sensor's placement marked in red colour.

2.2.6 Supercapacitors

In order to provide the needed electrical energy for the motors, two supercapacitors store this electrical energy and releases it when needed. They can be charged to a maximum of 380 Volts and at maximum power, the stored energy is enough to move the kart for the required 50 m sprint and a little bit more for the return to the start place, for a total of barely 100 meters of autonomy at maximum performance. For further information see [3].
Figure 2.7 shows the internal construction of the supercapacitors.

2.2.7 Safety

Current and voltage

The main danger of this Kart is his DC link voltage of 600V, which increases in the intermediate circuit when braking. This high voltage requires special efforts to protect the driver and bystanders against electric shocks. The used components are protected against contact by applicable standards, so emerging risks of this interconnection are intercepted by safety circuits. During construction, measures have been taken to prevent that the user comes into contact with electrical voltage. Furthermore, an emergency shutdown system was installed, so at any time the energy storage can be separated from the remaining components of the kart and the DC link

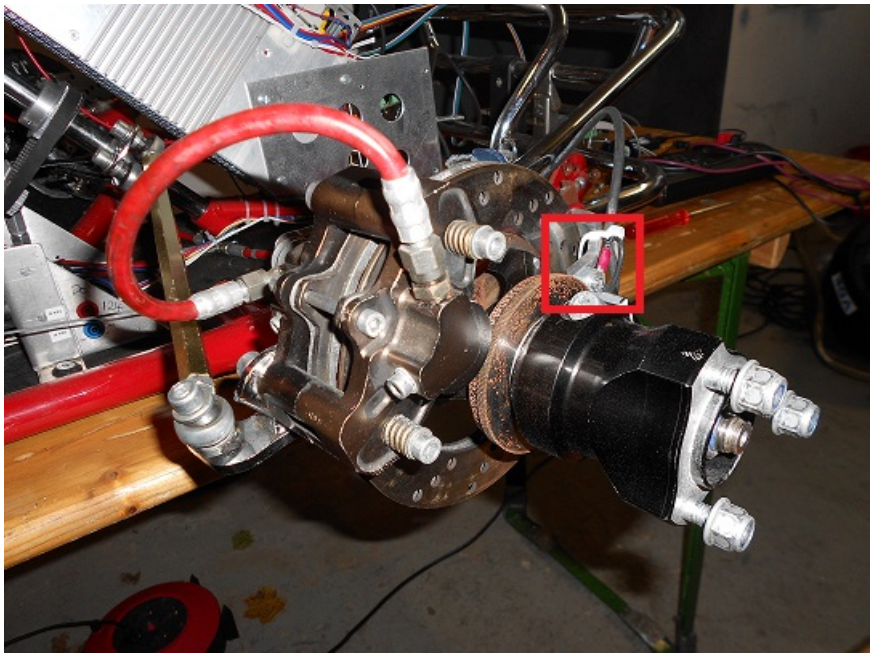

Figure 2.6: Lenor+Bauer speed sensor

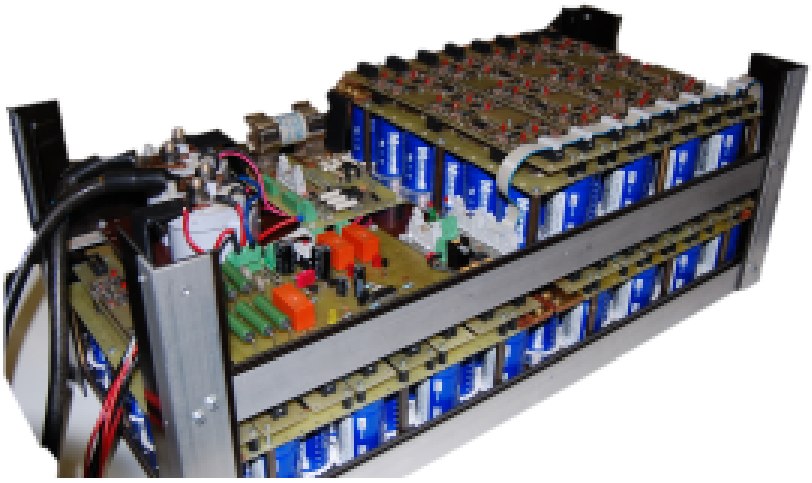

Figure 2.7: E-Kart supercapacitors

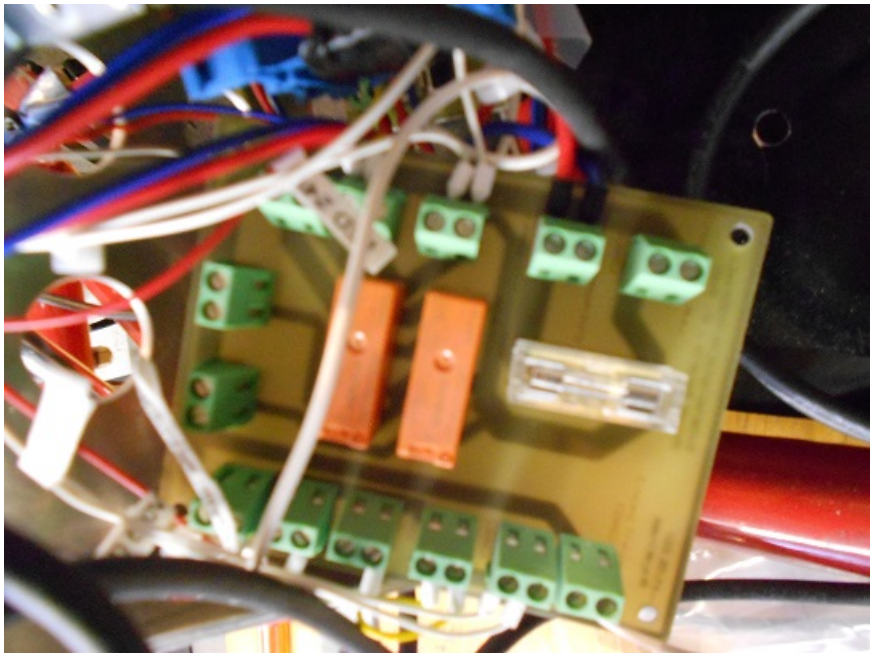

Figure 2.8: Custom security board

is discharged quickly. All components have been designed for long-term exposure to 5g forces.

Figure 2.8 shows the security custom board.

Kid's safe drive mode

The key switch "'VMAX" on the panel is used to protect inexperienced drivers. By using the Profinet communication, a speed limit of 50 Km/h should be sent to the CU320, corresponding to an engine speed of 6400 RPM. In addition, the torque vectoring is fixed to factor 0.2 (see 2.1), the braking force distribution and the response at Level 1 (see 2.2). The ASR should be active.

2.2.8 Information leds

There is the LED panel in the cockpit right in front of the driver's view, which allows the driver to have at any time information about the different operating conditions and errors. Figure 2.9 shows the position of the different led lights in the cockpit [3].

This LED panel gives information about:

- 24V power supply on/off
- 600V power supply on/off
- Battery almost empty

- Battery empty
- PC error
- Motors hot/error

Figure 2.9: LEDs panel

2.3 Functions

2.3.1 Torque Vectoring

Torque vectoring is a new technology employed in automobile differentials. A differential transfers engine torque to the wheels. Torque vectoring technology provides the differential with the ability to vary the power to each wheel. This method of power transfer has recently become popular in all-wheel drive vehicles. Some newer front-wheel drive vehicles also have a basic torque vectoring differential. As technology in the automotive industry improves, more vehicles are equipped with torque vectoring differentials.

The torque vectoring idea builds on the basic principles of a standard differential. A torque vectoring differential performs basic differential tasks while also transmitting torque independently between wheels. This torque transferring ability improves handling and traction in almost any situation [4] [5].

Torque vectoring in constant corner drive

The torque vectoring function in executed in the control unit as a mathematical formula, which gives the percentage of torque that must be substracted to the interior wheel and added to the exterior wheel.

Equation 2.1 models the behaviour of the torque vectoring [2]:

$$\Delta MTorqueVectoring = v^2 \cdot \sin^2 \delta_l \cdot d \cdot 1.85736 \tag{2.1}$$

where v is the kart's speed in m/s, δ_l is the steering angle in radians and d is factor which changes the sensibility of the torque vetoring. Table 2.1 shows the possible values of the "d"

factor:

Torque vectoring switch	Factor d
5/5 (off)	0
6/4	0.2
7/3	0.3
8/2	0.4
9/1	0.5

Table 2.1: Torque Vectoring Factor

The torque vectoring does not work in case of an error in the selector switch and in case of Kid's safety function active the factor is set to 0.2. In both cases the error LED is activated.
In order to know more about the calculations involving the torque vectoring function see [3].

Dynamic Torque Vectoring in the corner entry

At the transition from a straight line into a circular orbit and back special dynamics bring advantages in car racing. The faster the vehicle follows the specified steering movements, the sportier is the driving experience. Therefore, now when turning, the steering wheel selector "response" broughts into play.

In the actual corner entry, while the steering angle is still changing, a higher weighting factor "f" for the torque distribution is introduced to force the vehicle quicker to the desired circular path. If the kart returns to the constant steering angle, it changes again to the Torque Vectoring formula with the factor "d" (transition from corner entry to constant cornering).

Table 2.2 shows the possible values of the "f" factor:

Response	Factor f
1	0
2	0.2
3	0.4
4	0.6
5	0.8
6	1.0

Table 2.2: Torque Vectoring Response

2.3.2 Traction Control System

Critical driving situations that arise especially on slippery road surfaces can cause that the driver responds incorrectly and the vehicle's behavior is unstable. The Traction Control System con-

cerns the start and acceleration of the vehicle and guarantees a stable behavior, securing that the physical limits are not exceeded.

The Traction Control System monitors the rotational speed of the wheels using wheel speed sensors. By comparing the driven and non-driven wheels, the system determines whether a wheel is spinning or not and calculates the slip. If the slip is too high, the torque is reduced und thus the slip is also reduced [2].

2.3.3 Brake balance

The VDE E-Kart has a brake balance switch, that allows the driver to change the behaviour of brakes in that displacing more brake power to rear train or to front according to the driver's demand at any time.

2.3.4 Offline telemetry

At car racing it is very important to have as many as possible information about all the systems of the car. This information can be transmited in real time from the car to the engineers, which is known as Live-Telemetry, and it is very usefull to do some checks in the car and if the telemetry is bidirectional make changes in real time to improve the performance or tell the driver if something is going wrong.
At the moment there is not Live-Telemetry in the VDE E-Kart but all the information is stored in the computer's RAM and can be retrieved via serial port when the kart has finished his run.

The most common telemetry information stored is:

- Wheel speeds and global speed
- Gas and brake percentage
- Lateral and vertical G-forces
- Battery charge
- Motor temperatures

Chapter 3

Involved Technologies

In this chapter the main technologies involved in this Master Thesis are explained.

3.1 FPGA

A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by a customer or a designer after manufacturing, hence "field-programmable". The FPGA configuration is generally specified using a hardware description language (HDL), similar to that used for an application-specific integrated circuit (ASIC) (circuit diagrams were previously used to specify the configuration, as they were for ASICs, but this is increasingly rare).

Contemporary FPGAs have large resources of logic gates and RAM blocks to implement complex digital computations. As FPGA designs employ very fast I/Os and bidirectional data buses it becomes a challenge to verify correct timing of valid data within setup time and hold time. Floor planning enables resources allocation within FPGA to meet these time constraints [6]. FPGAs can be used to implement any logical function that an ASIC could perform. The ability to update the functionality after shipping, partial re-configuration of a portion of the design and the low non-recurring engineering costs relative to an ASIC design (notwithstanding the generally higher unit cost), offer advantages for many applications.

FPGAs contain programmable logic components called "logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together" somewhat like many (changeable) logic gates that can be inter-wired in (many) different configurations. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory [7].

Figure 3.1: Xilinx's FPGA chip

3.2 ITIV's custom Xilinx Spartan 6 board for VDE Kart

A custom FPGA board developed at ITIV institute for hosting the implementation of an embedded system including the software control part was chosen as the replacement for the old Siemens computer.
The board, which is equiped with a Spartan 6 FPGA, was in development in August 2013 and after exhaustive tests was delivered ready for use.

3.2.1 Spartan 6 family

The Spartan-6 family provides leading system integration capabilities with the lowest total cost for high-volume applications. The thirteen-member family delivers expanded densities ranging from 3,840 to 147,443 logic cells, with half the power consumption of previous Spartan families, and faster, more comprehensive connectivity. Built on a mature 45 nm low-power copper process technology that delivers the optimal balance of cost, power, and performance, the Spartan-6 family offers a new, more efficient, dual-register 6-input lookup table (LUT) logic and a rich selection of built-in system-level blocks.

These include 18 Kb (2 x 9 Kb) block RAMs, second generation DSP48A1 slices, SDRAM memory controllers, enhanced mixed-mode clock management blocks, SelectIO technology, poweroptimized high-speed serial transceiver blocks, PCI Express compatible Endpoint blocks, advanced system-level power management modes, auto-detect configuration options, and enhanced IP security with AES and Device DNA protection. These features provide a lowcost programmable alternative to custom ASIC products with unprecedented ease of use. Spartan-6 FPGAs offer the best solution for high-volume logic designs, consumer-oriented DSP designs, and cost-sensitive embedded applications. Spartan-6 FPGAs are the programmable silicon foundation for Targeted Design Platforms that deliver integrated software and hardware components that enable designers to focus on innovation as soon as their development cycle begins

[12].

Spartan 6 lx150

The FPGA model, that was chosen for the custom board was the spartan 6 lx150 and in Table 3.1 the main features of this model are enumerated:

Logic cells	147,443
Slices	23,038
Flip-Flops	184,304
Max distributed RAM	1,355 Kb
DSP48A1 Slices	180
Block RAM 18 Kb	268
Max RAM	4,824 Kb
CMT(2 DCMs + PLL)	6
Memory Controller Blocks	4
Max user I/O	576

Table 3.1: Spartan 6 lx150 features

3.2.2 Board features

The custom FPGA based board developed in the ITIV, which is going to replace the present Siemens industrial PC, has de following features:

- Power suply between 12 and 24 Volts
- 8 analog inputs with 12 bits precission (AD7927) in range of 0 to 10 Volts
- 24 digital inputs in range of 0 to 24 volts
- 24 digital outputs in range of 0 to 24 volts with a maximum frequency of 2 KHz
- 4 RS-422 interfaces
- 2 CAN bus interfaces
- 2 RS-485 interfaces
- Ethernet interface
- GSM interface
- 2 x 512 Mb SDRAM

Table 3.2 resumes the elements of the onboard electronics (sensors, leds, etc) and the conections with the interfaces of the board.

Interface	Onboard electronics
Analog inputs	Analog sensors
Digital inputs	Switches and buttons
Digital outputs	LEDs and circuit release signals
RS-422	Rotary encoders
CAN bus	Frequency converters

Table 3.2: Physical interfaces on the board

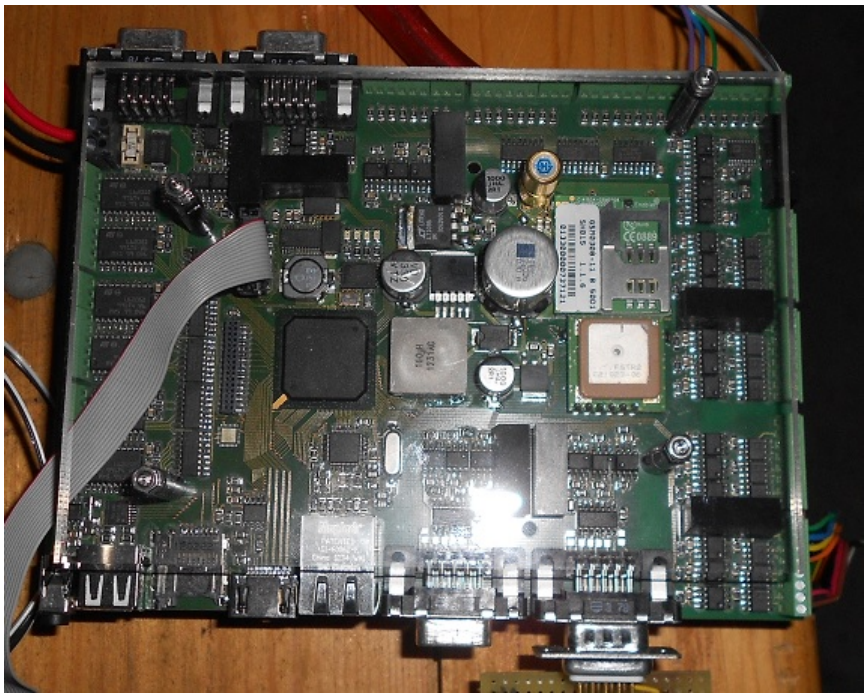

Figure 3.2: ITIV's Spartan 6 custom board

3.3 Hardware description languages

There are two main hardware description languages, which are used to describe hardware circuits in a similar form of a programming language by using conditional statements, assign statements, loops and all the stuff present in the programming languages but it differs in the ways of describing the propagation of time and signal dependencies (sensitivity).

3.3.1 VHDL

VHDL (VHSIC Hardware Description Language) is a hardware description language used in electronic design automation to describe digital and mixed-signal systems such as field-

programmable gate arrays and integrated circuits. VHDL can also be used as a general purpose parallel programming language.

The IEEE Standard 1076 defines the VHSIC Hardware Description Language or VHDL. It was originally developed under contract F33615-83-C-1003 from the United States Air Force awarded in 1983 to a team with Intermetrics, Inc. as language experts and prime contractor, with Texas Instruments as chip design experts and IBM as computer system design experts. The language has undergone numerous revisions and has a variety of sub-standards associated with it that augment or extend it in important ways

3.3.2 Verilog

Verilog, standardized as IEEE 1364, is a hardware description language (HDL) used to model electronic systems. It is most commonly used in the design and verification of digital circuits at the register-transfer level of abstraction. It is also used in the verification of analog circuits and mixed-signal circuits.

The designers of Verilog wanted a language with syntax similar to the C programming language, which was already widely used in engineering software development. Like C, Verilog is case-sensitive and has a basic preprocessor (though less sophisticated than that of ANSI C/C++). Its control flow keywords (if/else, for, while, case, etc.) are equivalent, and its operator precedence is compatible.

3.4 Programming languages

3.4.1 C programming language

C is a general-purpose programming language initially developed by Dennis Ritchie between 1969 and 1973 at AT & T Bell Labs. Like most imperative languages in the ALGOL tradition, C has facilities for structured programming and allows lexical variable scope and recursion, while a static type system prevents many unintended operations. Its design provides constructs that map efficiently to typical machine instructions, and therefore it has found lasting use in applications that had formerly been coded in assembly language, most notably system software like the Unix computer operating system [8].

C is one of the most widely used programming languages of all time, and C compilers are available for the majority of available computer architectures and operating systems [9].

3.5 CAN-bus

CAN bus (for controller area network) is a vehicle bus standard designed to allow microcontrollers and devices to communicate with each other within a vehicle without a host computer. CAN bus is a message-based protocol, designed specifically for automotive applications but now also used in other areas such as aerospace, maritime, industrial automation and medical

equipment.

Development of the CAN bus started originally in 1983 at Robert Bosch GmbH [10]. The protocol was officially released in 1986 at the Society of Automotive Engineers (SAE) congress in Detroit, Michigan. The first CAN controller chips, produced by Intel and Philips, came on the market in 1987. Bosch published the CAN 2.0 specification in 1991. In 2012 Bosch has specified the improved CAN data link layer protocol, called CAN FD, which will extend the ISO 11898-1.

CAN bus is one of five protocols used in the on-board diagnostics (OBD)-II vehicle diagnostics standard. The OBD-II standard has been mandatory for all cars and light trucks sold in the United States since 1996, and the EOBD standard has been mandatory for all petrol vehicles sold in the European Union since 2001 and all diesel vehicles since 2004 [11].

3.5.1 Features

- prioritization of messages
- guarantee of latency times
- configuration flexibility
- multicast reception with time synchronization
- system wide data consistency
- multimaster
- error detection and signalling
- automatic retransmission of corrupted messages as soon as the bus is idle again
- distinction between temporary errors and permanent failures of nodes and autonomous switching off of defect nodes

Chapter 4

Design of the system

In this chapter the chosen design of the final embedded system which is described focusing on the requirements, limitations and advantages. A widely description of each hardware component is done in the Chapter 5, *Hardware development* and for software components in the Chapter 6, *Software development*.

4.1 Hardware

Due to the FPGA based design of the final system, the main aproach to solve this task is to choose a main soft core, which acts as the main processor of the system and build around him an embedded system. There are a lot of possibilities in the field of soft cores. A soft microprocessor (also called softcore microprocessor or a soft processor) is a microprocessor core that can be wholly implemented using logic synthesis. It can be implemented via different semiconductor devices containing programmable logic (e.g., ASIC, FPGA, CPLD), including both high-end and commodity variations [13].
There are a lot of soft cores in the market, an the main options are showed in Table 4.1.

Processor	Developer	Open Source?	Bus Support	Word width
OpenSPARC T1	Sun (Oracle)	Yes	-	64-bit
MicroBlaze	Xilinx	No	PLB, OPB, FSL, LMB, AXI4	32-bit
Nios, Nios II	Altera	No	Avalon	32-bit
LEON3/4	Aeroflex Gaisler	Yes	AMBA2	32-bit
OpenRISC	OpenCores	Yes	Wishbone	32-bit

Table 4.1: Soft processors

The selected soft core for the embedded system was the "'Microblaze". The reasons for using this soft processor are the following:

- **FPGA Spartan 6:** due to use of a Spartan 6 FPGA from Xilinx in the design of the custom board, it makes sense to use the soft processor of the same manufacter in order to exploit all the given advantages

- **Block RAM:** This type of RAM memory is inside the Xilinx FPGA chip and is easily configurable in the development tools
- **Microblaze configurations:** The Microblaze is a fully and easily configurable processor aiming area-optimized, performance and maximum frequency designs
- **Wide bus support:** The Microblaze is compatible with different bus standards, which makes possible to include a huge amount of custom-made peripherals by the comunity in the system, even if there is not interface with this buses because it is possible to easily find a compatible bridge to join both interfaces
- **Xilinx development tools:** The xilinx Microblaze development tools offer a capable integrated development environment with both independent hardware and software development tools, each of one with their functionalities such as HDL simulator or timing reports in the case of the hardware tool and a debugger or block RAM memory initialization in the case of the software tool.
- **Experience with the design tools:** Experience with the design tools is another plus because the learning phase of the tools to get used to them can be avoided and instead of that it is possible to go faster and easier into development phase.

In the next section a detailed vision of the "Microblaze" is given.

4.1.1 Xilinx Microblaze

MicroBlaze is the industry-leader in FPGA-based soft processors, with advanced architecture options like AXI or PLB interface, Memory Management Unit (MMU), instruction and data-side cache, configurable pipeline depth, Floating-Point unit (FPU), and much more. MicroBlaze is a 32-bit RISC Harvard architecture soft processor core that is included with both Vivado Design Edition and IDS Embedded Edition. Highly flexible architecture, plus a rich instruction set optimized for embedded applications, delivers the exact processing system you need at the lowest system cost possible [14].

Microblaze features

- Low Latency Interrupt Mode
- LMB BRAM memory with parity protection on internal BRAMs and caches
- IEEE 754 compatible Floating Point Unit (FPU)
- Instruction and Data Caches configurable: 2kB - 64kB (Block RAM based)
- Branch Optimizations and prediction logic
- Data bus error and instruction bus error
- Divide and floating point exceptions

- Debug Logic

- JTAG control via a debug support core with up to 8 hardware break points

In figure 4.1 it can be seen the main structure of the Microblaze Core:

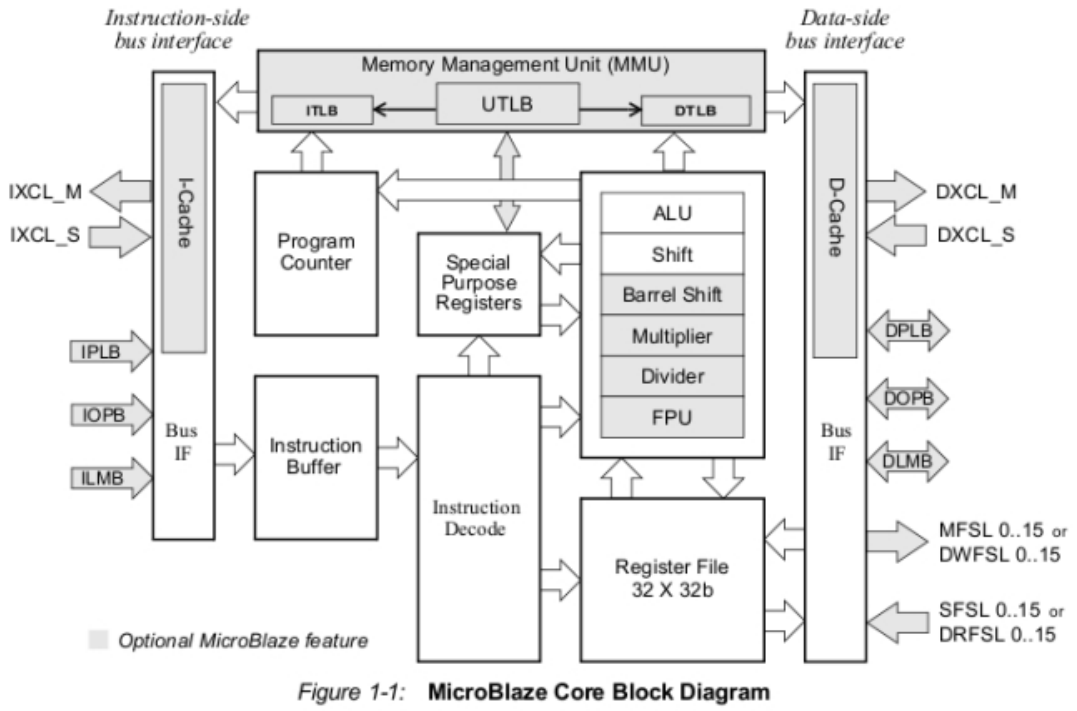

Figure 1-1: **MicroBlaze Core Block Diagram**

Figure 4.1: Microblaze Core block diagram

Hardvard Architecture

A Microblaze processor system, unlike most desktop computers, is based on the Modified Harvard architecture (very common in DSPs and Microcontrollers), which has splitted memories for instructions and for data. Indeed, in a Microblaze processor system the data and instruction memory are physically the same because the block RAM memory is a dual port memory and Microblaze access instructions and data from different buses. Both buses are 32-bit wide, that means up to 4 GBytes of data and instructions are addressable. This memory is accessed by the Microblaze core via a block RAM memory controller connected to his Local Memory Bus. When loading the program into memory, the loader is responsible for separating the user data from the instructions.

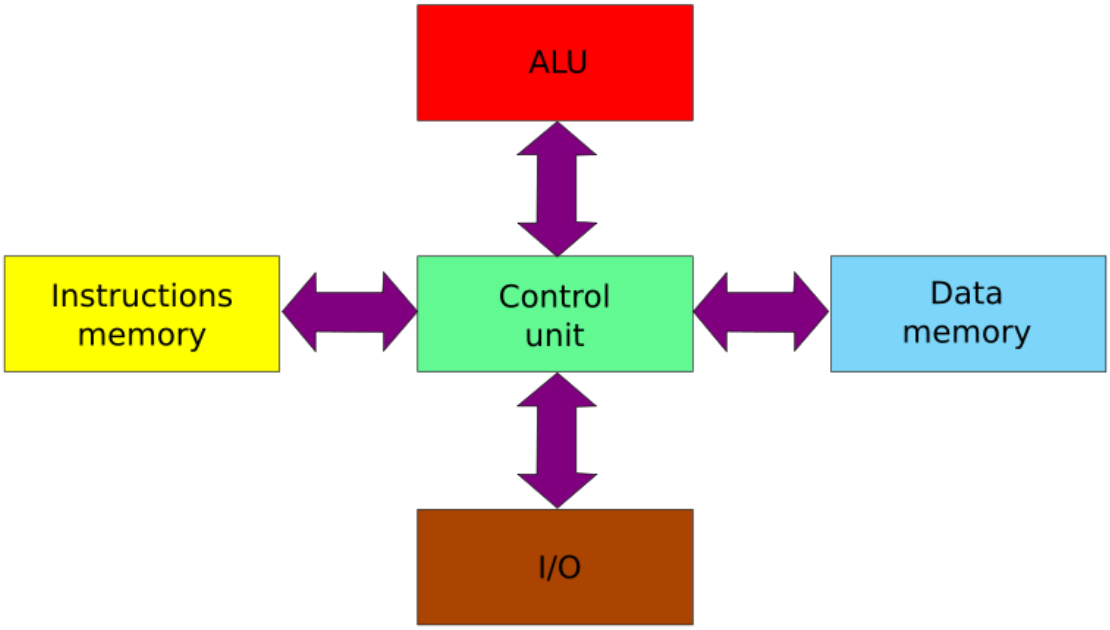

Figure 4.2: Harvard Architecture

4.2 Sotfware

The software options were evident given the selection of the hardware platform, which means that the software options were limited to those Microblaze-compatible. Xilinx offers the following software platforms for his Microblaze core integrated in his *Software Development Kit* tool:

- Xilinx Standalone
- Xilinx Xilkernel
- Linux

Between these three options, the Linux one was automaticaly discarded due to the higher memory requirements and the excesive and useless functionality for the system to be developed compared to the other two software platforms.

4.2.1 Xilinx Standalone

Standalone is the lowest layer of software modules used to access processor specific functions. Standalone is used when an application accesses board/processor features directly and is below the operating system layer [15].

Xilinx Standalone has the following features:

- Interrupt and Exceptions handling
- Data and Instruction Cache handling
- Fast Simplex Link interface macros
- File handling

4.2.2 Xilinx Xilkernel

Xilkernel is a small, robust, and modular kernel. It is highly integrated with the Platform Studio framework and is a free software library that you receive with the Xilinx Embedded Development Kit (EDK) [15]. Xilkernel:

- Allows a very high degree of customization, letting you tailor the kernel to an optimal level both in terms of size and functionality.
- Supports the core features required in a lightweight embedded kernel, with a POSIX API.
- Works on MicroBlaze, PowerPC 405, and PowerPC 440 processors.

Xilkernel includes the following key features [15]:

- High scalability into a given system through the inclusion or exclusion of functionality as required.
- Complete kernel configuration and deployment within minutes from inside of Platform Studio.
- Robustness of the kernel: system calls protected by parameter validity checks and proper return of POSIX error codes.
- A POSIX API targeting embedded kernels, win core kernel features such as:
  - Threads with round-robin or strict priority scheduling.
  - Synchronization services: semaphores and mutex locks.
  - IPC services: message queues and shared memory.
  - Dynamic buffer pool memory allocation.
  - Software timers.
  - User-level interrupt handling.
- Static thread creation that startup with the kernel.
- System call interface to the kernel.
- Exception handling for the MicroBlaze processor.
- Memory protection using MicroBlaze Memory Management (Protection) Unit (MMU) features when available.

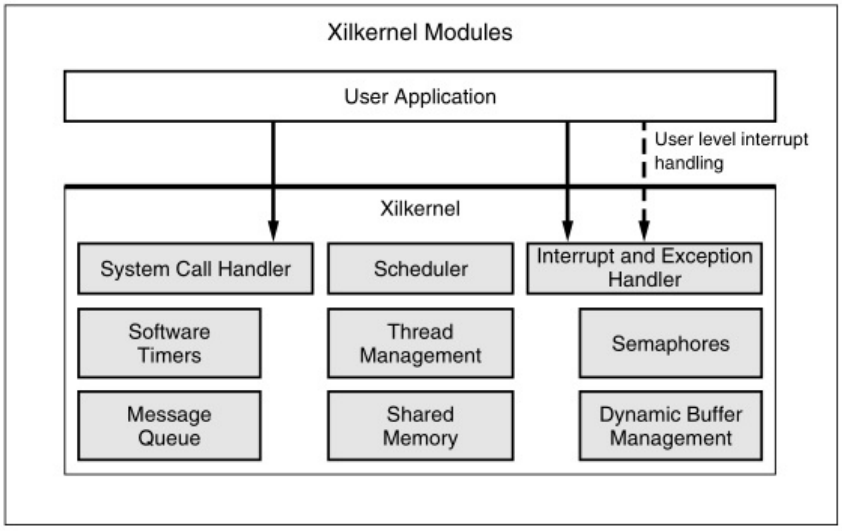

Figure 4.3: Xilkernel Modules

4.3 Restrictions

4.3.1 Memory

The amount of available memory is quite small because it depends on the FPGA where the system is deployed. In this case, the Spartan 6 lx150 provides 4,824 Kb of block RAM. All the functionality is fitted into this amount of memory but there is not enough memory space to hold all the telemetry data.

Performance

The performance of the embedded system is a critical parameter that depends on several factors:

- Clock frecuency: in the board, frequencies from 50 MHz to 120 MHz are achievable.
- Soft core configuration: using hardware mulpliers and caches (instruction and data).
- Speed of the memory: the access speed could vary if a external memory is used or the internal built-in block RAM

4.4 Selected Software platform

Xilinx Standalone is used as a software test platform in this Master Thesis in the early stages of the development process (due to his simplicity) to programm and test the firmware and low level functions of the developed peripherals written in VHDL.

Xilinx Xilkernel is chosen as the basis software development platform of the final system (control routine).

4.5 Advantages

Despite using FPGAs could be slower than using built-in microcontrollers, this aproach has also a lot of advantages, which are enumerated and explained in the following list:

- Taking advantage of the parallel processing capability of the FPGAs. Some algorithms and software routines could take a long CPU time to be completed and the parallelization is a good way to improve the performance. In a single processor system is not possible to parallelize in the software but if this system is implemented in a FPGA and there is enough place, this routine or algorithm can be implemented in hardware as a coprocessor or custom hardware module by using hardware description languages such as VHDL or Verilog. This solution has the advantage of saving CPU time because it is only needed to send the data to the coprocessor and retreive the data back
- No limits in the type of system or configurations to implement. The designer has the freedom of choosing the different components, use components made by the comunity and if needed replace them for new improved versions thereof in a search of better performance
- Easy expansion of the system. This is very important in a "open" system like the system of this Master Thesis is. New hardware functionalities, that were not planed at the beginning, can be added to the system if there is enough space remaining inside the FPGA

4.6 Outlook

Figure 4.4 shows how the new FPGA based design is integrated alongside with the sensors and further systems of the kart.

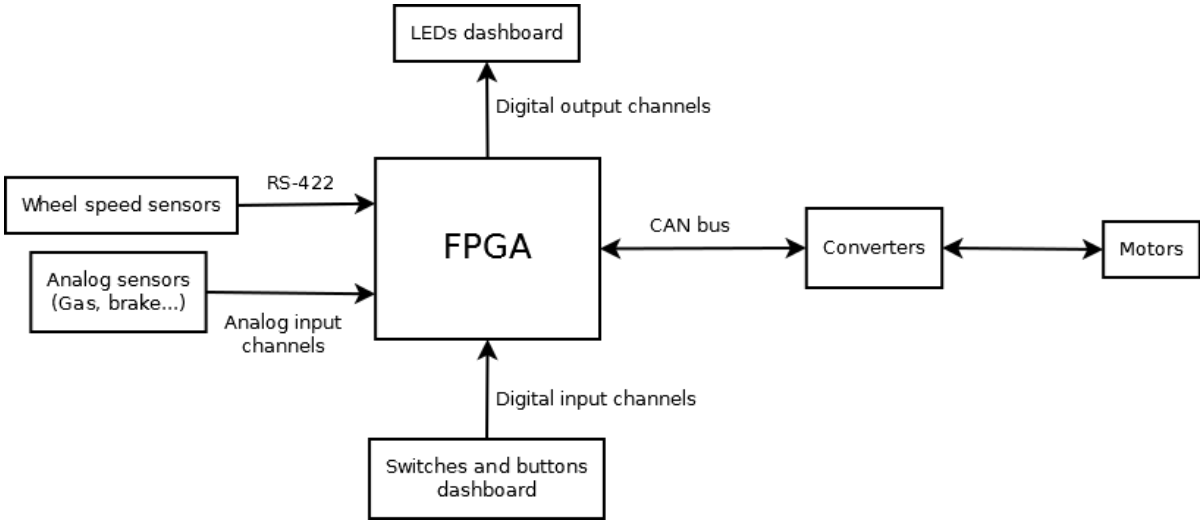

Figure 4.4: System outlook

Chapter 5

Hardware development

In this chapter a detailled description of the hardware development is given. In this work the hardware development is limited to implement the hardware interfaces to control the physical electronic circuits in the board and the architecture of the embedded system. Each component of the system is described in detail including the register interfaces, involved modules and performance and area results. In order to implement the hardware, the Xilinx development tools are used.

5.1 Microblaze

As explained in the previous design chapter, the Microblaze core has different configuration options. In this Master Thesis, the balance between maximum performance and good area optimization of the FPGA is desirable. The bottleneck of this system is the access throgh the PLB bus to the system pereipherals. The PLB bus is clocked at the same frecuency of the Microblaze, which means that the higher the frecuency of the Microblaze and the PLB bus, the transactions between PLB bus and Microblaze are faster. Indeed the Local Memory Bus, which is connected to the system's main memory, works at the same frequency also, so higher frequencies mean higher transactions between memory and processor. Thus the configuration aims higher frecuency and area parameters. In the next section, it is described the Microblaze configuration and how it is obtained:

5.1.1 Configuration

- No instructions and data caches
- No Floating Point Unit
- No integer hardware multiplier and divider
- Branch target cache optimization by using one additional 18 Kb block RAM memory for this purpose

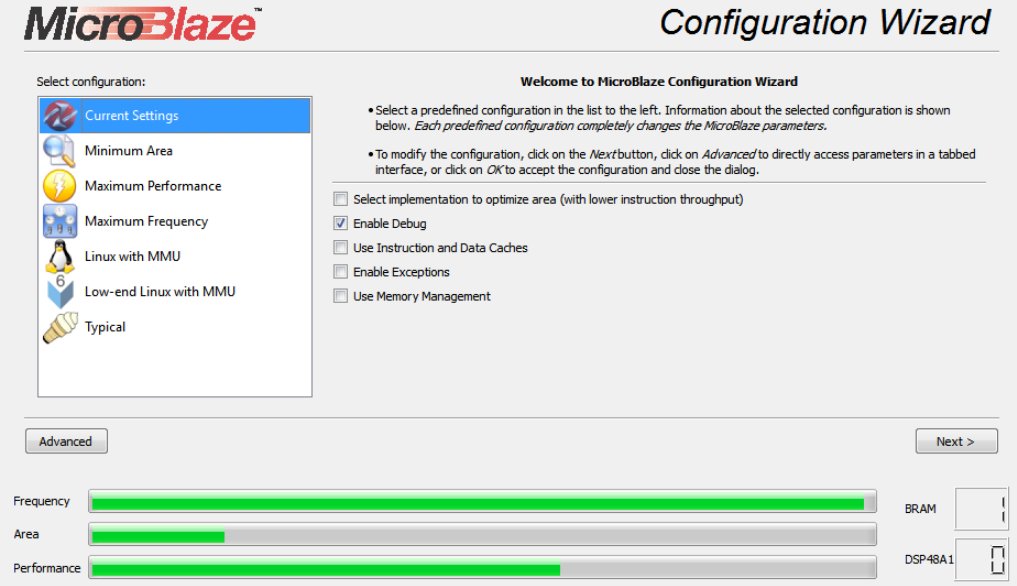

Figure 5.1: Microblaze configuration in EDK

5.2 Block RAM

The Xilinx block RAM is used as the system's main memory and it stores programm instruction as well as user data. To use block RAM in the Microblaze system is needed a compatible memory controller and a Local Memory Bus, which are provided by Xilinx in the *Embedded Development Kit* (EDK).

5.2.1 Interface

Local Memory Bus

The lmb_v10 module is used as the LMB interconnect for Xilinx FPGA based embedded processor systems. The LMB is a fast, local bus for connecting MicroBlaze instruction and data ports to high-speed peripherals, primarily on-chip block RAM (BRAM). The Local Memory Bus has the following features [16]:

- Efficient, single master bus (requires no arbiter)
- Separate read and write data buses
- Low FPGA resource utilization
- up to 125 MHz operation

LMB memory controllers

The LMB BRAM Interface Controller connects a block RAM memory to an lmb_v10 bus [17]. The controller supports:

- LMB v1.0 bus interfaces with byte enable support
- Used in conjunction with bram_block peripheral to provide fast BRAM memory solution for MicroBlaze ILMB and DLMB ports.
- Supports byte, half-word, and word transfers
- Supports optional BRAM error correction and detection.

In a MicroBlaze system, without ECC protection, the LMB BRAM Interface controller would typically be connected according to figure 5.2.

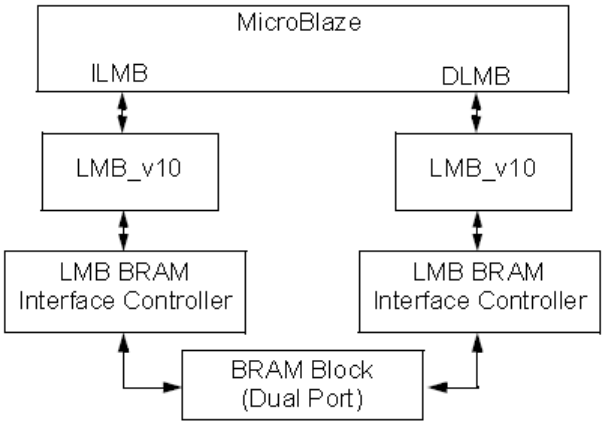

Figure 5.2: Microblaze System memory bus

As can be observed in figure 5.2, it is needed two memory controllers because of the dual port block RAM. One is used for programm instructions and other for user data meaning higher speeds in the memory accesses.

5.2.2 Configuration

The only configuration parameter of the block RAM is the size and it is adjusted with the address range of the memory controller. The Xilinx *Embedded Development Kit* tool joins automaticaly blocks of 18 Kbits block RAM to build bigger RAM memory structures but due to a limitation in the controller, the maximum addressable memory range is 64 KBytes. It does not mean that the maximum amount of memory is 64 KBytes, indeed it is possible to use more than one memory controller for both instruction and data ports. Each controller must have consecutive memory address ranges in order to build a bigger memory address space. Apart from that, to use the full capacity of the memory there are some minor changes in the software part, specifically in the linker script and the *bmm* file (memory initialization file). These changes are explained in the software chapter.

The developed system has four memory controllers with four 64 KBytes block RAM memories for a total of 256 KBytes of RAM to host the operative system, software routines and peripheral firmwares.

5.3 PLB Bus

The embedded system needs a main data bus to communicate the Microblaze processor with all of his peripheral components. There are two main options provided by Xilinx, using AXI bus or PLB bus. The first belongs to the AMBA specification introduced by ARM and the second is included in the IBM core connect specification. The AXI bus has a higher perfomance compared to the PLB but on the other hand it requires more space in the FPGA because of his shared bus architecture. The PLB bus was chosen due to his low space requirements and reasonable performance for the target system.

The Xilinx PLB consists of a central bus arbiter, the necessary bus control and gating logic, and all necessary bus OR/MUX structures. The Xilinx PLB provides the entire PLB bus structure and allows for direct connection with a configuration number of masters and slaves. It has the following main features:

- Arbitration support for a configurable number of PLB master devices
- 128-bit, 64-bit, and 32-bit support for masters and slaves
- PLB address pipelining
- PLB watchdog timer
- Supports a configurable number of slave devices
- No external OR gates required for PLB slave input signals
- PLB Reset circuit

Figure 5.3 shows the structure of the PLB connections for a system with masters and slaves peripherals.

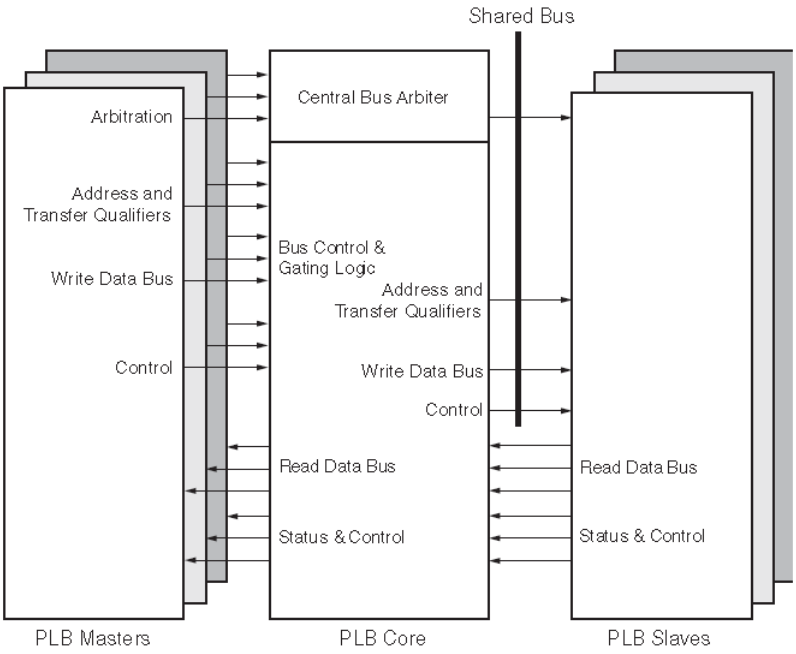

Figure 5.3: Xilinx PLB bus

For additional information, please see the Xilinx documentation in [23].

5.4 Wishbone Bus

Despite the PLB is the main bus of the system and all of the developed peripherals have a register interface, which can be easily adapted to the PLB bus via a wrapper, the CAN transceiver peripheral has a Wishbone bus interface, which is adapted to the PLB bus via a bridge explained in section 5.5.

The WISHBONE System-on-Chip (SoC) Interconnect Architecture for Portable IP Cores is a portable interface for use with semiconductor IP cores. Its purpose is to foster design reuse by alleviating system-on-a-chip integration problems. This is accomplished by creating a common, logical interface between IP cores. This improves the portability and reliability of the system, and results in faster time-to-market for the end user. WISHBONE itself is not an IP core but it is a specification for creating IP cores. It has the following features:

- Simple, compact, logical IP core hardware interfaces that require very few logic gates.
- Modular data bus widths and operand sizes.
- Supports both BIG ENDIAN and LITTLE ENDIAN data ordering.
- Variable core interconnection methods support point-to-point, shared bus, crossbar switch, and switched fabric interconnections.
- Supports single clock data transfers.

- MASTER/SLAVE architecture for very flexible system designs.

Figure 5.4 shows the structure of the Wishbone connections for a system with masters and slaves peripherals.

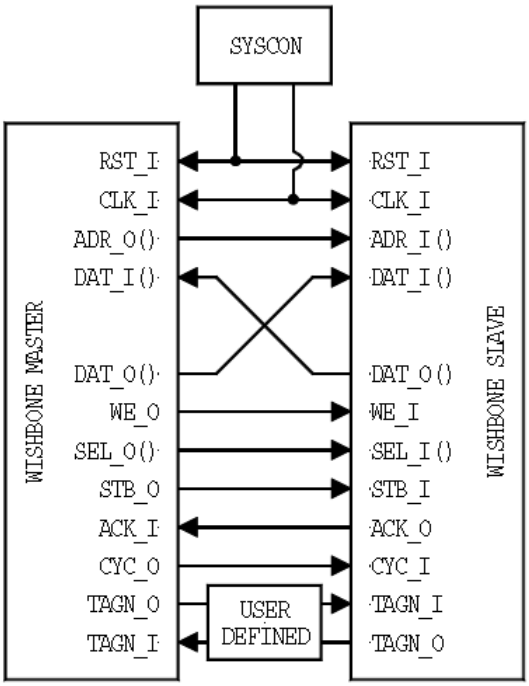

Figure 5.4: Wishbone bus

To get a more detailed vision, please see the OpenCores documentation in [24].

5.5 PLB to WB Bridge

As explained in the previous section, a bridge is needed to connect the interface between the CAN transceiver peripheral and the PLB bus. For this purpose, a free core hosted in the OpenCores community [31] was used.
The core is a simple CoreConnect PLBv46 to Wishbone bridge that can allow Wishbone peripherals to be used on Xilinx processor designs. It conforms to the sub-set of the PLBv46 specification adopted by Xilinx in the EDK and has the following features:

- PLBv46 Slave Attachment (non-bursting)
  - Native 32-bit slave interface to PLBv46 bus.
  - 32-bit master interface to Wishbone bus.
- Directly integrated into EDK tools as a custom pcore (synthesizable VHDL).
- Microprocessor Peripheral Definition (MPD) file provided.

- Handling of Bus Errors

Figure 5.5 shows the structure of the system with the PLB to Wishbone bridge connected.

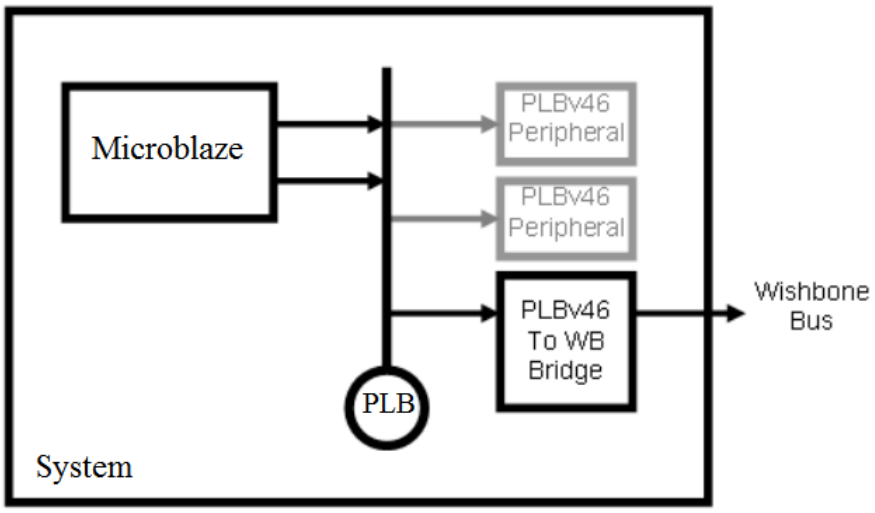

Figure 5.5: PLB to Wishbone bridge

For more information about the PLB to Wishbone bridge core, see the documentation hosted in [25].

This core was imported in the Xilinx EDK as a custom perihperal, added to the embedded system and connected to the CAN transceiver peripheral (see section 5.7). No further modifications to the core were needed.

5.6 PLB Wrapper

The peripherals in this Master Thesis have been developed thinking to make them usable and portable to any FPGA-based system. In order to achieve this commitment, all the peripherals have a standard register interface. The use of the PLB bus as the basis bus for the system forces to adapt the standard register interface to the PLB bus interface. This can be achieved by using a wrapper, which maps the registers with an memory address accesible for the processor. The *Xilinx Embedded Delopment Kit* provides a wizard to generate vhdl or verilog wrapper templates for custom peripheral as well as software template to write the firmware. The process to make the wrapper is as follows:

1. Generation of the wrapper template using the utility of Xilinx development environment to create a new custom peripheral compliant with the PLB bus.

2. Modify the generated wrapper template to connect the standard register interface of the peripheral with the bus slave handler.

5.6.1 Wrapper template generation

To generate the template the wizard for creating a new peripheral must be executed. When the type of interface is requested select Processor Local Bus and user registers option to create a peripheral with user readable registers and PLB bus interface. The number of register must be specified depending on the real number of register that the custom peripheral has.

5.6.2 Modifications to the template

Once the template is generated, it is needed to modify certain files, that appear in the directory of the custom peripheral.
Figure 5.6 shows the structure of the directory created by the wizard for the custom peripheral.

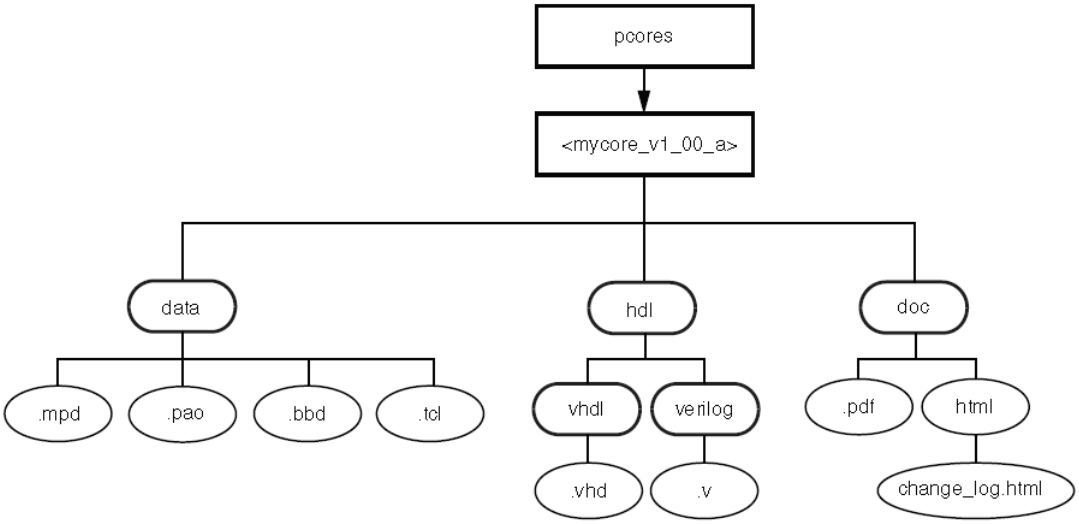

Figure 5.6: EDK custom peripheral directory structure

The following list describes the type of files to be modified and their function [26]:

- File with “pao” extension: This file establish the VHDL files hierarchy of the peripheral. It must be edited to take into account the new VHDL files of the custom peripheral.
- File with “vhd” extension with the name of the peripheral given in the wizard in the hdl directory: This file is top level of the wrapper. Here the ports of the custom peripheral must be mapped. Also the external ports must be defined in this file.
- File with “vhd” extension with the name “user_logic” in the hdl directory: This file is where the logic of the peripheral is implemented. The mapping between the PLB bus registers and the peripheral register interface must be done here.

Netlist files

There is a special case when the custom peripheral has not only vhdl files but also netlist files, with “ngc” extension. In this case, the following modifications must be done in the directory and file structure to acomplish the implementation:

- A new folder with the name “netlist” must be created at the same level of the “data” and “hdl” folders (see Figure 5.6). Inside this folder, the “ngc” netlist files must be placed.

- The file with “bbd” extension inside the “data” folder must be modified. A list of the netlist files included in the custom peripheral (elements of the list separated by commas) must be typed in the file.

5.6.3 Peripherals to have the PLB wrapper

The following list shows the peripherals of the system, which need a PLB wrapper.

- Traction Control System (see section 5.9)

- Torque Vectoring (see section 5.8)

- Inputs and Outputs (see section 5.10)

5.7 CAN transceiver

A CAN transceiver is included in the design to fulfill the communication requirements of the new converters for the motors, which has a CAN interface. There are a lot of alternatives in the market for CAN transceiver IP cores, some with royalties but other are completely free. For this system, a free CAN transceiver IP core was selected.

5.7.1 OpenCores CAN transceiver core

This CAN controller is based on the Philips SJA1000 and has a compatible register map with a few exceptions. It also supports both BasicCAN (PCA82C200 like) and PeliCAN mode. In PeliCAN mode the extended features of CAN 2.0B is supported. The mode of operation is chosen through the Clock Divider register. The core is written in Verilog.

Register interface

All registers are one byte wide and the addresses are also byte addresses. Byte reads and writes must be used when interfacing with the core. The core is little endian.

5.7.2 Wrapper

Figure 5.7 shows the structure of the CAN transceiver peripheral.

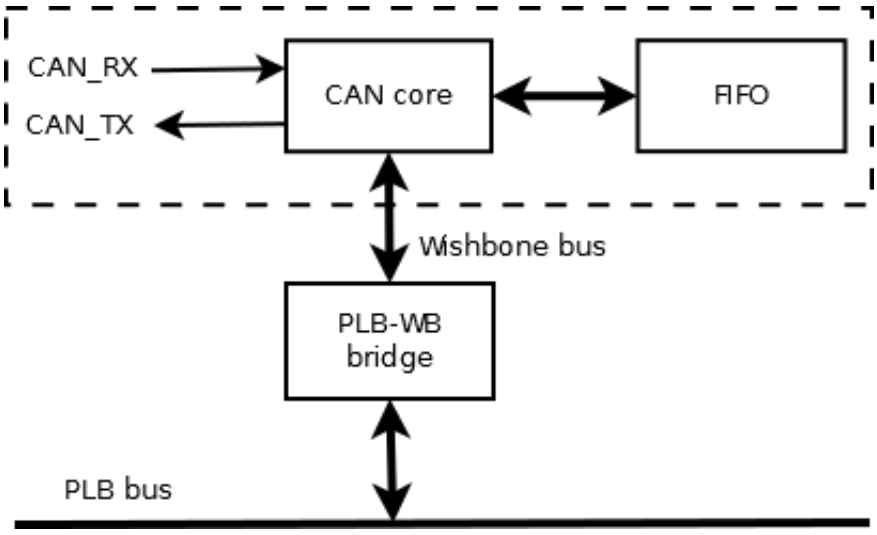

Figure 5.7: CAN transceiver peripheral

5.8 Torque vectoring

Until now, the torque vectoring functionality in the kart (see section 2.3.1) is implemented in the software routine that controlls the kart. Because of its considerable computational cost relative to other tasks carried out by the software and so as to release CPU time, this functionality is implemented in a hardware core as coprocessor.

In this section, the development process of the peripheral for the Torque Vectoring calculations is explained. The core is written in VHDL and uses Xilinx primitives to take advantage of the Xilinx features of the Spartan 6 family.

5.8.1 Analysis

Retrieving and analysing the Torque Vectoring formula 2.1, it can be concluded the number and type of operations and thus the hardware components needed.

$$\Delta MTorqueVectoring = v^2 \cdot \sin^2 \delta_l \cdot d \cdot 1.85736$$

From the formula 2.1 can be deducted that all operations need specialized floating point hardware. It consists in one sine operation, two floating point power operations and three floating point multiplications. The power operations can be considered from the viewpoint of hardware as two multiplications, so the final number of multiplications is five.
To perform floating point operations in the FPGA, it is avaible a configurable LogiCORE IP Floating-Point core provided by Xilinx and generable by the Core generator tool. It is also needed a core to perform the sinus operation and for this task Xilinx provides the LogiCORE IP CORDIC. In the next section the features of these cores and their configuration for the purposes of this Master Thesis are explained.

LogiCORE IP Floating-Point Operator v5.0

The Xilinx Floating-Point core provides designers with the means to perform floating-point arithmetic on an FPGA device. The core can be customized for operation, word length, latency, and interface [27]. The core has the following features:

- Compliance with IEEE-754 Standard
- Use of XtremeDSP slice for multiply
- Optimizations for speed and latency
- Fully synchronous design using a single clock
- Supported operators:
  - multiply
  - add/subtract
  - divide
  - square-root
  - comparison
  - conversion from floating-point to fixed-point
  - conversion from fixed-point to floating-point
  - conversion between floating-point types

LogiCORE IP CORDIC

The CORDIC core implements a generalized coordinate rotational digital computer (CORDIC) algorithm, initially to iteratively solve trigonometric equations, and later generalized to solve a broader range of equations, including the hyperbolic and square root equations. The CORDIC core implements the following equation types:

- Rectangular $\longleftrightarrow$ Polar Conversion
- Trigonometric
- Hyperbolic
- Square Root

Two architectural configurations are available for the CORDIC core:

- A fully parallel configuration with single-cycle data throughput at the expense of silicon area
- A word serial implementation with multiple-cycle throughput but occupying a small silicon area

The CORDIC core works with fixed point arithmetics, so conversions between fixed point data and IEEE-754 Standard floating point data are needed. If using Trigonometric functions, the input must be in radians [28].

Hardware requirements

The hardware to be used in the implementation of the Torque Vectoring core is resumed in Table 5.1.

IP core	Configuration	Operations
Floating-Point Operator	Multiplier	Multiply and power
Floating-Point Operator	Float to Fixed	Adapt floating point data to CORDIC's fixed point format
Floating-Point Operator	Fixed to Float	Adapt CORDIC's fixed point results to floating point format
CORDIC	Sine	Sine operation

Table 5.1: Torque Vectoring floating point hardware

5.8.2 Design

With the basic understanding of the Xilinx IP cores, in this design stage the block structure, behaviour flow diagram and signals and register interface are described.

Figure 5.8 shows the flow diagram of the Torque Vectoring module.
Note that Figure 5.8 represents the flow of a complete calculation of the torque vectoring factor. Each box represents an operation, which is performed in one clock cycle.

Figure 5.9 shows the block diagram of the Torque Vectoring module.
The inputs and outputs of the Torque Vectoring module are described in table 5.2.

Signal	Type	Description
Clk	Input	Reference clock signal
Reset	Input	Active low reset signal
Speed(31:0)	Input	Register to store the speed
Steering_input(31:0)	Input	Register to store the steering angle
Torque_factor(31:0)	Input	Register to store the torque factor
Torque_const(31:0)	Input	Register to store the torque constant
Torque(31:0)	Output	Register where the result is stored

Table 5.2: Torque Vectoring module signals

5.8.3 Implementation

In order to implement in VHDL the module, only one process is used. This process controls by using a counter, the actual cycle and performs the corresponding floating point operation. When all the operations have been executed, the result is stored in a register, which is accesible by the user, and the core starts a new calculation with the actual values in the input registers.

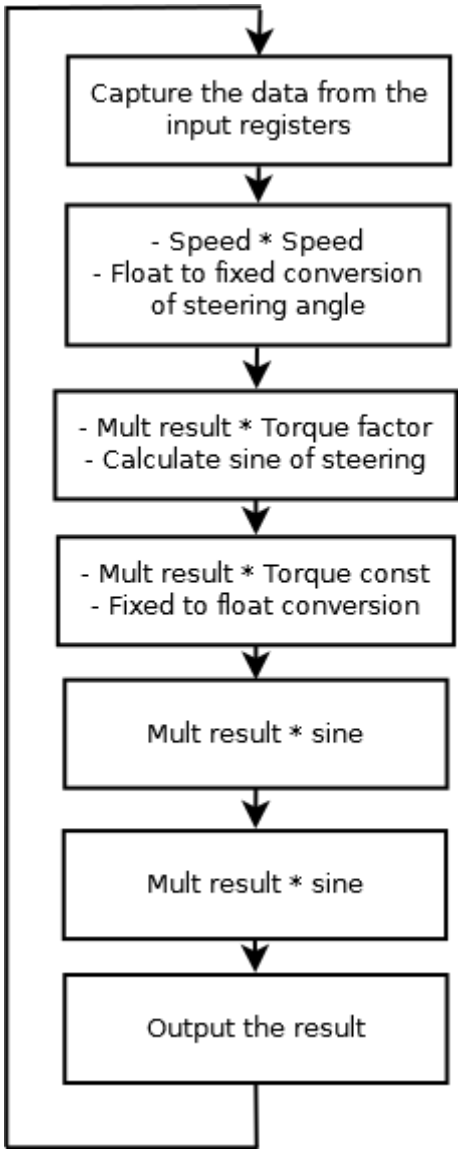

Figure 5.8: Flow Diagram of the Torque Vectoring module

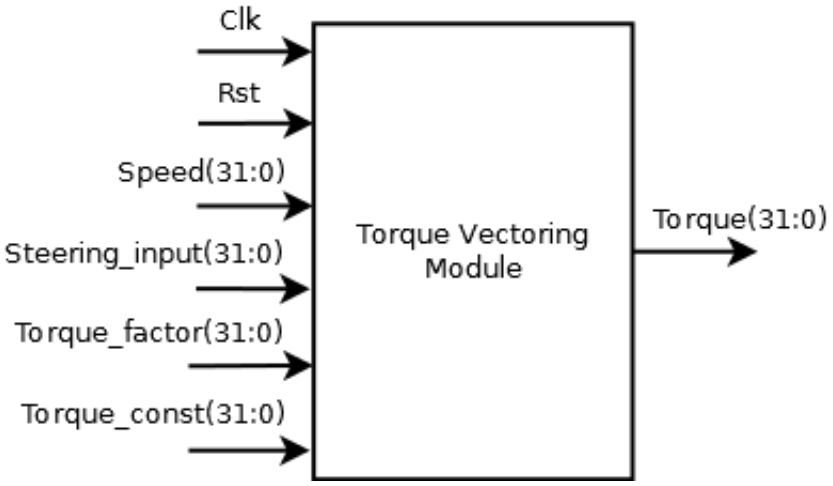

Figure 5.9: Torque Vectoring module core

This peripheral needs a PLB wrapper because it is made to be accesible from the software. Due to his only output register but four input registers, the PLB wrapper template must be created to have five software accesible register, four of them write registers and one read register.

5.8.4 Performance

The performance of the peripheral is described in the next list:

- Maximum frequency: it is limited by the CORDIC core to 6.25 MHz
- Performance: A complete operation cycle takes 7 clock ticks, which means a maximum performance of 1120 ns to calculate the torque vectoring percentage.
- FPGA resources: this module needs 1400 slices of the FPGA to be implemented

5.9 Traction Control System

Like the torque vectoring functionality, the traction control system is implemented in the software routine that controlls the kart. Because of its considerable computational cost relative to other tasks carried out by the software and so as to release CPU time, this functionality is implemented in a hardware core as coprocessor.

In this section, the development process of the peripheral for the Traction Control System calculations is explained. The core is written in VHDL and uses Xilinx primitives to take advantage of the Xilinx features of the Spartan 6 family.

5.9.1 Analysis

To learn more about the hardware requirements Figure 5.10 shows the Simulink model of the Traction Control controller for the Kart [2].

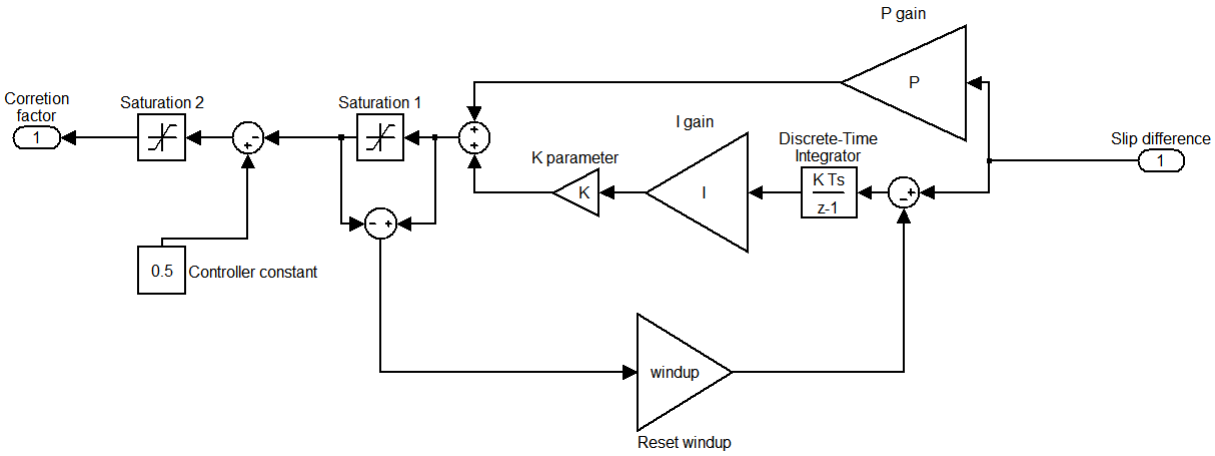

Figure 5.10: Simulink model of the Traction Control System

Figure 5.10 shows that the Traction Control System controller is implemented with a proportional-integral controller with windup reset to not saturate the integrator. There can be also observed two saturation stages to keep the result in the desired range. The input of the controller is the slip difference between the wheels and the output is the correction factor to be applied to the actual torque. All the operations inside the controller are performed in floating point IEEE-754 Standard.

To perform floating point operations in the FPGA, it is avaible a configurable LogiCORE IP Floating-Point core provided by Xilinx and generable by the Core generator tool, which is explained in section 5.8.1.

Hardware requirements

The hardware to be used in the implementation of the Traction Control System core is resumed in Table 5.3

IP core	Configuration	Operations
Floating-Point Operator	Multiplier	Multiply
Floating-Point Operator	Add and sub	Adds and subs
Floating-Point Operator	Comparator less than	Saturation comparation
Floating-Point Operator	Comparator greather than	Saturation comparation

Table 5.3: Traction Control System floating point hardware

5.9.2 Design

With the basic understanding of the Xilinx IP cores, in this design stage the block structure, behaviour flow diagram and signals and register interface are described.

Figure 5.11 shows the flow diagram of the Traction Control System module.

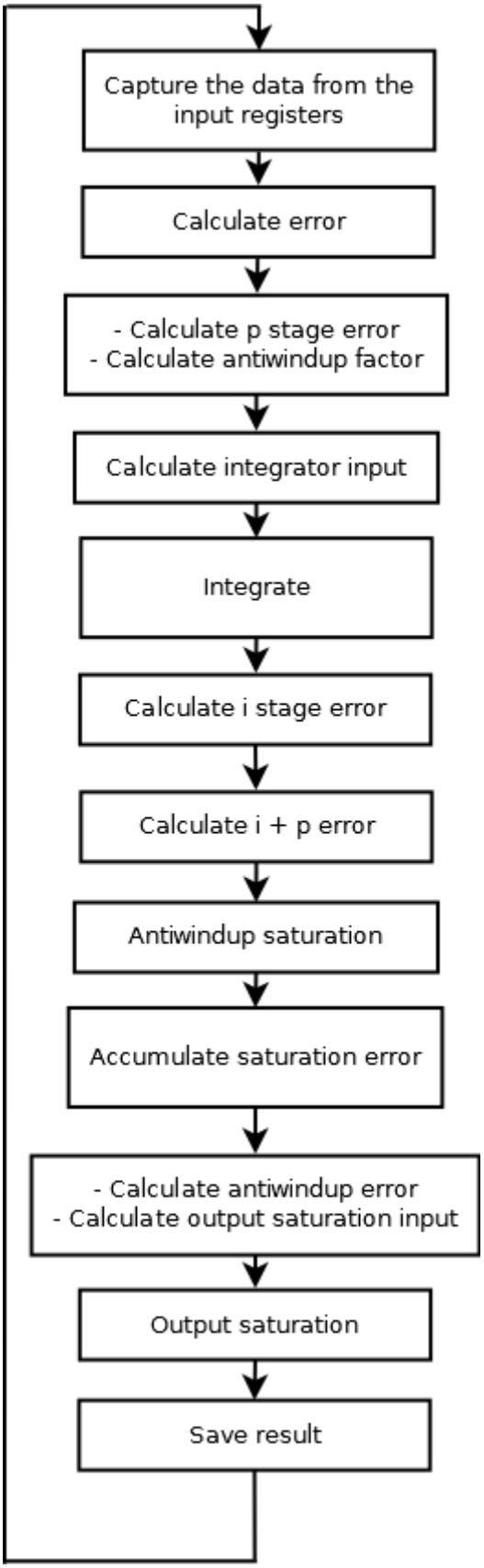

Figure 5.11: Flow Diagram of the Traction Control System module

Note that Figure 5.11 represents the flow of a complete calculation of the Traction Control correction factor. Each box represents an operation, which is performed in one clock cycle.

Figure 5.12 shows the block diagram of the Traction Control System module:.

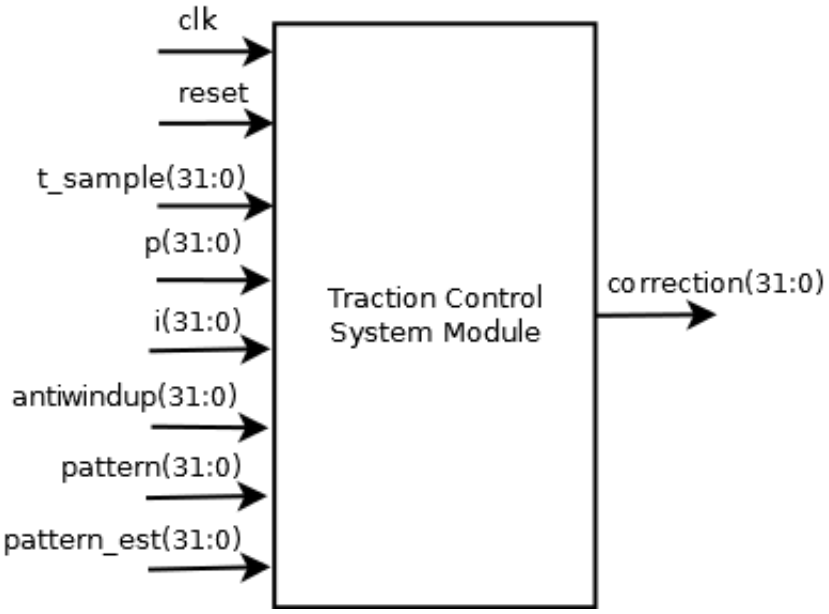

Figure 5.12: Traction Control System module core

The inputs and outputs of the Traction Control System module are described in table 5.4.

Signal	Type	Description
clk	Input	Reference clock signal
reset	Input	Active low reset signal
t_sample(31:0)	Input	Sampling time of the controller
p(31:0)	Input	P gain of the controller
i(31:0)	Input	I gain of the controller
antiwindup(31:0)	Input	Antiwindup gain
pattern(31:0)	Input	Actual slip input
pattern_est(31:0)	Input	Set point
correction(31:0)	Output	Register where the result is stored

Table 5.4: Traction Control System module signals

The core has also a generic called "'Latency". The value of the "Latency" generic determines a delay between the result is calculated and the next sample is captured.

5.9.3 Implementation

In order to implement in VHDL the module, only one process is used. This process controls by using a counter, the actual cycle and performs the corresponding floating point operation.

When all the operations have been executed, the result is stored in a register, which is accesible by the user, and the core starts a new calculation with the actual values in the input registers. The saturation comparations are made parallel in the same clock cycle. The value of the generic called "'Latency" is used to delay the capture of the next sample.

This peripheral needs a PLB wrapper because it is made to be accesible from the software. Due to his only output register, the PLB wrapper template must be created to have seven software accesible register, six of them write registers and one read register.

5.9.4 Performance

The performance of the peripheral is described in the next list:

- Maximum frequency: it is limited by the floating point operators to 50 MHz
- Performance: A complete operation cycle takes 12 clock ticks, which means a maximum performance of 240 ns to calculate the traction control correction factor.
- FPGA resources: this module needs 450 slices of the FPGA to be implemented

5.10 Inputs/Outputs

The kart has a widely amount of switches and buttons to set some parameters, such as torque vectoring, traction control, speed limit, boost and more and analog inputs such as gas pedal or brake pedal. There are also some LEDs, which allow the driver to see if an error ocurrs or the state of the batteries. On the board there are physical electronic components that are responsible for driving the inputs and the outputs but it is needed hardware interfaces to control these components and to give a software accesible register interface. Each case is explained in a dedicated section.

5.10.1 Digital Inputs

In this section, the development process of the peripheral for the control of the digital inputs is explained. The core is written in VHDL.

Analysis

In this development stage the requirements to be acomplished and the resources needed are explained.

In order to describe in VHDL the behaviour of the interface, it is needed a full understanding of how the electronics on the board work. The electronics on the board were selected to use the minimum amount of communication lines between the input module and the FPGA. To acomplish this task, the board has three *74HCT166D* synchronous 8-bit parallel-in/serial-out shift registers, which are serial connected for a total of 24 digital inputs.

Figure 5.13 shows the block diagram of the parallel-in/serial-out shift register.

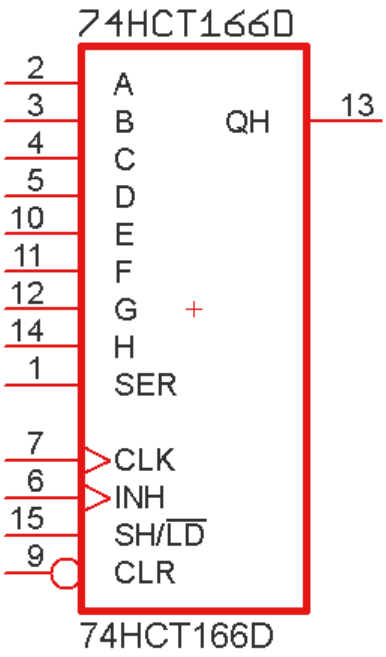

Figure 5.13: 74HCT166D shitf register

As seen in Figure 5.13, there are 8 inputs for parallel load, 1 serial output and control inputs such as clock and shift/load. For the input module, the signals clock, shift/load and the serial output are needed and connected to the FPGA. The 24 inputs values are transmitted to the FPGA using the serial output of the chip. Thus the input module to be developed must generate the clock and shift/load signals to control the chips on the board and at the same time receive the values of the serial output of the chips and deserialize the information. This information is given to the user through a software accesible register interface.

Design

Using all the requirements and the basic understanding of the electronic chips on the board in this design stage the block structure, behaviour flow diagram and signals and register interface are described.

Figure 5.14 shows the flow diagram of the Digital Inputs module.

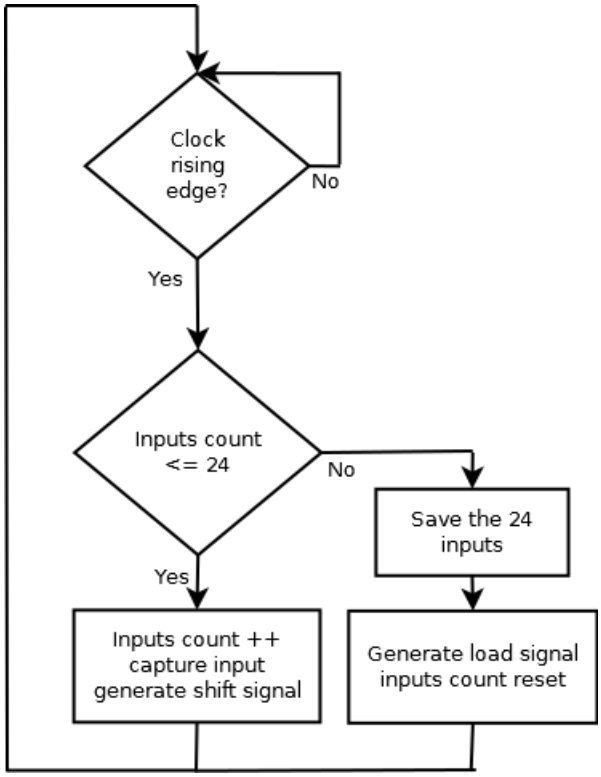

Figure 5.14: Flow Diagram of the Digital Inputs module

Figure 5.15 shows the block diagram of the Digital Inputs module.

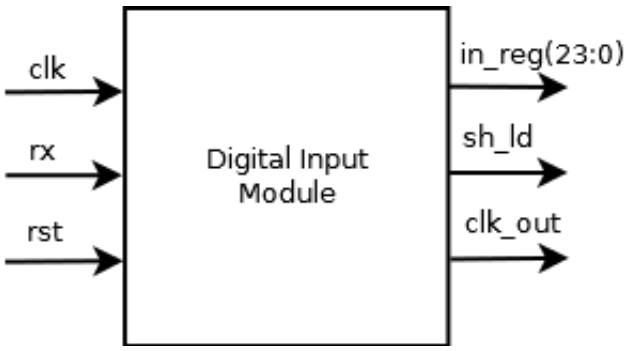

Figure 5.15: Digital Inputs module core

The inputs and outputs of the Digital Inputs module are described in table 5.5.

Signal	Type	Description
clk	Input	Reference clock signal
rst	Input	Active low reset signal
rx	Input	Serial input
clk_out	Output	Clock signal to *74HCT166D*
sh_ld	Output	Shift/Load signal to *74HCT166D*
in_reg(23:0)	Output	Register interface to software

Table 5.5: Digital Inputs module signals

Implementation

In order to implement in VHDL the module, only one process is used. This process controls by using a counter, the number of the input captured at the moment of a clock rising edge. Once all the 24 inputs are received, this process saves the 24 inputs in the output software register and initiates a new capture cycle. Depending on the capture process, the signals clock and shift/load are generated to control the electronic logic.

Other important point is the PLB wrapper of this peripheral. Due to his only output register, the PLB wrapper template must be created to have only one software accesible register.

Performance

The performance of the peripheral is described in the next list:

- Maximum frequency: it is limited by the *74HCT166D* maximum frequency to 50 MHz
- Performance: A complete read cycle takes 25 clock ticks, which means a maximum performance of 500 ns to read the 24 digital inputs.
- FPGA resources: this module needs 20 slices of the PFGA to be implemented

5.10.2 Digital Outputs

In this section, the development process of the peripheral for the control of the digital outputs is explained. The core is written in VHDL.

Analysis

In order to describe in VHDL the behaviour of this interface, it is needed a full understanding of how the electronics on the board work. The electronics on the board were selected to use the minimum amount of communication lines between the input module and the FPGA. To acomplish this task, the board has five *74HC595* 8-bit serial-in, parallel-out shift register with output latches, which are serial connected for a total of 24 digital outputs and otuputs enable.

Figure 5.16 shows the block diagram of the 8-bit serial-in, parallel-out shift register with output latches.

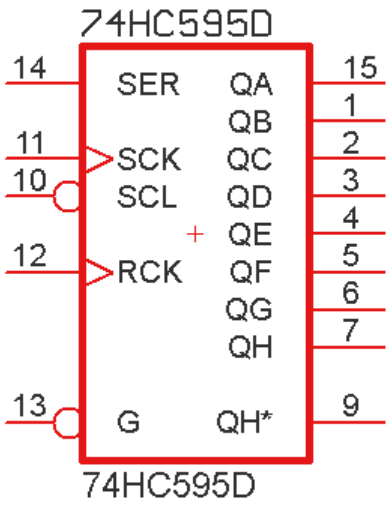

Figure 5.16: 74HC595D shitf register

As seen in Figure 5.16, there are 8 outputs, for parallel communication, 1 serial input and control inputs such as shift register clock input and storage register clock input. For the output module, the two clock signals and the serial input are needed and connected to the FPGA. The 24 outputs values plus the enable signal are transmitted to the electronic chip from the FPGA using the serial input of the chip. Thus the output module to be developed must generate the clock signals to control the chips on the board and at the same time send the serialized values to the chips trough the serial output. This information (outputs and enables) is given by the user through a software accesible register interface.

Design

Using all the requirements and the basic understanding of the electronic chips on the board in this design stage the block structure, behaviour flow diagram and signals and register interface are described.

Figure 5.17 shows the flow diagram of the Digital Outputs module.

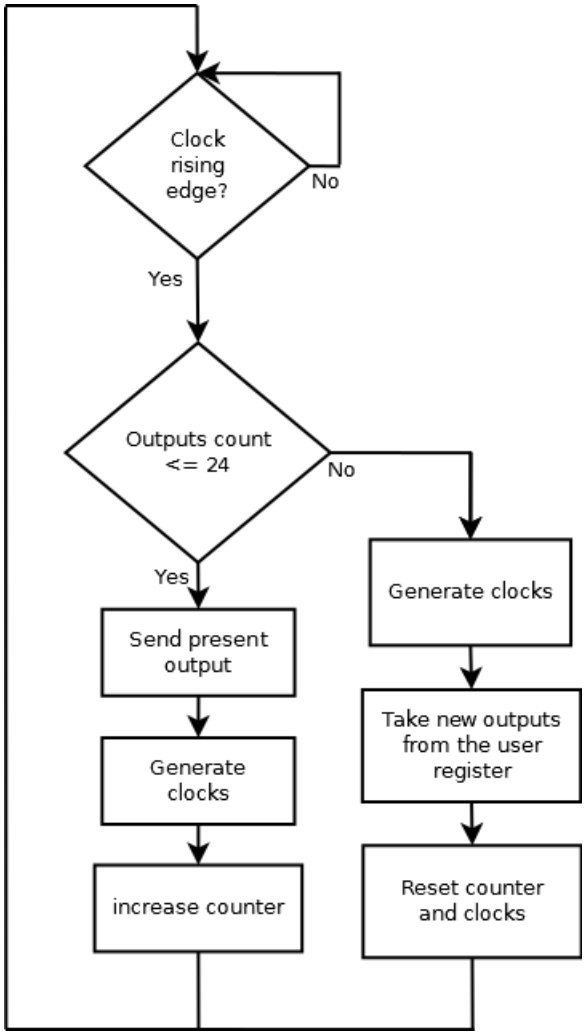

Figure 5.17: Flow Diagram of the Digital Outputs module

Figure 5.18 shows the block diagram of the Digital Outputs module.

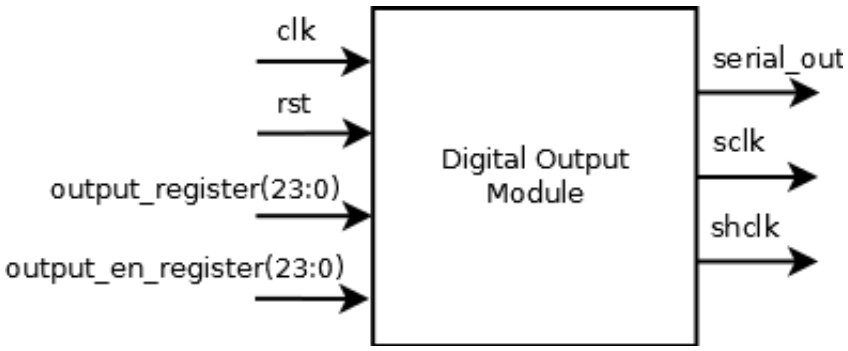

Figure 5.18: Digital Outputs module core

The inputs and outputs of the Digital Inputs module are described in table 5.6.

Signal	Type	Description
clk	Input	Reference clock signal
rst	Input	Active low reset signal
output_register(23:0)	Input	Register interface to software
en_output_register(11:0)	Input	Register interface to software
serial_out	Output	Serial output to *74HC595D*
sclk	Output	Store clock to *74HC595D*
shlk	Output	Shift clock to *74HC595D*

Table 5.6: Digital Outputs module signals

Implementation

In order to implement in VHDL the module, only one process is used. This process controls by using a counter, the number of outputs sent to the serial output at the moment of a clock rising edge. Once all the 24 outputs and the 12 enable signals are sent, this process captures the new 24 outputs and enables from the software registers and initiates a new send cycle. Depending on the send process, the clock signals are generated in a proper way to control the electronic logic.

Other important point is the PLB wrapper of this peripheral. Due to his two input registers (write registers), the PLB wrapper template must be created to have two software accesible registers.

Performance

The performance of the peripheral is described in the next list:

- Maximum frequency: it is limited by the *74HCT166D* maximum frequency to around 35 MHz
- Performance: A complete read cycle takes 42 clock ticks, which means a maximum performance of 1200 ns to update the 24 digital outputs
- FPGA resources: this module needs 22 slices of the PFGA to be implemented

5.10.3 Analog Inputs

In this section, the development process of the peripheral for the control of the analog inputs is explained. The core is written in VHDL.

Analysis

In order to describe in VHDL the behaviour of this interface, it is needed a full understanding of how the electronics on the board work. To acomplish the task, the board has a *AD7927* analog to digital converter, which performs the task of converting analog inputs in digital values

suitable to be procesed.

Figure 5.19 shows the block diagram of the analog to digital converter.

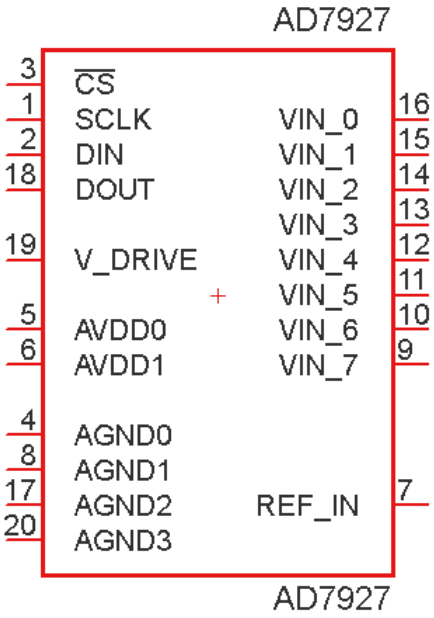

Figure 5.19: AD7927 analog to digital converter

As seen in Figure 5.19, the converter has eight analog channels, either one corresponds to an analog input. The AD7927 has also a clock signal, a chip select, and two serial interfaces, one to receive configuration data and the other to send the result of the conversions.
For the analog inputs module, the four control signals are needed and connected to the FPGA. The *AD7927* requires a well determined power up secuence next explained.

When supplies are first applied to the AD7927, the ADC may power up in any of the operating modes of the part. To ensure that the part is placed into the required operating mode, the user should perform a dummy cycle operation as outlined in Figure 5.20.

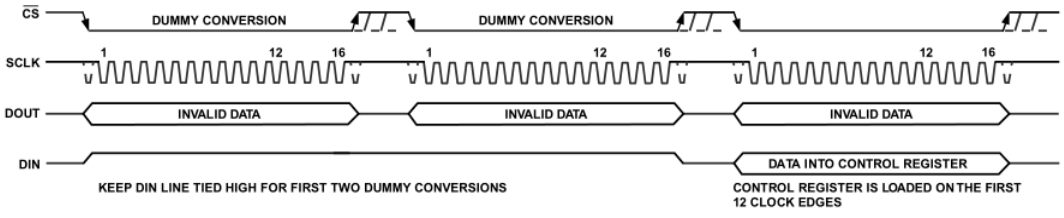

Figure 5.20: AD7927 power up

The three-dummy-conversion operation outlined in Figure 5.20 must be performed to place the part into the auto shutdown mode. The first two conversions of this dummy cycle operation are

performed with the DIN line tied high, and for the third conversion of the dummy cycle operation, the user should write the desired control register configuration to the AD7927 to place the part into the auto shutdown mode. On the third CS rising edge after the supplies are applied, the control register contains the correct information and valid data results from the next conversion.

In the particular case of the system to be developed, it is needed to update all eight channels. The converter has a mode, which allows the converter to perform continuous conversions for a sequence of channels. In order to set this mode, the following configuration must be sent to the converter:

- Set normal mode: this ensures the fastest throughput rate performance
- Set sequencial mode: this mode allows the converter to perform consecutive convertions without modifying the control register
- Set range to $2 \cdot V_{ref}$: this allows full range from 0 to 10 Volts
- Set binary mode

Figure 5.19 shows the flow diagram of the configuration and operation mode of the converter.

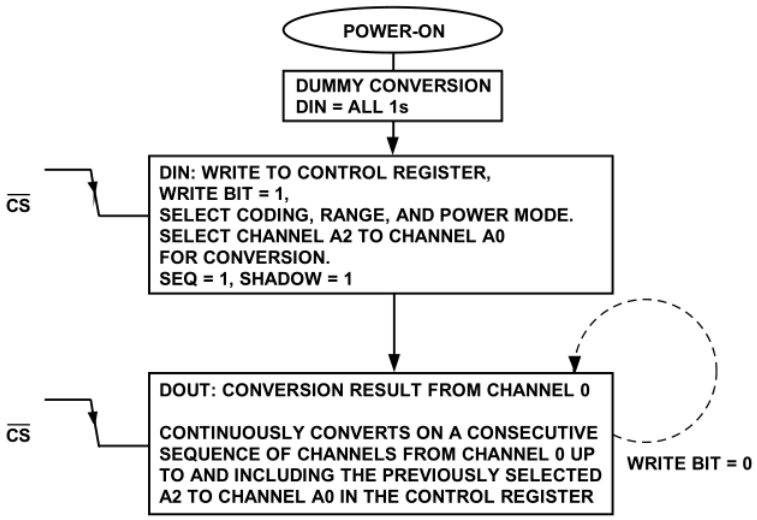

Figure 5.21: AD7927 configuration and operation mode

Design

Using all the requirements and the basic understanding of the electronic chips on the board in this design stage the block structure, behaviour flow diagram and signals and register interface are described.

Figure 5.22 shows the flow diagram of the Digital Outputs module.

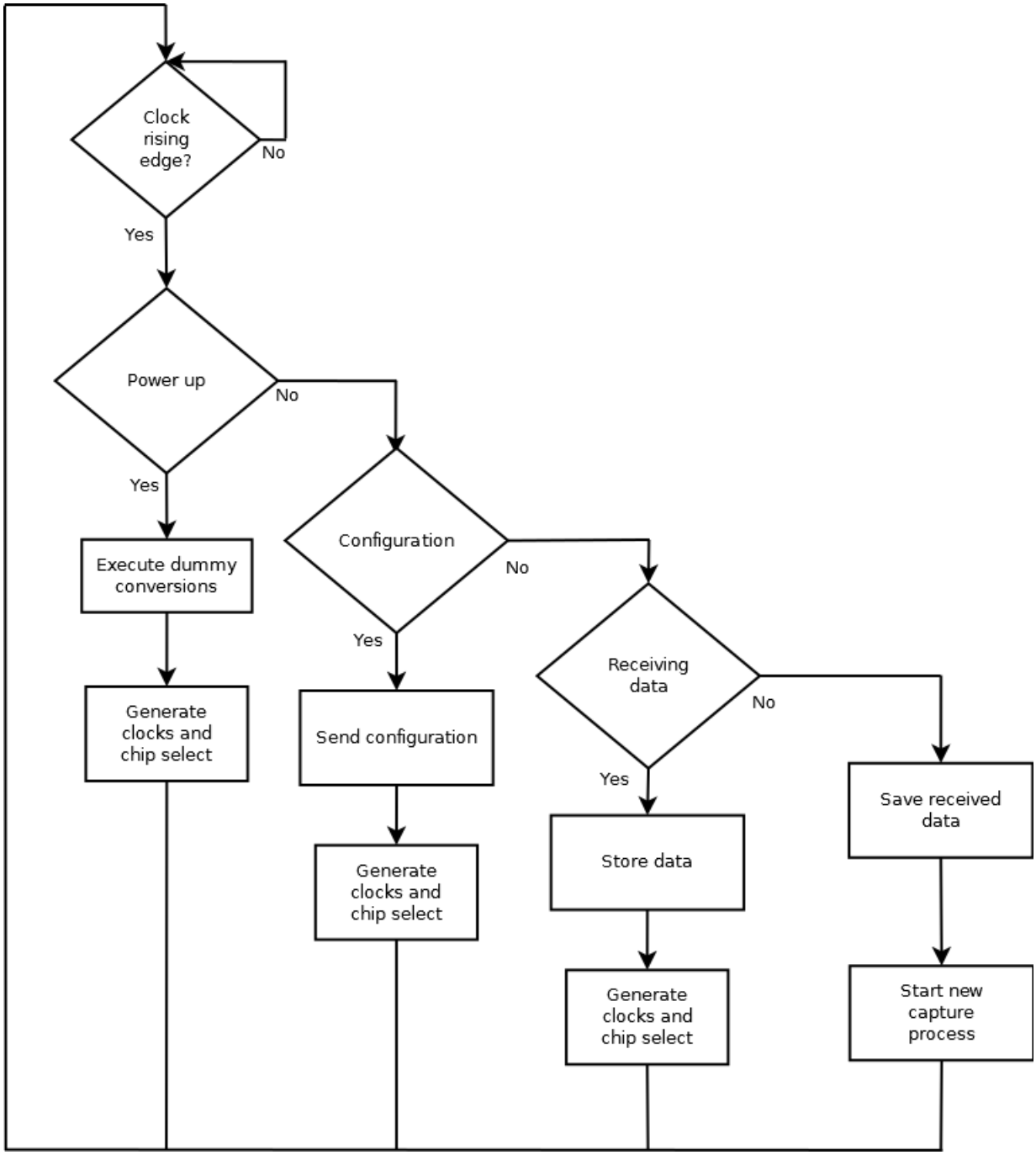

Figure 5.22: Flow Diagram of the Analog inputs module

Figure 5.23 shows the block diagram of the Analog inputs module.

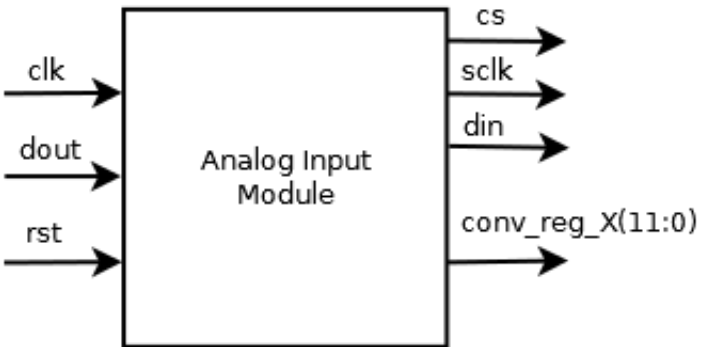

Figure 5.23: Analog inputs module core

Note that there are 8 conversion register but in the diagram a general one is represented to simplify.

The inputs and outputs of the Digital Inputs module are described in table 5.7

Signal	Type	Description
clk	Input	Reference clock signal
rst	Input	Active low reset signal
dout	Input	Serial out of *AD7927*
cs	Output	Chip select signal to *AD7927*
sclk	Output	Clock signal to *AD7927*
din	Output	Serial input of *AD7927*
conv_reg_0(11:0)	Output	Conversion channel 0 register
conv_reg_1(11:0)	Output	Conversion channel 1 register
conv_reg_2(11:0)	Output	Conversion channel 2 register
conv_reg_3(11:0)	Output	Conversion channel 3 register
conv_reg_4(11:0)	Output	Conversion channel 4 register
conv_reg_5(11:0)	Output	Conversion channel 5 register
conv_reg_6(11:0)	Output	Conversion channel 6 register
conv_reg_7(11:0)	Output	Conversion channel 7 register

Table 5.7: Analog Inputs module signals

Implementation

In order to implement in VHDL the module, there are 7 processes implemented. In the table 5.8 all the processes are described.

Process	Description
CS generator	Generates the chip select signal
DIN start	Sincronizes the CS signal with the start of sending data
Dummy	Performs the two dummy conversions in the power up of the converter
Configuration	Sends the configuration register to the converter
ACK configuration	Controls the correctly configuration of the converter
Read inputs	Reads the incomming frames from the converter and extracts the data
Save inputs	Saves the extracted data from the conversions in the registers

Table 5.8: Analog Inputs module processes

Other important point is the PLB wrapper of this peripheral. Due to his eight output registers (read registers), for the eight analog inputs, the PLB wrapper template must be created to have eight software accesible registers.

Performance

The performance of the peripheral is described in the next list:

- Maximum frequency: it is limited by the *AD7927* maximum frequency to 20 MHz
- Performance: A complete read cycle takes 128 clock ticks, which means a maximum performance of 6400 ns to update the 8 analog inputs
- FPGA resources: this module needs 76 slices of the PFGA to be implemented

5.11 Power on reset

5.11.1 Analysis

The designed custom board does not have any switch or button that can be used as a global or initial reset, so to do this task a power on cirtuit was developed from scratch using VHDL.
A power-on reset (PoR) generator is a microcontroller or microprocessor peripheral that generates a reset signal when power is applied to the device. It ensures that the device starts operating in a known state.

5.11.2 Design

Figure 5.24 shows the flow diagram of the Power on Reset, which describes the behaviour of the module.

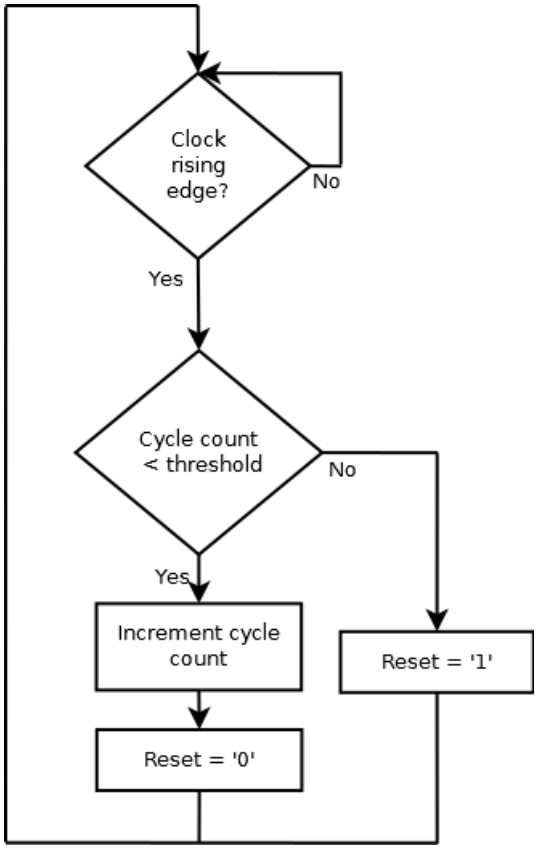

Figure 5.24: Flow Diagram of the Power on Reset core

Figure 5.25 shows the block diagram of the Power on Reset.

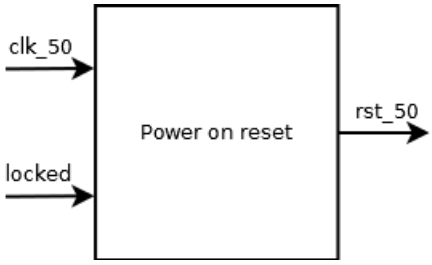

Figure 5.25: Power on Reset core

The inputs and outputs of the Power on Reset are described in table 5.9.

Signal	Type	Description
clk_50	Input	Reference clock signal
locked	Input	Active high when clk_50 is stable
rst_50	Output	Active low reset signal

Table 5.9: Power on Reset signals

5.11.3 Implementation

In order to implement in VHDL the module, only one process is used. This process controls, by using a counter, the number of clock cycles for which the reset has to be maintained to low level. After the count, the reset is released.

5.12 System Timer

A hardware timer must be included in the system in order to measure the time performance and as compulsory requirement to run the Xilkernel. Xilkernel uses this timer for multithreading schduling purposes. The selected hardware timer was the *LogiCORE IP XPS Timer/Counter* included in the Xilinx hardware development tools and easily configurable.

This timer can be managed manually by the user if a *Xilinx Standalone* system is implemented. The user can configure and access directly, in the source code, to the functions of the timer writing and reading from the hardware registers using the input/output functions provided by the *Xilinx Standalone* layer.
If a *Xilinx Xilkernel* operative system is implemented, the timer must be configured prior to the compilation of the operative system. Once compiled, the user can access to the timer's functionality via functions provided in the *Xilinx Xilkernel* API. A detailed explanation of the configuration parameters is given in the chapter *Software Development*.

To get a more detailed vision of the timer, please see the Xilinx documentation in [19].

5.13 Interrupt Controller

There are some peripherals in the system, which produce interruption signals when an important event has taken place. These events could be: full reception buffers from a communication peripheral like the CAN transceivers for example, a timer finished count and more. Because of that, it is very recommended to have an interrupt controller in the system. For this purpose, the *LogiCORE IP XPS Interrupt Controller*, provided by Xilinx, was chosen. This Interrupt Controller concentrates multiple interrupt inputs from peripheral devices to a single interrupt output to the system processor.

The Interrupt Controller is a compulsory requirement to run the *Xilinx Xilkernel* and the interrupt port if the timer explained in the previous section must be connected to the Interrupt Controller.

For additional information, please see the Xilinx documentation in [20].

5.14 LEDS

The board has four leds, which can be used for debug or signalization purposes. In order to control the switching-on and switching-off of the leds in the software, an input/output peripheral is needed. Xilinx provides the *XPS GPIO* core for that.

5.14.1 GPIO

The XPS GPIO design provides a general purpose input/output interface to a Processor Local Bus (PLB). The XPS GPIO can be configured as either a single or a dual channel device. The channel width is configurable and when both channels are enabled, the channel width of each of the channels can be configured individually. The ports can be configured dynamically for input or output by enabling or disabling the 3-state buffer. The channels may be configured to generate an interrupt when a transition on any of their inputs occurs.

The major interfaces and modules of the pripheral are shown in figure 5.26:

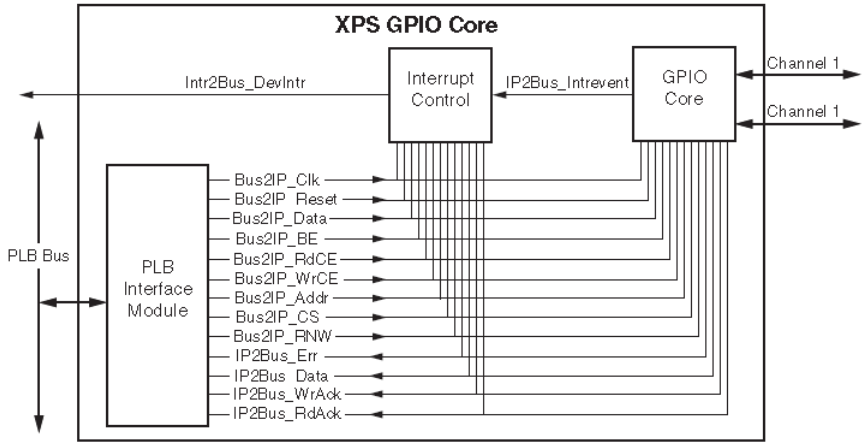

Figure 5.26: Xilinx GPIO

For additional information, please see the Xilinx documentation in [21].

5.15 UART

The Embedded System has to comunicate with the exterior for information exchange with other external systems and for debug and data visualization purposes. Xilinx provides in his EDK development environment the *XPS UARTlite* core, which implements a very basic UART. Although it is a very basic UART, for the needs of system being developed it is adequate. In the system to be developed, two UARTs are implemented and their function is described in the following sections.

5.15.1 XPS Uartlite

The XPS UART Lite performs parallel-to-serial conversion on characters received through PLB and serial-to-parallel conversion on characters received from a serial peripheral. It is capable of transmitting and receiving 8, 7, 6 or 5-bit characters, with 1-stop bit and odd, even or no parity, and transmit and receive independently.

The device can be configured and its status can be monitored via the internal register set. The XPS UART Lite generates an interrupt when Receive FIFO becomes non-empty or when transmit FIFO becomes empty. This interrupt can be masked by using an interrupt enable/disable signal. It also contains a 16-bit programmable baud rate generator and independent 16-word Transmit and Receive FIFOs, which can be enabled or disabled through software control.

The XPS UART Lite modules are shown in the top-level block diagram in figure 5.27:

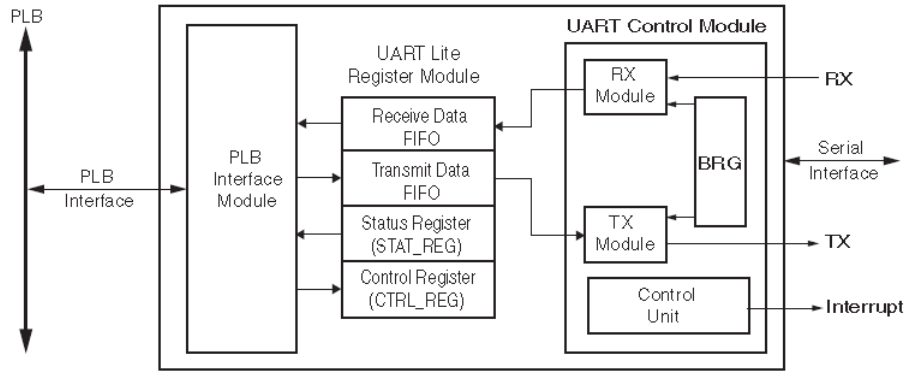

Figure 5.27: XPS UARTlite core

For additional information, please see the Xilinx documentation in [22].

5.15.2 STDIO UART

This UART is used as the standard input/output of the system, which comprises all the input/output functions like *printf*, *scanf* and others. Table 5.10 shows the configuration parameters:

Parameter	Value
Baud rate	9600
Parity	Even
Data bits	8
Stop bits	1

Table 5.10: STDIO UART configuration

5.15.3 Communication UART

This UART is used for communication purposes between the Embedded System and the Siemens Industrial PC. The next table shows the configuration parameters:

Parameter	Value
Baud rate	115200
Parity	None
Data bits	8
Stop bits	1

Table 5.11: Communication UART configuration

5.15.4 UART Enable GPIO

The two built-in UARTS have an switching-on/off connection. If not used, a UART can be deactivated in oder to save power. To control this two activation outputs a GPIO peripheral is included in the system.

5.16 Clock generators

The Microblaze system to be developed has different clocks for the different components in the system. The board has a 50 MHz Crystal oscillator, which produces the input clock signal of the system. This clock signal is conected to a Xilix clock generator IP, which produces a stabilized clock output.

5.16.1 System Clock Generator

The Clock Generator core takes in common clock requirement through its parameters and generates the architecture-specific clocking circuitry. The circuitry is implemented in a VHDL source. When the Clock Generator cannot generate circuitry for the given requirement, it provides failure analysis. The generation algorithm is implemented in C++ programming language and currently it is only integrated with EDK implementation tool, PlatGen and SimGen. It supports up to 16 different clock requirements and generates synthesizable structural VHDL code [18].

Figure 5.28 shows the block diagram of the System Clock Generator of Xilinx for this specific system.

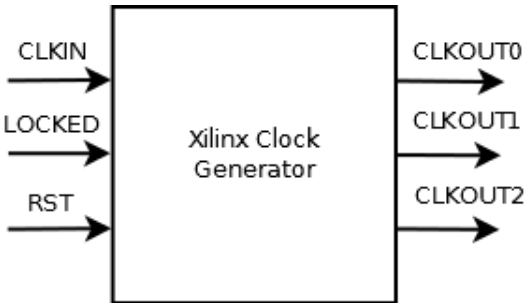

Figure 5.28: Clock Generator core

The inputs and outputs of the System Clock Generator are described in table 5.12.

Signal	Type	Description
CLKIN	Input	Reference clock signal
RST	Input	Active high reset signal
LOCKED	Output	Active high when the output clocks are stabilized
CLKOUT0	Output	Clock output 0
CLKOUT1	Output	Clock output 1
CLKOUT2	Output	Clock output 2

Table 5.12: Clock Generator core signals

The clock output frequency values and use are described in the section 5.16.3 (System Clocks).

5.16.2 Peripheral Clock Divider

There is an existing limitation in the previous System Clock Generator core because the core can generate frequencies from 1 MHz to 1 GHz but not frequencies lower than 1 MHz. In the system there are some peripherals, which do not need such high frequencies to work but rather frequencies in the KHz range. To solve this drawback, a clock divider was developed from scratch using VHDL and later included in the embedded system.

Figure 5.29 shows the flow diagram of the Peripheral Clock Divider, which describes the behaviour of the module:

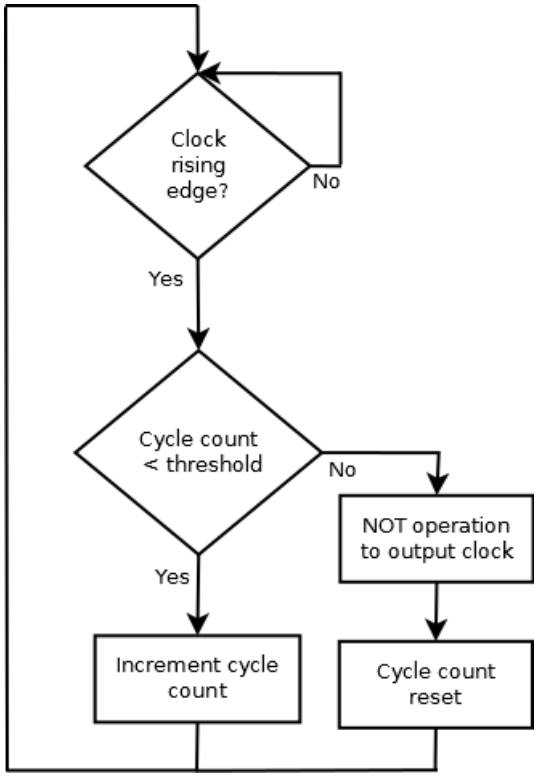

Figure 5.29: Flow Diagram of the Peripheral Clock Divider

Figure 5.30 shows the block diagram of the Peripheral Clock Divider:

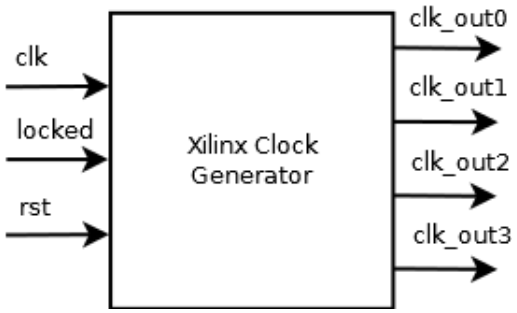

Figure 5.30: Peripheral Clock Generator core

To make the Peripheral Clock Divider configurable VHDL generics were used to determine the clock division factor. They are described in table 5.13.

Generic	Default value	Description
clk_0	600	Clock 0 division factor
clk_1	2000	Clock 1 division factor
clk_2	312	Clock 2 division factor
clk_3	2	Clock 3 division factor

Table 5.13: Peripheral Clock Divider core generics

The inputs and outputs of the Peripheral Clock Divider core are described in the table 5.14

Signal	Type	Description
clk	Input	Reference clock signal
rst	Input	Active low reset signal
locked	Input	Active high when input clock is stabilized
clk_out0	Output	Clock output 0
clk_out1	Output	Clock output 1
clk_out2	Output	Clock output 2
clk_out3	Output	Clock output 3

Table 5.14: Peripheral Clock Divider signals

The clock output frequency values and use are described in the section 5.16.3 (System Clocks).

5.16.3 System Clocks

Table 5.15 shows the different clock signals of the embedded system, who has generated the signal, the frequency and the modules, which use each clock signal.

Clock Signal	Generated by	Frequency	Clock Source of
CLK_S	Crystal oscilator	50 MHz	System Clock Generator, Power on Reset
CLKOUT0	System Clock Generator	50 MHz	Microblaze, PLB bus, Peripheral Clock Divider
CLKOUT1	System Clock Generator	10 MHz	CAN transceivers
CLKOUT2	System Clock Generator	6.25 MHz	Torque Vectoring
clk_out0	Peripheral Clock Divider	88.33 KHz	Digital outputs
clk_out1	Peripheral Clock Divider	25 KHz	Digital inputs
clk_out2	Peripheral Clock Divider	160.25 KHz	Analog inputs
clk_out3	Peripheral Clock Divider	2 KHz	Traction Control System

Table 5.15: System Clocks

Table 5.16 shows the different operation times of the peripherals of the embedded system.

Peripheral	Clock frequency	Operation time
Analog inputs	160.25 KHz	800 μs
Digital inputs	25 KHz	1 ms
Digital outputs	88.33 KHz	960 μs
Torque Vectoring	6.25 MHz	1120 ns
Traction Control	2 KHz	10 ms

Table 5.16: System operation times

5.17 Memory map

The complete memory map with all the components of the embedded system is described in the table 5.17.
Note that, as explained in section 5.2.2, the four memory controllers must be in a address range of four consecutive 64 KB addresse spaces in order to build a memory space for a total of 256 KBytes.

Component	Base Address	Offset	Bus	Description
dlmb_cntlr_3	0x00000000	0xFFFF	LMB	data memory controller 3
ilmb_cntlr_3	0x00000000	0xFFFF	LMB	instruction memory controller 3
dlmb_cntlr	0x00010000	0xFFFF	LMB	data memory controller 0
ilmb_cntlr	0x00010000	0xFFFF	LMB	instruction memory controller 0
dlmb_cntlr_1	0x00020000	0xFFFF	LMB	data memory controller 1
ilmb_cntlr_1	0x00020000	0xFFFF	LMB	instruction memory controller 1
dlmb_cntlr_2	0x00030000	0xFFFF	LMB	data memory controller 2
ilmb_cntlr_2	0x00030000	0xFFFF	LMB	instruction memory controller 2
RS232_activation	0x81400000	0xFFFF	SPLB	UART activation
LEDS	0x81420000	0xFFFF	SPLB	LEDs GPIO
xps_intc_0	0x81800000	0xFFFF	SPLB	LEDs GPIO
xps_timer_0	0x83C00000	0xFFFF	SPLB	System timer
xps_uartlite_0	0x84000000	0xFFFF	SPLB	STDIO UART
RS232	0x84020000	0xFFFF	SPLB	Communication UART
mdm_0	0x84400000	0xFFFF	SPLB	Hardware debugger
digital_output	0xC1E00000	0xFFFF	SPLB	Digital outputs
digital_input	0xC5000000	0xFFFF	SPLB	Digital inputs
analog_input	0xC5200000	0xFFFF	SPLB	Analog inputs
tcs_system_0	0xC7200000	0xFFFF	SPLB	Left side Traction Control
tcs_system_1	0xC7220000	0xFFFF	SPLB	Right side Traction Control
plbv46_2_wb_1	0xC9800000	0xFFFF	SPLB	CAN transceiver 1
plbv46_2_wb_0	0xC9820000	0xFFFF	SPLB	CAN transceiver 0
torque_vectoring	0xC9A00000	0xFFFF	SPLB	Torque Vectoring

Table 5.17: Embedded System Memory Map

5.18 Final system

Figure 5.31 shows the block diagram of the complete embedded system. In the system can be differentiated four types of peripherals connected to the PLB bus depending on the function. The fifth group is the set of peripherals, which are not connected to the PLB bus.

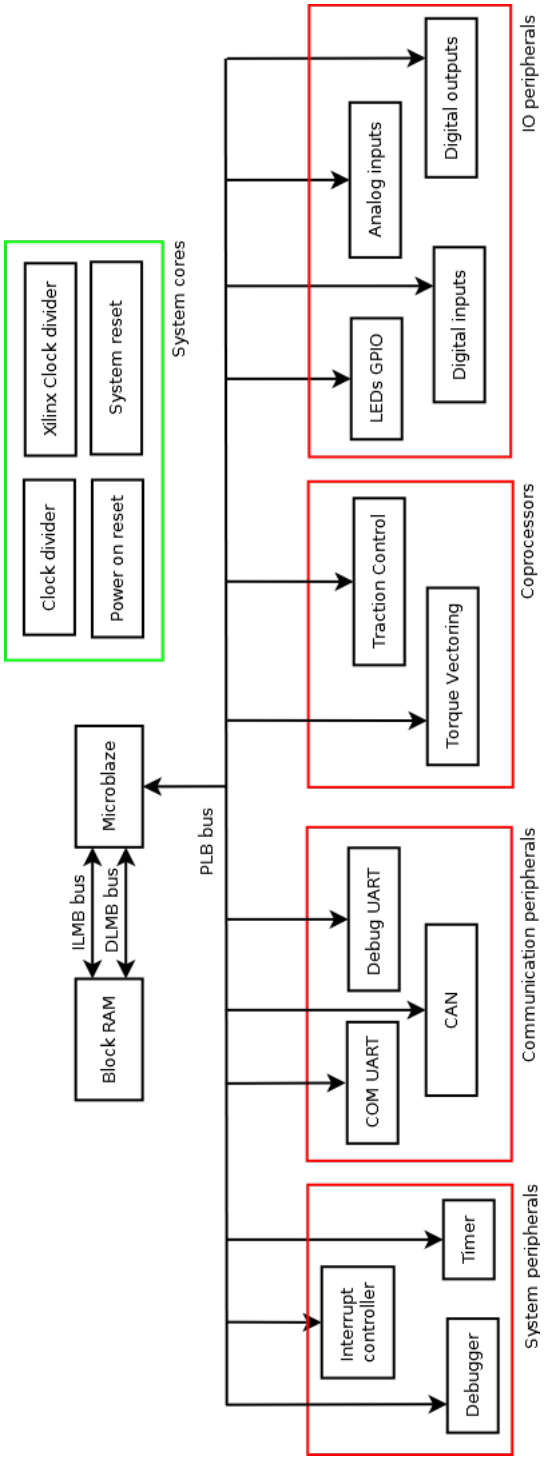

Figure 5.31: Embedded System

Chapter 6

Control Software development

In this chapter a detailled description of the control software development including all the stages in the software lifetime cycle is given. In this work the scope of the control software development is focused on the development of the new control algorithm and specification of the base software platform, including the information exchange protocols. During the development stage the Xilinx Software development tool was used.

6.1 System requirements

In this section, the requirements of the system are enumerated.

6.1.1 Functional requirements

- **Control loop**: The system must control all the automobile systems of the kart
- **Drive paramenters change**: The driver must be able to change the behaviour of the traction control and torque vectoring by selecting the desired mode
- **Traction control**: The system must apply a traction control regulation to the output torque when the wheels slip above the desired slip factor
- **Torque Vectoring**: The system must apply a torque vectoring regulation to the output torque depending on the direction of the actual curve
- **Telemetry**: The system must store relevant telemetry data in order to later analyse this data and extract conclussions.
- **Communication with motor converters**: The software must send the torque to the motor converters
- **Information displaying**: The software must display the important debug or runtime information

6.1.2 Information requirements

- **Telemetry**: The desired storable telemetry data is:
  - Timestamp
  - Gas pedal
  - Global speed
  - Speed of each individual wheel
  - Applied torque to each motor
  - Slip of each side of the kart
  - Traction control factor of each side
  - Torque vectoring tunning factor
- **Communications**: The exchanged information with the motors is:
  - Torque
  - Speed

6.1.3 Non functional requirements

- **Security**: If a severe error occurs, the must be stoped as soon as possible
- **Reliability**: The communication must be reliable
- **Performance**: The system must execute the software at least as fast as the previous system
- **Portability**: The system must present an abstraction layer to secure the portability of the drivers.

6.2 Software Life model

The software development model used is the incremental model. With this model of development, a new version of the software is produced in each cycle. The software growns gradually in benefits and additional functionality. The features ,that had each of these versions, are described in the following pages.

6.2.1 First iteration

In the first iteration of the development, the firmware of the different hardware components was developed. A basic software platform for testing purposes was developed using a standalone operating system to have access to the low level functions of the system.

6.2.2 Second iteration

In the second iteration, the main software of the kart (the control loop) and his communication routines was developed. At this stage, the operating system was changed to a real time one with multithreading capabilities.

6.2.3 Third iteration

In the third iteration, a special test version of the control software was developed. That was a forced development due to the impossibility of controlling the Siemens converters (Profinet interface) with the existing hardware, software and documentation avaible. The important data for the calculations is sent to the Embedded System from the Siemens industrial PC.

6.3 Analysis

6.3.1 Use Case model

Actors

This section describes the different roles played by users interacting with the system. The roles of actors can be individuals, external systems or even the time (temporal events).

Table 6.1 decribes the actors of the system.

Actor	Description
Driver	It is the main user of the system
Time	Time triggers some events and processes
Sensors	Data comes to the system and changes his behaviour

Table 6.1: Actors of the system

Use Case diagram

Figure 6.1 represents the Use Case diagram, which shows the main functionality of the system at the highest abstraction level.

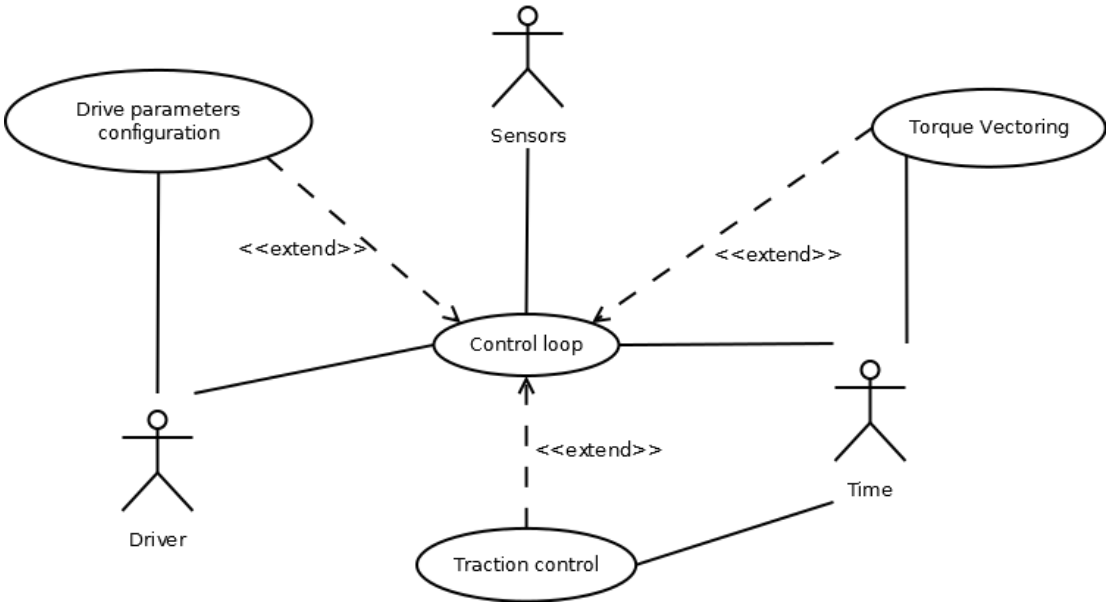

Figure 6.1: Control software Use Case diagram

In the following pages, all the use cases are explained.

Use Case: Control loop

- **Use Case**: Control loop
- **Scenarios**:
  - Main: Normal control loop
  - Alternative 1: Normal control loop with traction control
  - Alternative 2: Normal control loop with torque vectoring
- **Description**: Controls all the functionality of the Kart
- **Actors**: Driver, time and sensors
- **Preconditions**: The software system must be initialized
- **Postconditions**: All must be stoped and all the resources released
- **Main Scenario**:
  1. The driver starts the system
  2. The system initializes all his peripherals
  3. The system configure the software timers
  4. The driver configures the parameters of the system, e.g. Torque vectoring and Traction Control
  5. Update paramenters if changes were made.

6. The system updates all the inputs
7. The system processes the inputs
8. The system checks if Traction Control or/and Torque Vectoring are set
9. The system do the calculations
10. The system updates all the outputs, including communication interfaces
11. Repeat from 4

- **Extensions**:

  8.a If Traction Control set, extend to Use Case "Traction Control".

  8.b If Torque Vectoring set, extend to Use Case "Torque Vectoring".

  9.a If an error occurs, depending on the severity, inform or stop completely the system.

Use Case: Drive parameters configuration

- **Use Case**: Drive parameters configuration
- **Scenarios**:
  - Main: Change drive parameters
- **Description**: Updates the drive parameters like Torque Vectoring setting and Traction Control
- **Actors**: Driver
- **Preconditions**: None
- **Postconditions**: Drive parameters are changed
- **Main Scenario**:
  1. The driver wants to change the drive parameters
  2. The driver adjusts the Torque Vectoring factor and response
  3. The driver adjusts the Traction Control
  4. The system stores the new configuration

Use Case: Torque Vectoring

- **Use Case**: Torque Vectoring
- **Scenarios**:
  - Main: Do the Torque Vectoring calculation
- **Description**: Do the Torque Vectoring calculation by using the dedicated coprocessor
- **Actors**: Time, Sensors

- **Preconditions**: Torque Vectoring is set to activated
- **Postconditions**: Torque Vectoring correction factor is calculated
- **Main Scenario**:
    1. The system wants to calculate the Torque Vectoring response (Torque Vectoring timer is triggered)
    2. The system sends to the coprocessor the actual speed of the car, the steering angle and the torque vectoring factor.
    3. The coprocessor processes the data meanwhile the system does other tasks
    4. The system updates the value calculated by the coprocessor.
    5. The system restarts the Torque Vectoring timer.

Use Case: Traction Control

- **Use Case**: Traction Control
- **Scenarios**:
    - Main: Do the Traction Control calculation
- **Description**: Do the Traction Control calculation by using the dedicated coprocessor
- **Actors**: Time, Sensors
- **Preconditions**: Traction Control is set to activated
- **Postconditions**: Traction Control correction factor is calculated
- **Main Scenario**:
    1. The system wants to calculate the Traction Control correction factor (Traction Control timer is triggered)
    2. The system sends to each coprocessor (one for the left side, one for the right side) the actual slip of the left and right side respectively.
    3. The coprocessor processes the data meanwhile the system does other tasks
    4. The system updates the value calculated by the coprocessor.
    5. The system restarts the Traction Control timer.

6.3.2 Data model

The system is divided in different modules to split the functionality in input/output, communication and calculations. Most of the modules are written as a firmware to control the hardware on the board. Figure 6.2 represents the Data model diagram, which shows the data modules, their attributes and the relation between them. The attributes of each entity are described in tables.

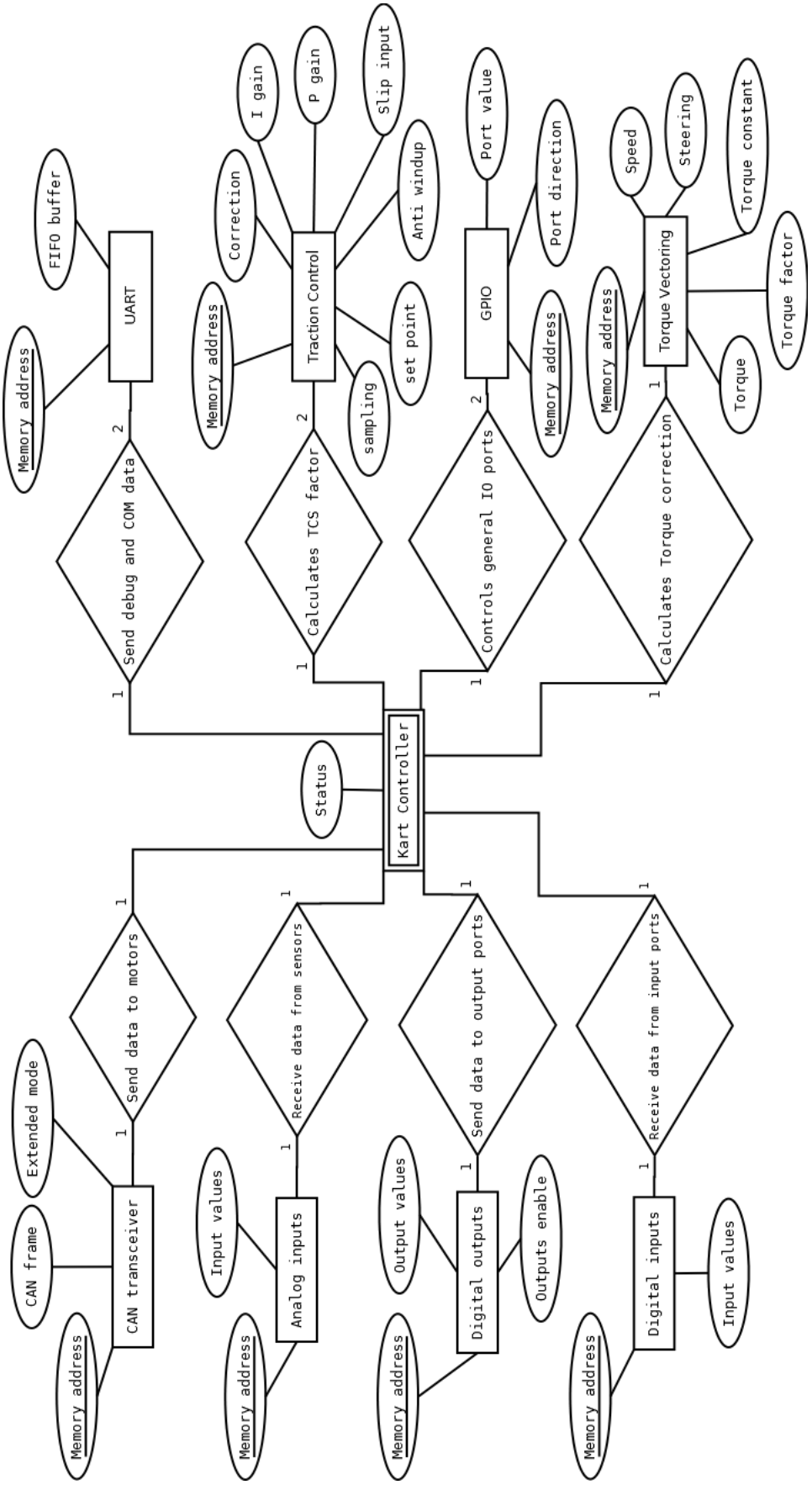

Figure 6.2: Control software Data model diagram

Digital inputs

Attribute	Type	Description
Memory address	Integer	Unique memory address of the peripheral
Input values	Boolean vector	Digital inputs value

Table 6.2: Entity: Digital Inputs

Digital outputs

Attribute	Type	Description
Memory address	Integer	Unique memory address of the peripheral
Output values	Boolean vector	Digital outputs value
Outputs enable	Boolean vector	Digital outputs enable

Table 6.3: Entity: Digital Outputs

Analog inputs

Attribute	Type	Description
Memory address	Integer	Unique memory address of the peripheral
Input values	Boolean vector	Analog inputs value

Table 6.4: Entity: Analog Inputs

CAN Transceiver

Attribute	Type	Description
Memory address	Integer	Unique memory address of the peripheral
CAN frame	CAN frame type	Contains the ID, extended ID and 8 Bytes of data
Extended mode	Boolean	Extended mode active

Table 6.5: Entity: CAN transceiver

UART

Attribute	Type	Description
Memory address	Integer	Unique memory address of the peripheral
FIFO buffer	Char vector	16 Byte send/receive FIFO

Table 6.6: Entity: UART

GPIO

Attribute	Type	Description
Memory address	Integer	Unique memory address of the peripheral
Port value	Boolean vector	Value of the input/output port
Port direction	Boolean vector	Set the port as input/output

Table 6.7: Entity: GPIO

Traction Control

Attribute	Type	Description
Memory address	Integer	Unique memory address of the peripheral
Sampling	Float	Sample time for the PI controller
Set point	Float	Set point for the desired wheel slip
Anti windup	Float	Anti windup gain
Slip input	Float	Actual slip of the wheels
P gain	Float	P stage gain
I gain	Float	I stage gain
Correction	Float	Traction Control gas correction factor

Table 6.8: Entity: Traction Control

Torque Vectoring

Attribute	Type	Description
Memory address	Integer	Unique memory address of the peripheral
Speed	Float	Speed of the kart
Steering	Float	Steering angle of the steering wheel
Torque constant	Float	Torque vectoring constant
Torque factor	Float	Torque multiplicative constant
Torque	Float	Torque correction factor

Table 6.9: Entity: Torque Vectoring

6.3.3 Behaviour model

In this section, the behaviour model of the software system at the analysis stage is described. The used tools are state diagrams, which explains the real behaviour of the system at high level of abstraction.

Figure 6.3 represents the States model diagram.

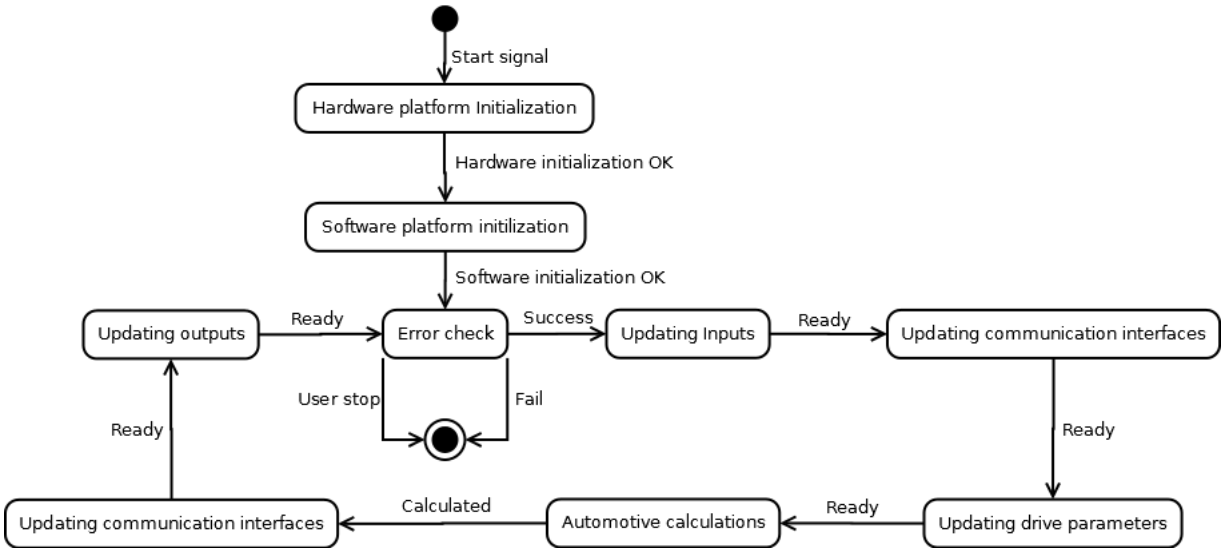

Figure 6.3: Control software State model diagram

Each single state is described in the following list.

- **Hardware Platform Initialization**: Initializates the hardware platform (peripherals).
- **Software Platform Initialization**: Initializates the operating system, threads and timers.
- **Error check**: Test if an error (hard/soft) or stop signal from the driver occurs.
- **Updating Inputs**: Updates digital and analog inputs of the system.
- **Updating Communication interfaces (input)**: Updates information from input communication buffers.
- **Updating drive parameters**: Updates the traction control and torque vectoring parameters.
- **Automotive calculations**: Calculates traction control and torque vectoring.
- **Updating Communication interfaces (output)**: Updates information to output communication buffers.
- **Updating Outputs**: Updates digital outputs of the system.

6.3.4 External interfaces

In this section, the external interfaces of the software system are explained. It can be differentiated two types of interfaces, the communication interfaces and the peripehrals and coprocessors of the system.

Communication

The communication interfaces are those, whose task is to exchange information between two computation systems. CAN bus is the communication interface used to send data to the motors. The following list shows the important information of this interface

CAN bus

- Related processes: CAN information exchange task
- Origin: Embedded System
- Destination: Motor converters
- Exchanged data: CAN frames described in table 6.10
- Periodicity: It depends on the speed of the system, but a maximum of 1 Mbit per second can be sent
- Triggering event: CAN timer
- Security requirements: Data must be in the valid range, otherwise the motors could be damaged. Ranges shown in Table 6.11

ID	Signal	Data	Description
0x301	Ctrl_Enable	Byte_0(0)	Turn on/off the controler
0x301	Ctrl_Mode	Byte_0(3:1)	Control mode of the controller
0x301	Ctrl_TargetSetpoint	Byte_2(15:8) Byte_1(7:0)	Set point of torque
0x311	Ctrl_Status1_ActSpeed	Byte_6(15:8) Byte_5(7:0)	Actual speed

Table 6.10: CAN frames

ID	Signal	Type	Size (bits)	Range
0x301	Ctrl_Enable	Bit	1	1
0x301	Ctrl_Mode	Unsigned	3	0..0
0x301	Ctrl_TargetSetpoint	Signed	16	-10000 to 10000
0x311	Ctrl_Status1_ActSpeed	Signed	16	-32768 to 32767

Table 6.11: CAN frames data range

Peripherals

The communication with the hardware peripherals is done according to the following specification.

- Related processes: Control loop
- Origin: Embedded System
- Destination: On-board peripherals
- Exchanged data: State of the inputs and activation of the outputs.
- Periodicity: Execution in the control loop
- Triggering event: None
- Security requirements: None

6.4 Design

6.4.1 System architecture

The chosen architecture for the system is the basic input-process-output architecture. The input-process-output (IPO) model, also known as the IPO+S model, is a functional model and conceptual schema of a general system. The IPO architecture identifies a program's inputs, its outputs, and the processing steps required to transform the inputs into the outputs. Data has to flow into the system in some form. Input is the data flowing into the system from outside. The next stage in the information flow is the input data being manipulated in some way. Processing is the action of manipulating the input into a more useful form. Output is the information flowing out of the system.

Packet diagram

Figure 6.4 shows the packet diagrams, which gives the modular organization of the software, and the functionality of each software module.

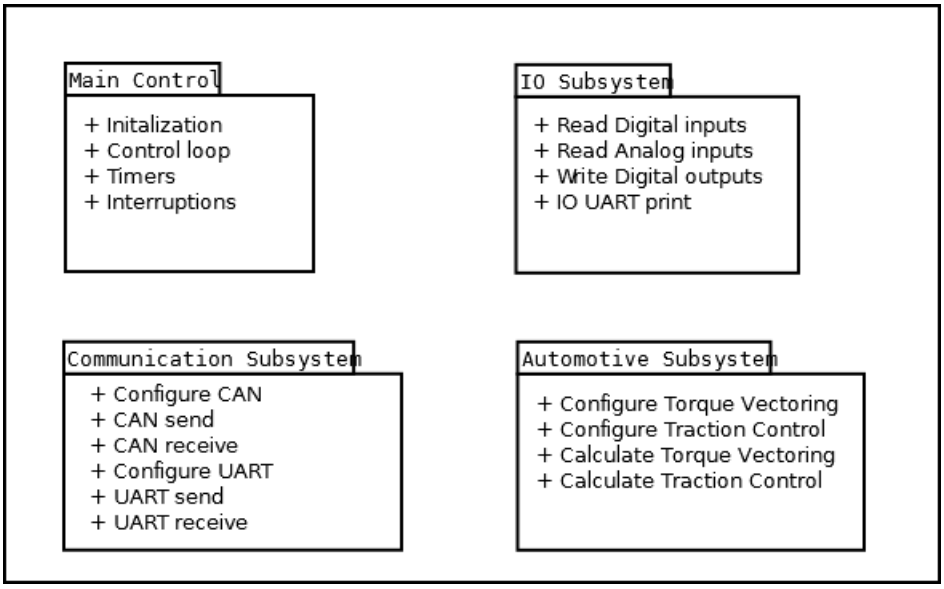

Figure 6.4: Control software Packet diagram

6.4.2 Behaviour model

In order to describe the behaviour of the system, Data Flow Diagrams are used in the structured programming paradigm to describe this behaviour. DFDs are divided in different levels of abstraction, starting at level 0, which represents the highest abstraction level and finishing at level 3, the most precise description of a single module of the system. In the next pages, the DFDs of the system are presented.

Figure 6.5 represents the level 0 Data Flow Diagram.

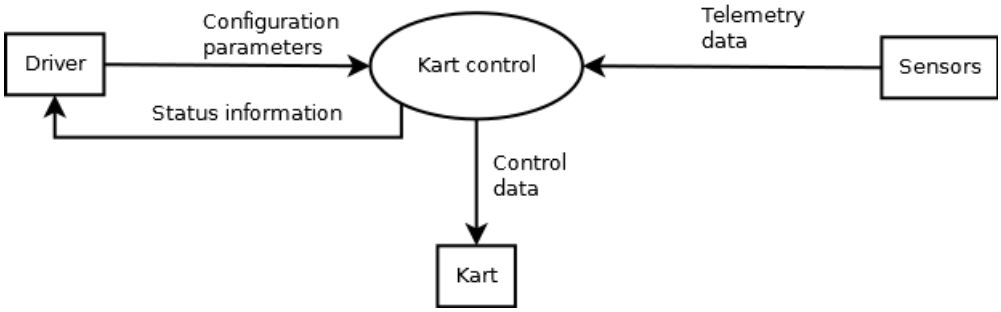

Figure 6.5: DFD level 0

As external entities, the driver is the user of the system and has the ability of changing the behaviour of the system by changing the drive parameters, such as the strength of the torque vectoring, the brake balance and the traction control. The sensors provide the data for the automotive calculations and the kart receives the control signals. There is a global process, called Kart Control, which performs the control routine.
Figure 6.6 represents the level 1 Data Flow Diagram.

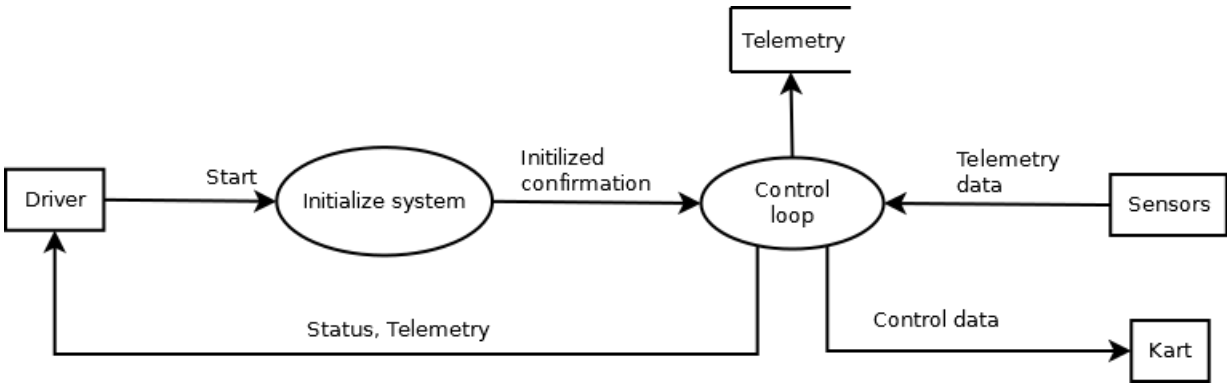

Figure 6.6: DFD level 1

Decreasing to a lower level of abstraction, the Control routine executes to main processes, the initialization process, which sets up the hardware to a initial working state, and the main control routine, which produces besides the control signals, the telemetry data, which is stored for a further necessity.

Figure 6.7 represents the level 2 Data Flow Diagram for the initilization process.

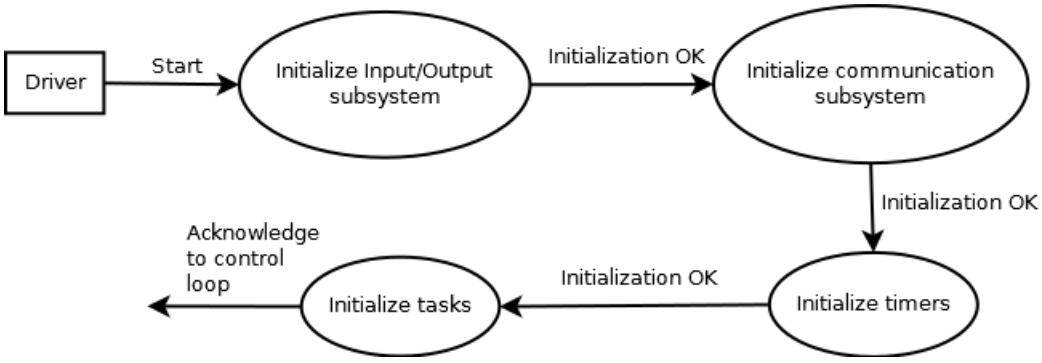

Figure 6.7: DFD level 2 initialization

In a more lower level of abstraction, the initialization routine has the mission of setting up the IO modules, communication modules, timers and working threads of the system.

Figure 6.8 represents the level 2 Data Flow Diagram for the control process.

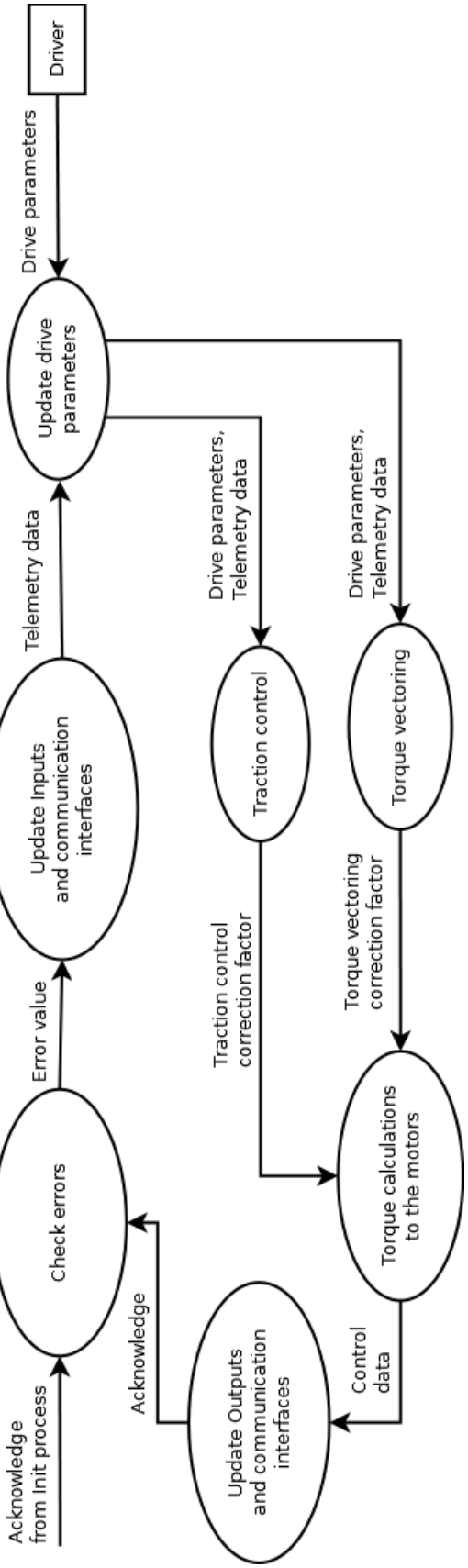

Figure 6.8: DFD level 2 control

The control routine is the most important part of the software. It performs all the automotive calculations, such as the traction control and the torque vectoring. This two calculations, when needed, are parallel performed, taking advantage of the parallel processing of the hardware. Figure 6.9 represents the level 3 Data Flow Diagram for the traction control process.

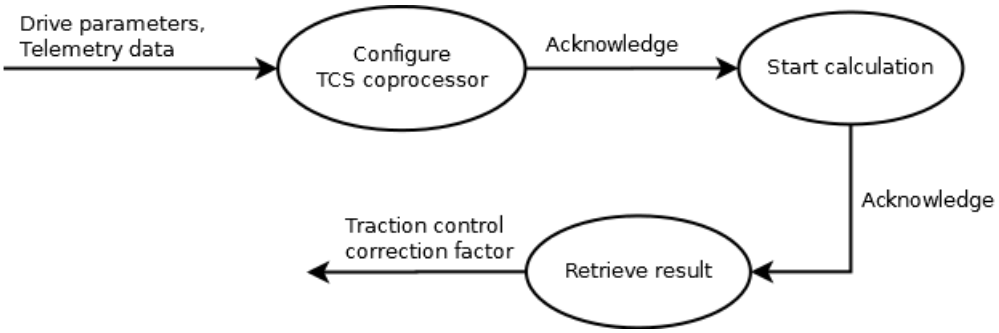

Figure 6.9: DFD level 3 TCS

Figure 6.10 represents the level 3 Data Flow Diagram for the torque vectoring process.

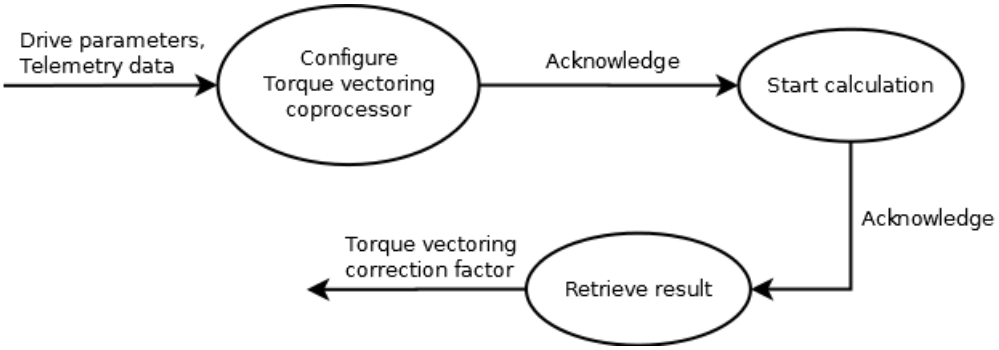

Figure 6.10: DFD level 3 Torque Vectoring

In the lower level of abstraction (both figures 6.9 and 6.10), the parameters needed for the calculations are sent to the hardware coprocessors and after the calculation time, the result is retrieved by the software

6.5 Implementation

6.5.1 CAN bus

The CAN bus controller must be configured in a proper way to match the specifications of the converters. Table 6.12 specifies the configuration.

Speed	1 Mbit/s
Mode	Normal
Sampling	Triple
Reception filter	0x00000301
Reception mask	0x1FFFFFFF

Table 6.12: CAN transceiver configuration

6.5.2 Multi-threading

Due to certain asynchronous communication tasks, a Multi-threading enviroment is required. The software is structured in two threads, the main thread executes the control loop and shares a memory zone with a secondary thread in charge of be receiving and sending information. The shared information between the two threads is resumed in Table 6.13.

Item	Type	Description
speed_left	float (input)	Speed of the left side motor
speed_right	float (input)	Speed of the right side motor
torque_left	float (output)	Torque to the left side motor
torque_right	float (output)	Torque to the right side motor

Table 6.13: Control software loop shared data

6.5.3 Xilkernel Operating System

As explained in section 4.2.2, Xilkernel is the chosen Operating System to support all the control software of the system. Because Xilkernel is a modular configurable operating system, it is necesary to configure individually each functionality of the system. By adding only the necesary functionality, the operating system compiles to the smallest size possible, which is very important to fullfill the memory requirements. In the next sections, the configuration of the customized Xilkernel version for the embedded system is explained.

STDIO

The Standard Input/Output is an essential part of any standard computer system but also for embedded systems. It allows to see important information for debugging the system and also can be used to import or export data from the system, e.g. telemetry data and configuration parameters.

The only avaible option for the implemented hardware is to connect the STDIO to a UART and then use the input and output functions by using an hyperterminal. In section 5.15.2 the features of the UART for input/output purposes are explained. This UART has the name “xps_uartlite_0”, so this must be selected in the STDIO option configuration in the Xilkernel configuration wizard.

Multi-threading

Xilkernel was chosen because it is a real time operating system with multithreading support. Multithreading is very important in real time systems to reach the best possible performance by balancing all the task of the system. With a well balanced system, the CPU time is well exploited. In control real time systems there are also some communications processes which need to be executed periodically at a certain time interval and threads are the best solution to keep an efficient but readable source code.

The threads support must be activated in the Xilkernel configuration wizard but there are other important parameters to be configurated, such as the maximum number of threads or the stack size. All can be configurated in the mentioned configuration wizard.

Scheduler

In a multi-thread system there are two options to control execution of the threads:

1. Manually control of the execution
2. Using a Scheduler

Both give advantages and disadvantages depending on the target system but for this case, using a Scheduler was chosen. By using a Scheduler, the management of the threads is easier and the time requirements are kept.
The Shcheduler support must be activated in the Xilkernel configuration wizard but there are other important parameters to be configurated, such as the type of Scheduler. The chosen Scheduler is a Round Robin one, which gives the same amount of time to each thread. This is made to keep synchronized the control loop thread with the communication threads.

Mutex locks

The threads must often share variables or a buffer. To avoid the incorrect modification of a shared resource between the threads mutex are used. Mutual exclussion locks secure that this critical section of code is only accesed by one thread at time.

Mutex are by default disabled in the Xilkernel core, so it is necesary to activate them in the configuration wizard.

Timers

Timers are a basic piece and for the real time systems very important because it is the task of the timers to control the Schduler and support time measures. The timer must also, like the previous features, activated in the configuration wizard. The options to be configured are the following:

- “xps_timer_0” must be selected as the system timer
- The working frequency must be set to the base frequency of the system (50 MHz)
- The system timer interval must be configured to 1 ms. That means that every millisecond a kernel tick is generated. This tick is used for scheduling and timing measure purposes

Interrupt Controller

The Interrupt Controller is used by the Xilkernel for the timers interruptions but can also be used for catching external interruptions. The system timer must be specified in the configuration wizard, otherwise Xilkernel does not compile.

Summary

Figure 6.11 shows the final configuration as it was introduced in the Software Development tool.

Name	Value	Default	Type	Description
stdin	xps_uartlite_0	none	peripheral	Specify the instance name of the standard input peripheral
stdout	xps_uartlite_0	none	peripheral	Specify the instance name of the standard output peripheral
sysintc_spec	xps_intc_0	none	peripheral	Specify the instance name of the interrupt controller device driving system interrupts
◢ systmr_spec	true	true	boolean	Configure kernel timer parameters
systmr_dev	xps_timer_0	none	peripheral	Specify the instance name of the kernel timer device (Microblaze only):
systmr_freq	50000000	100000000	integer	Specify the clock frequency of the timer (Hz). For PPC it is the PPC405's frequency. For M...
systmr_interval	1	10	integer	Specify the time interval for each kernel tick (in milliseconds). This controls the CPU bud...
◢ config_pthread_support	true	true	boolean	Configure pthread support in the kernel
max_pthreads	10	10	integer	Maximum number of simultaneous threads that can be handled by the kernel
pthread_stack_size	1000	1000	integer	Size of the stack to be allocated for each thread
config_pthread_mutex	true	false	boolean	Configure pthread mutex lock support in the kernel
max_pthread_mutex	10	10	integer	Maximum number of mutex locks allocated and supported by the kernel at any point in ti...
max_pthread_mutex_waitq	10	10	integer	Length of each mutex lock's wait queue. Controls the maximum the number of processes...
static_pthread_table			array	Static specification of pthreads. These threads will be created at Xilkernel startup
◢ config_sched	true	true	boolean	Configure the scheduling scheme used by the kernel
sched_type	SCHED_RR	SCHED_RR	enum	Choose the global kernel scheduling policy
n_prio	32	32	integer	The number of priority levels if scheduling is priority based
max_readyq	10	10	integer	Length of each ReadyQ. This is the maximum number of processes that can be active at a...
◢ config_time	true	false	boolean	Configure time/timer related feature support in the kernel
max_tmrs	10	10	integer	Maximum number of soft timers that will be supported by the kernel at any point in time
▷ config_sema	false	false	boolean	Configure semaphore support in the kernel
▷ config_shm	false	false	boolean	Configure shared memory support in the kernel
▷ config_bufmalloc	false	false	boolean	Configure buffer memory pool allocation support in the kernel
▷ copyoutfiles	false	false	boolean	Copy OS files to user specified directory
▷ config_debug_support	false	false	boolean	Control various debugging features of the kernel
▷ enhanced_features	false	false	boolean	Configure enhanced features of the kernel

Figure 6.11: Xilkernel configuration

6.5.4 Linker Script and BMM file

There are two important files to be modified in order to load the software in the RAM memory of the system. In the next sections they are explained.

Linker Script

The final step in creating an executable from object files and libraries is linking. This is performed by a linker which accepts linker command language files called linker scripts. The primary purpose of a linker script is to describe the memory layout of the target machine, and specify where each section of the program should be placed in memory.

Xilinx SDK provides a linker script generator to simplify the task of creating a linker script. The linker script generator GUI examines the target hardware platform and determines the available memory sections. The only action required is to assign the different code and data

sections in the ELF file to different memory regions.

Due to the four block RAM memory controllers added to the system, the linker script file must be modified to create a new contiguous memory space joining the four address spaces of the controllers.

BMM file

A Block RAM Memory Map (BMM) file is a text file that has syntactic descriptions of how individual block RAMs constitute a contiguous logical data space. Data2MEM tool uses BMM files to direct the translation of data into the proper initialization form. Since a BMM file is a text file, it is directly editable.

Due to the four block RAM memory controllers added to the system, the BMM file must be modified to create a new contiguous memory space joining the four address spaces of the controllers.

Chapter 7

Test Software development

In this chapter a detailled description of the software development for testing purposes is given. A different version of the control software is needed due to some limitations in the actual hardware and software.

7.1 Limitations and adopted solution

The existing limitations are the following:

- The Ethernet MAC on the custom board is a class 2 type one, which means that it is not possible to use this MAC for Profinet communications. The Profinet network present in the Kart uses MACs of type 3, which are not backwards compatible.
- Requirement of an external software Profinet Stack

Due to this limitations, the adopted solution was to communicate the old Siemens IPC with the Embedded System on the FPGA. The Siemens IPC has the task of procesing the information of the sensors, communicate with the FPGA and controlling the converters. The Embedded System on the FPGA has the task of exchange the information with the IPC and use this to do the torque vectoring and traction control calculations.

7.2 Industrial PC software

In this section, the changes made to the software, that runs in the industrial PC, are explained. These changes affect to the stored telemetry data and the communication with the developed embedded system via the serial interface. The changes were made in a way to have the minimal impact on the performance of the system because of its real time behaviour.

7.2.1 RMOS3

RMOS3 is a real time operating system with multitasking functionality. Along with the ability of real-time, short reaction times determinism is an important feature included. Thanks to that the reaction times are guaranteed on a short time interval.

Robust operating systems also offer strong performance in emergency situations. RMOS3 is optimized for use in embedded applications. In harsh environmental conditions can be used in place of hard drives or memory CompactFlash Memory Cards. Also RMOS3 is suitable for active and inactive service. RMOS3 supports the C and C++ programming languages.

7.2.2 UART communication

The UART is the only free and remaining interface available on the Siemens computer usable for the communication purposes. To control the UART on the Siemens IPC, the BYT is available. The BYT driver serves for simultaneous management of multiple, byte-oriented distributed I/O devices (parallel driver). Byte-oriented units include, for example, terminals and printers. it is possible, for example, to defined custom control characters for terminals. The driver can be customized to suit the requirements of all standard controllers. The BYT driver can be operated in unrestricted half duplex mode, i.e. no simultaneous transmission and receive operations, as well as in restricted duplex mode. The terminal and transparent modes differ in terms of the handling of terminal/protocol control characters required for calls of specific read functions of the BYT driver [29].

RmIO function

In order to control the UART driver, the RMOS3 operating system provides the function "'RmIO" in the API. This function is able to control up to 255 IO drivers in the system. A typical call of this function looks like [30]:

```
int _FIXED _FAR RmIO(uint Function,
                     uint DeviceID,
                     uint UnitID,
                     uint FlagID,
                     uint FlagMask,
                     RmIOStatusStruct *pState,
                     void *pParam);
```

Table 7.1 describes the parameters of the function abovementioned.

Parameter	Description
Function	Type of IO function (see 7.2.2)
DeviceID	ID of the Driver
UnitID	ID of the physical device
FlagID	ID of the event flag group
FlagMask	Mask of the flag
pState	Pointer to a 8 byte status field
pParam	Pointer to I/O parameters (see 7.2.2)

Table 7.1: Parameters to the RmIO Siemens function

Send information

To send the information by using the onboard UART of the Siemens IPC, the previously presented parameters must be set as following.

- The function parameter must be set to **BYT_WRITE**. This mode allows the UART to send information.
- The DeviceID parameter must be set to **stddata.stdout_dev**, where stddata is a struct of type STDSTRUCT, which holds the important information concerning the input/output devices. The stdin_dev references the standard output of the system, in this case, the UART.
- The UnitID parameter must be set to **stddata.stdout_unit**.
- The fields of the parameters block must be set as following:
  - string: reference to the first element on the sending buffer
  - strlen: size of the sending buffer

Receive information

To receive the information by using the onboard UART of the Siemens IPC, the previously presented parameters must be set as following.

- The function parameter must be set to **BYT_POLL_XBUF**. This mode allows the user to poll the receiving buffer of the UART to retreive information.
- The DeviceID parameter must be set to **stddata.stdin_dev**, where stddata is a struct of type STDSTRUCT, which holds the important information concerning the input/output devices. The stdin_dev references the standard input of the system, in this case, the UART.
- The UnitID parameter must be set to **stddata.stdin_unit**.
- The fields of the parameters block must be set as following:
  - buffer: reference to the first element on the sending buffer
  - timlen: size of the sending buffer

7.2.3 IO configuration

By default, the software for the Siemens IPC was developed without serial communication capabilities, which means that the input direction of the standard IO of the system is blocked. With the default configuration of the software is only possible to send information to the outside. This must be changed in order to have a bidirectional communication, so further changes in the software were needed. In the entry point of the main task, the following code had to be added:

```
fdureopen((entry.ide & 0xFFFF), //Device
          entry.id,     //Unit
          "r",          //Mode
          stdin);       //Stream
```

The C standard API's function "fdureopen" is used to redirect a stream to a driver. As showed in the previous extract of code, the input stream of the system is redirected to the UART driver in read mode, which allows the system to read the input buffer of the UART.

7.2.4 UART timing

The UART included in the Siemens PC has a minimum access time of around 10 to 20 ms to access his buffers through the RmIO function. If the RmIO function is accessed repeatedly without a time interval of 10 to 20 ms between calls, the system could be blocked. To secure that restriction, a delay is added before a the software has to send data and also when the software tries to access the input buffer to read the incoming data.

7.2.5 Synchronization switch

The communication between the IPC and the Embedded System is started by the Siemens IPC meanwhile the Software of the Embedded System waits for the first message. To start the communication, the "max speed limit" switch was chosen, so activating this switch controls the start and the stop of the communication. It is important to stop the communication at the end of the run in order to read all the telemetry data stored during the test run of the kart.

7.2.6 Telemetry information

In order to make a later analysis of the telemetry information, this information must be stored and later retrieved. To store the information, a vector of floating point data is created with a maximum length of 400000, which means that at 2000 samples per second and with six parameters to store, it can be recorded 33,3 seconds of data, enough to cover a complete run of the car. The information is organized in a csv file type format and it is stored in a log file, which is saved by the hyperterminal application used to receive the data. Later, this data can be used to draw charts using Matlab or another similar tool.

7.3 Test Software

The changes on the test software runnning on the embedded system are made to keep the communication with the Siemens IPC and exchange the necesary information to control the motors.

7.3.1 Multi-threading

Due to certain asynchronous communication tasks, a Multi-threading enviroment is required. The software is structured in two threads, the main thread executes the control loop and shares

Item	Type	Description
slip_l	float (input)	Left side slip
slip_r	float (input)	Right side slip
speed	float (input)	Global speed of the kart
steer_ang	unsigned (input)	Steering wheel angle
tcs_act	unsigned (input)	TCS on/off
torque_fact	float (input)	Torque Vectoring factor
gas	float (input)	Gas pedal percentage
torque_left	float (output)	Torque to the left side motor
torque_right	float (output)	Torque to the right side motor
tcs_left_fact	float (output)	TCS torque correction to the left side motor
tcs_right_fact	float (output)	TCS torque correction to the right side motor

Table 7.2: Test software shared data

a memory zone with a secondary thread in charge of be receiving and sending information. The shared information between the two threads is resumed in Table 7.2.

7.3.2 Communication protocol

In order to exchange the important data to do the calculations in the Embedded System and send back to the Siemens IPC, which is controlling the motors, a custom communication protocol is required.

Protocol

The protocol is designed due to limitations of the UART implemented in the Embedded System. This UART has a small buffer, which can only contain 16 Bytes. Because the exchanged data is larger than 16 Bytes, the information has been splitted in two parts. An intermediate ACK is needed for synchronization purposes. Also due to limitations in the configuration of the UART in the Siemens IPC, the information cannot be transmitted in binary format, so the information is sent in ASCII code an converted in the destination system.

1. The Siemens IPC sends the first part of the sending buffer. Meanwhile the software in the Embedded System waits to receive the whole buffer.

2. Once the first part is received, the Embedded System sends an ACK to the Siemens IPC.

3. When the IPC receives the ACK, he sends the second part of the buffer.

4. The Embedded System receives the whole buffer and sends to the IPC the data produced by the torque vectoring and traction control calculations.

Figure 7.1 shows the data exchange protocol created to comunicate both computers, the Siemens IPC and the Embedded System.

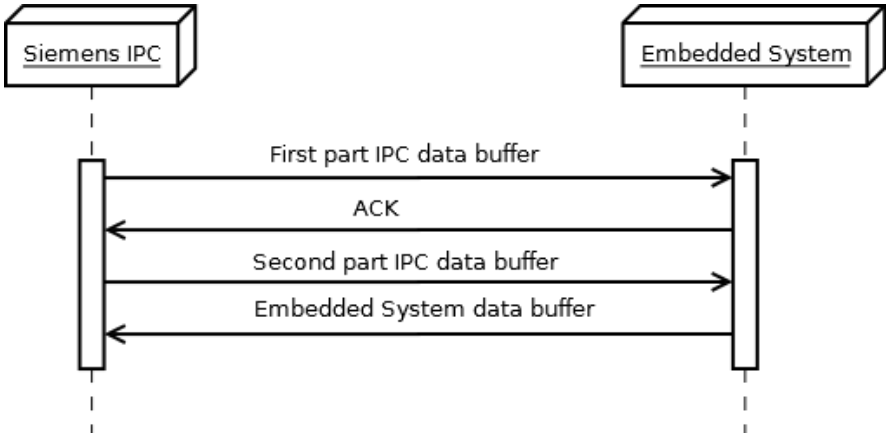

Figure 7.1: Communication protocol

Table 7.3 resumes the exchanged data.

Item	Size	Format	Range
slip_l	5 Bytes	x.xxx	0.100 to 1.000
slip_r	5 Bytes	x.xxx	0.100 to 1.000
speed	6 Bytes	xx.xxx	0.000 to 38.889
steer_ang	3 Bytes	xxx	0 to 100
torque_fact	3 Bytes	x.x	0.0 to 1.0
tcs_act	1 Byte	x	0 to 1
gas	7 Bytes	xxx.xxx	0.000 to 100.000
torque_left	7 Bytes	xxx.xxx	0.000 to 100.000
torque_right	7 Bytes	xxx.xxx	0.000 to 100.000
tcs_left_fact	5 Bytes	x.xxx	0.100 to 1.000
tcs_right_fact	5 Bytes	x.xxx	0.100 to 1.000

Table 7.3: Shared data format

Chapter 8

Test and Validation

In this chapter a detailled description of the testing stage of the development is given for both hardware and software parts. This stage was executed in parallel with the development process to detect errors easily and at earlier stages of the development. All the tests were manually executed except the VHDL testbenchs and the testing environment is described in each section.

8.1 Hardware

The hardaware is the biggest part of this Master Thesis and each developed hardware component had to be tested in a proper way to be fault proven.

8.1.1 Simulator

The simulator is an important part of the hardware test and it is the very first tool, that the hardware developer has to test the hardware modules before implementing them in the FPGA. By using the simulator, common errors as well not common are easier to find and eliminate. The signal waves in the simulator shows the behaviour of the hardware and the developer can see if this behaviour matchs with the expected behaviour. In this Master Thesis two types of simulations have been done unsing for that the Simulator iSim, included in the Xilinx ISE suite.

Behavioral simulations

The behavioral simulations are the simpler simulations. As his name says, they are used to test the proper behaviour of the developed hardware module. In this testing stage, concept and programming errors in the VHDL code can be found. If the simulation shows the expected behaviour, it is a good point but it does not mean that the developed hardware is going to work in the FPGA because this simulation is executed even before the synthesis of the hardaware, which means that the delay of the signals, clock skew, and other important parameters are not taken into count. These simulations can be performed manually but there is an important tool to make them automatic. This tool is the called VHDL testbench. A testbench is a normal VHDL file used for testing (it is not possible to do the synthesis stage) purposes, where stimulus signals can be generated and connected to the VHDL modules, which are intended to be tested.

Post place and route simulations

Post place and route simulations are generated by the Xilinx ISE right after the place and route stage, which is the last one before generating the final bitstream. These type of simulations includes, additional to the behavioral simulations features, the delay of the signals in the model, so they are really usefull to test if the timing constraints are met, the delays, the clock skew and the hold to set up. When a post place and route simulation shows the desired behaviour and delays, the developed hardware is going to work on the FPGA.

8.1.2 Testing platform

Apart from the simulator, all the developed peripherals and coprocessors were tested on the board. In the case of the peripherals because they control physical devices on the board and it is important to test if they work properly. In the case of the coprocessor just to test if they produce the expected calculations. For both, all the functionality included in the drivers were tested.

For testing purposes, a small Embedded platform was developed following the same design as the final system but only with the strictly necesary components to test the components. The selected software was the Standalone platform instead of the Xilkernel. Standalone does not need configuration and runs with lower memory requirements.

8.2 Software

After the hardware was developed, the software was developed and had to be tested. This software part includes both drivers and control routines.

8.2.1 Testing platform

The testing platform was the final Embedded System with a Xilkernel operating system.

8.2.2 Modules

Drivers

The software drivers of each peripheral and coprocessor were tested at the same time of the hardware. Very simple software routines were writen in order to test the features of the hardware. In some cases, the interaction of the user is required to test the hardware and software. For example, when the analog and digital inputs were tested, the user had to select the different positions of each switch and button to test the digital inputs. In the case of the analog inputs, the interaction with the different analog sensors (accelerator, brake and other sensors) was required.

Control

The control routine was tested manually using the onboard debugger and the standard input/output through an Hyperterminal. All the important parameters were displayed in the Hyperterminal at each control loop round.

Communications

The communication software was tested using the abovementioned Hyperterminal as well a secondary communication module in the case of the CAN bus VHDL module.

8.2.3 Integration

The completelly integrated system, both hardware and software, was tested with all the sensor connected to the board. It was only necessary to prove that the parameters and values in the software were the expected.

8.3 System

8.3.1 Traction control

The traction control coprocessor was tested mainly in the garage. Two ways of testing the traction control were performed:

1. Simulation of the slip values: A software routine was developed to simulate the slip values of the wheels. The slip telemetry data was recorded in a previous test in June 2013, when the original software traction control was in testing stage. The reaction of the hardware coprocessor was stored and used to create a chart to compare his behaviour with the software version.

2. In the car: The wheel slip values in real time were transmited to the coprocessor and the calculated correction factor was used to reduce the amount of torque in each moment.

Validation Charts

Figure 8.1 shows the behaviour of the Traction Control core. The slip of the wheels was simulated in a simular way to real values from a previous test in June 2013.

8.3.2 Torque vectoring

The torque vectoring coprocessor was tested only in the garage. In order to test the torque vectoring is needed a wide testing place to drive the kart in constant curve trajectories. Battery autonomy is also an important point. Because the autonomy is not more than 100 meters at full power and the wide testing place required to move the kart from the South campus to the East campus of the Univeristy, the torque vectoring behaviour was tested with a software simulation routine. Because this torque vectoring is calculated with a mathematical formula, it is easy to compare his behaviour with the expected values.

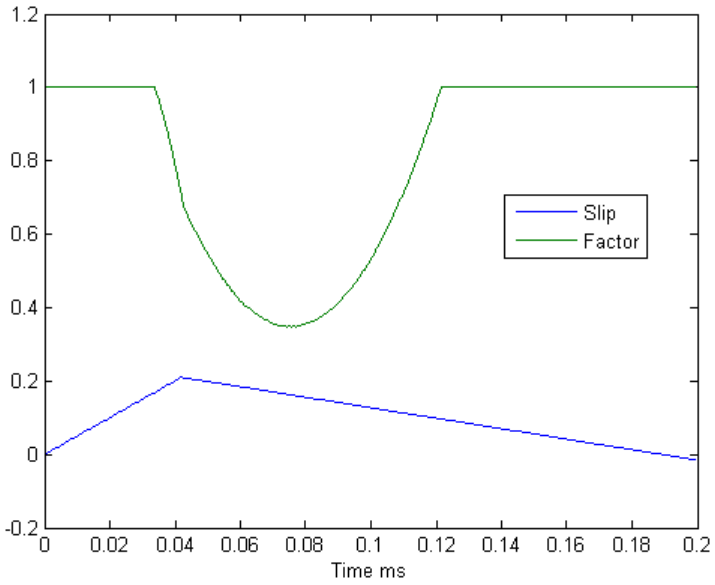

Figure 8.1: Traction Control simulation

Validation Charts

Figure 8.2 shows the behaviour of the Torque Vectoring core. It was simulated from 0 to 40 m/s (the maximum speed of the kart) and for different steering angles.

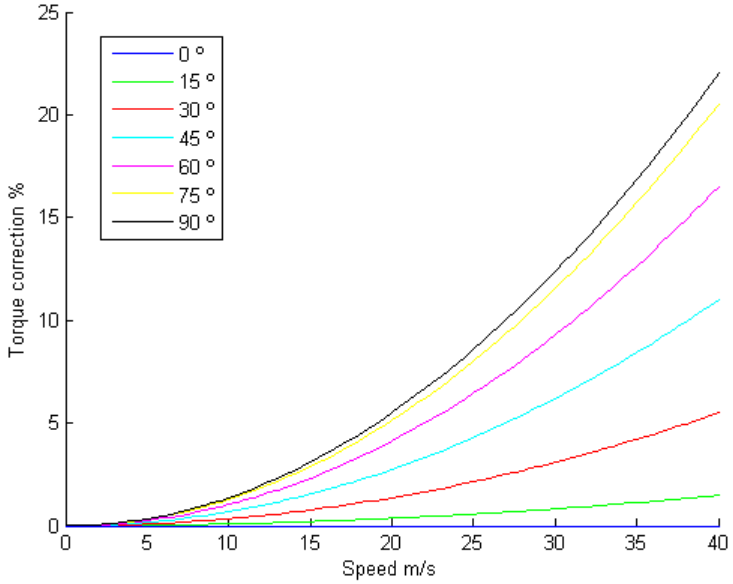

Figure 8.2: Torque Vectoring simulation

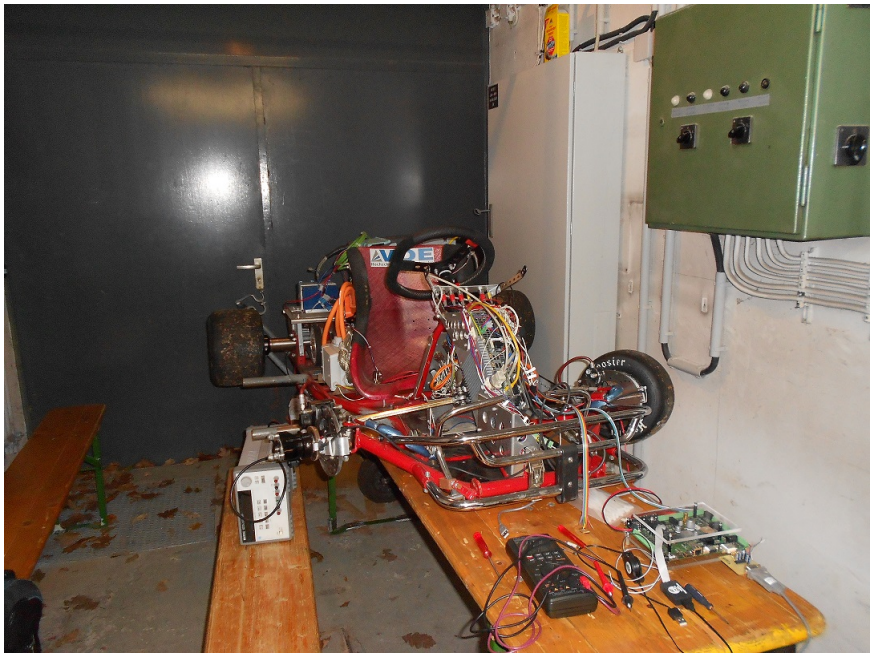

Figure 8.3: Garage testing environment

8.3.3 Testing environment

The system was tested in two different environments, in the garage and in the street.

Garage

In the garage the majority of the test (static tests) were performed. The power supply in this case was connected direct to the power network and set to a stable and constant 24 Volts with a maximum meassured current of 4,5 Amps. Figure 8.3 shows the testing environment in the garage, where the first tests in the kart were made.

Street

At the street the dynamical tests were performed. Due to evident limitations (weather, testing place and volunteers) only a few test journeys could be done but important telemetry data for posterior analysis was colected.

Figure 8.4 shows the set up of the kart for testing purposes. The new custom board is fixed in the chassis and connected via serial port with the Siemens IPC.

Figure 8.5 shows the testing environment in the street, where the traction control was tested on the new custom board.

Figure 8.4: Street testing set up

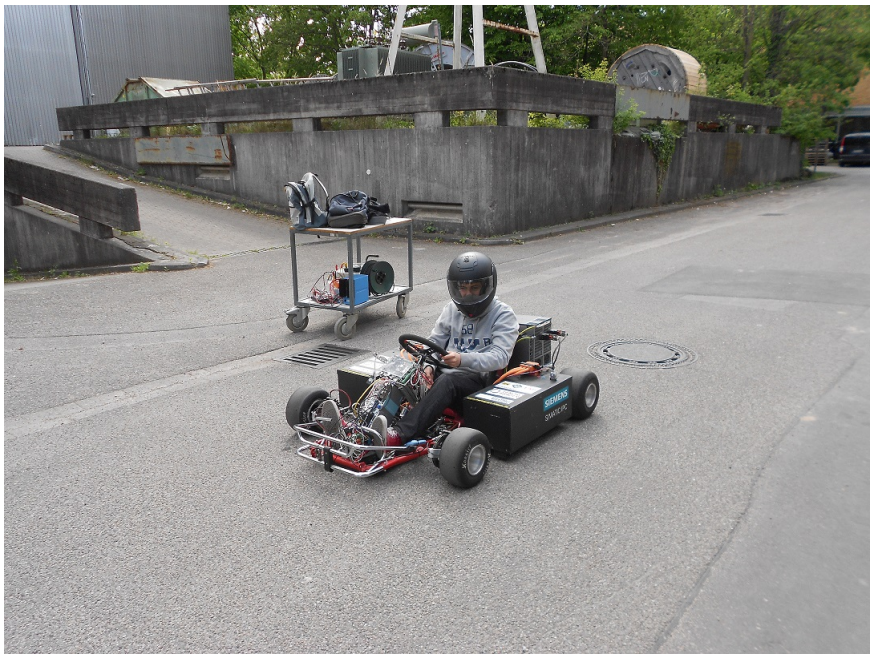

Figure 8.5: Street testing environment

Chapter 9

Conclusions

In this chapter, the technical and personal valoration, as well as the possible further work are presented.

9.1 Technical valoration

- Traction Control: The Traction Control core behaves in the same way as the software version and reduces load in the software, so a better overall performance is reached.
- Torque Vectoring: The Torque Vectoring core reduces also the load of the software and helps to reach a higher performance.
- I/O and COM Peripherals: The I/O and COM peripherals offer a standard register interface to connect or adapt them to any existing type of bus and embedded system.
- Set the basis for further Bachelor/Master Thesis: There are many more things to implement in this Kart and this Thesis is only the basis and the beginning for upcomming Thesis.
- Performance: The new system has a better performance than the older one, which means that additional features can be added in the source code without handicapping the base performance thanks to this given performance margin.

9.2 Personal valoration

This Master Thesis has been the second contact (first was the Bachelor Thesis) with the Automotive Industry. During this Master Thesis, I have improved my knowledge on certain fields, that I have treated in the past, as well I have learnt about new technologies and interesant topics:

- Xilinx Tools: I had 3 years of experience in the Xilinx Embedded Development Kit, thanks to the previous work on my Bachellor Thesis and further personal interest. In this Master Thesis, I have improved my knowledge about this tools and also I have learnt how different is the design of an Embedded System for a, unknown to this moment, different FPGA target Spartan 6 instead of Spartan 3.

- New operating systems: In the development of the software, I had to learn the use of two different operating systems, both completely new for me. In the case of the Xilkernel, how it works, how to configurate the kernel to get a custom and small system and of course how to develop software for the Xilkernel using the available features of the system. In the case of the RMOS3 of Siemens, I had to understand how the operating system works as well as the control software routines. These two operating system are widely used in the industry, so this can be considered as an starting point to learn these technologies.

- Electronics: I had only the basic knowledge about electronics but in this Master Thesis I have learnt a lot of new concepts, from the basic knowledge about the physical chips on the board and their functionaly to the electric motors, converters and industrial electronics.

- Mechanical engineering: Because the main basis of this Thesis is the VDE Kart, some implemented functionalities, such as the traction control and torque vectoring, requiered the understanding of the basic mechanical priciples in order to know, how they work and translate this knowledge to the electronical and software design.

- Communications: I have improved my knowledge about the CAN bus but I have also learnt about Profinet protocol standard.

- Embedded Systems: The system of this Thesis is really big compared with other embedded systems, that I have designed in the past, so it gave me the opportunity to learn how to design and integrate the hardware components aiming the best possible performance but also trying to keep the design compact.

- Simulation and test of hardware: I have learnt how to write VHDL testbenchs for the automatization of the hardware tests.

9.3 Further work

- Increase the frequency of the system: It is unknown, which requirements are needed for the future. The System has been implemented thinking in that and it does not run at his maximum performance. The actual frequency of the system is set to 50 MHz but this design could allow up to 120 MHz. In the future, more components are going to be added, thus the performance at 50MHz could be affected. The easiest way to increase the global performance of the system is to increase the base frequency.

- Multi-core system: Other way to increase the performance of the system, if needed, is to add more processors to the system and built a multi core version. Multi core versions could run at lower frequencies, which could represent a better power consumption. Due to the flexibility of the FPGAs and the remaining space in the selected FPGA this extension is possible and not a complicated task thanks to the Xilinx development tools.

- Wireless Live Telemetry using GSM: Telemetry is an important aspect in motor racing and if it can be accessed live, it could allow the engineers to have a exact view point of the situation at each moment and tell the driver if some changes are needed to reach

a better performance. The term "'live" means that the transmission of the information must be performed wireless. The GSM option has been considered and the custom board has a GSM transceiver, so making a few changes to the hardware system, it is possible to send the information via GSM by using a simple UART.

- Display: It is also possible to add a display, where the driver can see in real time the most important parameters of the kart, such as speed, the charge of the batteries, temperature of the motors and more. The custom board has an HDMI connector, so a HDL HDMI controller could be added to the system to control the display.

- Launch control: The launch control is a system similar to the traction control but they are by concept used for different purposes. Meanwhile the traction control is dynamic, the launch control is static, which means that his behaviour is programed before the start. Both can be used to achieve better traction at the start but the launch control will not work appart from the start because it is only designed for this purpose. Despite that, the launch control has a better performance at the start compared to the traction control.

- Regenerative braking (Active motor braking): The kart has only disk brakes on the front wheels, which means that the braking power is focused on the front wheels. The weight distribution of the kart is approximately 25/75, which means that the majority of the weight is on the rear axle. For braking dynamics it is really bad because the front wheels have less grip and it is really easy to block them. In order to add more braking power and to balance the brakes, it is possible to brake with the electric motors. In addition, if the system allows regenerative braking, the energy generated by the motors can be used to charge the batteries.

- Delegate more tasks to hardware: Certain tasks could be performed by the hardware in the future, releasing load to the processor.

Appendix A

Development tools

Doxygen

Doxygen is the de facto standard tool for generating documentation from annotated C++ sources, but it also supports other popular programming languages such as C, Objective-C, C#, PHP, Java, Python, IDL (Corba, Microsoft, and UNO/OpenOffice flavors), Fortran, VHDL, Tcl, and to some extent D.

Doxygen can help you in three ways:

1. It can generate an on-line documentation browser (in HTML) and/or an off-line reference manual (in LaTeX) from a set of documented source files. There is also support for generating output in RTF (MS-Word), PostScript, hyperlinked PDF, compressed HTML, and Unix man pages. The documentation is extracted directly from the sources, which makes it much easier to keep the documentation consistent with the source code.

2. You can configure doxygen to extract the code structure from undocumented source files. This is very useful to quickly find your way in large source distributions. Doxygen can also visualize the relations between the various elements by means of include dependency graphs, inheritance diagrams, and collaboration diagrams, which are all generated automatically.

3. You can also use doxygen for creating normal documentation (as I did for the doxygen user manual and web-site).

Please see http://www.stack.nl/~dimitri/doxygen/ to have a more detailed look about Doxygen.

Dia

Dia is roughly inspired by the commercial Windows program 'Visio,' though more geared towards informal diagrams for casual use. It can be used to draw many different kinds of diagrams. It currently has special objects to help draw entity relationship diagrams, UML diagrams, flowcharts, network diagrams, and many other diagrams. It is also possible to add

support for new shapes by writing simple XML files, using a subset of SVG to draw the shape.

It can load and save diagrams to a custom XML format (gzipped by default, to save space), can export diagrams to a number of formats, including EPS, SVG, XFIG, WMF and PNG, and can print diagrams (including ones that span multiple pages).

Please see https://wiki.gnome.org/Apps/Dia to have a more detailed look about Dia.

Xilinx Project Navigator

Project Navigator integrates the tools you need and gets your design process started quicker in an easy-to-use graphical interface. All Editions of the ISE Design Suite include the ISE Project Navigator which provides project and design source management, easy access to running all necessary steps in the ISE design flow, and access to viewing and analyzing design results. In addition, designers have access to intuitive Architecture Wizards and IP catalog, language templates, graphical tools to assist with I/O planning, constraint entry, and design analysis, ISim HDL simulator, error navigation to Answer Records on the Web, and much more.

Please see http://www.xilinx.com/tools/projnav.htm to have a more detailed look about Project Navigator.

ISim

ISim provides a complete, full-featured HDL simulator integrated within ISE. HDL simulation now can be an even more fundamental step within your design flow with the tight integration of the ISim within your design environment.

Please see http://www.xilinx.com/tools/isim.htm to have a more detailed look about ISim.

EDK

The Embedded Development Kit (EDK) is an integrated development environment for designing embedded processing systems. This pre-configured kit includes Xilinx Platform Studio and the Software Development kit, as well as all the documentation and IP that you require for designing Xilinx Platform FPGAs with embedded PowerPC hard processor cores and/or MicroBlaze soft processor cores.

Please see http://www.xilinx.com/tools/platform.htm to have a more detailed look about Xilinx EDK.

SDK

The Software Development Kit (SDK) is the Xilinx Integrated Design Environment for creating embedded applications on any of Xilinx' award winning microprocessors from Zynq-7000 All Programmable SoCs, to the industry-leading MicroBlaze. SDK is the first application IDE to deliver true homogenous and heterogenous multi-processor design and debug.

Please see http://www.xilinx.com/tools/sdk.htm to have a more detailed look about Xilinx EDK.

Bibliography

[1] VDE group Karlsruhe website: http://vde-karlsruhe.de/das-projekt

[2] Francisco Caicedo Montalvo: Entwurf und Implementierung einer Antriebsschlupfregelung. Bachelor's Thesis, Karlsruher Institut für Technologie, September 2012.

[3] Philipp Bäuerle: Das VDE E-Performance Kart. Bachelor's Thesis, Karlsruher Institut für Technologie, October 2011.

[4] Malcom Burgess: Torque Vectoring http://www.vehicledynamicsinternational.com/downloads/VDI_Lotus_Vector.pdf

[5] Torque vectoring technology: http://torque-vectoring.belisso.com/

[6] Wisniewski, Remigiusz (2009). Synthesis of compositional microprogram control units for programmable devices. University of Zielona Gora. ISBN 978-83-7481-293-1 http://zbc.uz.zgora.pl/Content/27955/Remigiusz_Wisniewski_Synthesis_of_CMCUs_for_Programmable_Devices.pdf

[7] FPGA Architecture for the Challenge. Electrical and Computer Engineering Department, University of Toronto http://www.eecg.toronto.edu/~vaughn/challenge/fpga_arch.html

[8] Dennis M. Ritchie. "The History of the C Programming Language". Retrieved February 26, 2014. http://cm.bell-labs.com/who/dmr/chist.html

[9] "Programming Language Popularity". 2009. Retrieved 16 January 2009. http://www.langpop.com/

[10] CAN wiki: http://www.can-wiki.info/doku.php

[11] CAN bus standard document: http://www.bosch-semiconductors.de/media/pdf_1/canliteratur/can2spec.pdf

[12] Xilinx datasheet 160: http://www.xilinx.com/support/documentation/data_sheets/ds160.pdf

[13] Xilinx Microblaze FAQ: http://www.xilinx.com/products/design_resources/proc_central/microblaze_faq.pdf

[14] Xilinx Microblaze website: http://www.xilinx.com/tools/microblaze.htm

[15] Xilinx OS library: http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/oslib_rm.pdf

[16] Local Memory Bus (LMB) v1.0: http://www.xilinx.com/support/documentation/ip_documentation/lmb.pdf

[17] IP Processor LMB BRAM Interface Controller http://www.xilinx.com/support/documentation/ip_documentation/lmb_bram_if_cntlr/v3_00_b/lmb_bram_if_cntlr.pdf

[18] LogiCORE IP Clock Generator: http://www.xilinx.com/support/documentation/ip_documentation/clock_generator.pdf

[19] LogiCORE IP XPS Timer/Counter: http://www.xilinx.com/support/documentation/ip_documentation/xps_timer.pdf

[20] LogiCORE IP XPS Interrupt Controller: http://www.xilinx.com/support/documentation/ip_documentation/xps_intc.pdf

[21] XPS General Purpose Input/Output (GPIO): http://www.xilinx.com/support/documentation/ip_documentation/xps_gpio.pdf

[22] LogiCORE IP XPS UART Lite: http://www.xilinx.com/support/documentation/ip_documentation/xps_uartlite/v1_02_a/xps_uartlite.pdf

[23] LogiCORE IP Processor Local Bus: http://www.xilinx.com/support/documentation/ip_documentation/plb_v46.pdf

[24] Opencores Wishbone bus standard: http://cdn.opencores.org/downloads/wbspec_b3.pdf

[25] PLB to Wishbone core: http://opencores.org/project,plbv46_to_wb_bridge

[26] Xilinx Platform Specification Format: http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_2/psf_rm.pdf

[27] Xilinx Floating point operator: http://www.xilinx.com/support/documentation/ip_documentation/floating_point_ds335.pdf

[28] Xilinx CORDIC IP core: http://www.xilinx.com/support/documentation/ip_documentation/cordic/v5_0/ds858_cordic.pdf

[29] RMOS3 reference manual part 2: http://cache.automation.siemens.com/dnl/jU/jUzODg4OQAA_84119572_HB/RMOS3_Reference_Manual_Part2.pdf

[30] RMOS3 reference manual part 3: http://cache.automation.siemens.com/dnl/zQ/zQ5Mjc3AAAA_84125183_HB/RMOS3_Reference_Manual_Part3.pdf

[31] Open Cores Community: http://opencores.org/

GNU Free Documentation License

Version 1.3, 3 November 2008

<http://fsf.org/>

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.
This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.
We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.
A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.
A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the

Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "**publisher**" means any person or entity that distributes copies of the Document to the public.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this

License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.
You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.
If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.
If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.
It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.
You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.
You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.
The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.
The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.
In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.
You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.
If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.
If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.
However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.
Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.
Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/. Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.
"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.
"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.
An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.
The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with ... Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.
If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.