An embedded controller for an active magnetic bearing and drive electronic system

A dissertation presented to

The School of Electrical, Electronic and Computer Engineering

North-West University

In partial fulfilment of the requirements for the degree

Magister Ingeneriae

in Electrical and Electronic Engineering

by

Rikus le Roux

Supervisor: Prof. G. van Schoor

Assistant-supervisor: Mr. J. Jansen van Rensburg

Project manager: Dr. E.O. Ranft

November 2009

Potchefstroom Campus

Declaration

I hereby declare that all the material incorporated in this thesis is my own original unaided work except where specific reference is made by name or in the form of a numbered reference. The word herein has not been submitted for a degree at another university.

Signed:	______________________	______________________
	Rikus le Roux	Date

Project information

Personal contact details:

Name	:	Mr. R.R le Roux
Organization	:	North-West University, School for Electrical and Electronic Engineering
Address	:	Private bag X6001, Potchefstroom 2520
Telephone	:	(018) 299 4298
Fax	:	(018) 299 1977
E-Mail	:	13077643@nwu.ac.za

Supervisor contact details:

Name	:	Prof. G. van Schoor
Organization	:	North-West University, School for Electrical and Electronic Engineering
Address	:	Private bag X6001, Potchefstroom 2520
Telephone	:	(018) 299 1962
Fax	:	(018) 299 1977
E-Mail	:	12134457@nwu.ac.za

Project manager:

Name	:	Dr. E.O Ranft
Organization	:	North-West University, School for Electrical and Electronic Engineering
Address	:	Private bag X6001, Potchefstroom 2520
Telephone	:	(018) 299 1975
Fax	:	(018) 299 1977
E-Mail	:	12133094@nwu.ac.za

Assistant-supervisor:

Name	:	Mr. J. Jansen van Rensburg
Organization	:	North-West University, School for Electrical and Electronic Engineering
Address	:	Private bag X6001, Potchefstroom 2520
Telephone	:	(018) 299 4298
Fax	:	(018) 299 1977
E-Mail	:	12576786@nwu.ac.za

Summary

The North-West University is currently conducting research in the area of active magnetic bearings (AMBs). The aim of this research is to establish a foundation for the development of AMB systems to be used in industrial applications. These systems should be reliable, effective and economical. The main research objective for this project is to further develop key technologies in order to realize an economical, reliable high-speed AMB drive system to be used in high-speed machinery. The proposed system is the AMB and drive electronic system (ADES), which is a digital control system for controlling AMBs in an industrial environment. The development of the ADES was a group effort. The focus of this dissertation was on selecting and implementing a suitable controller to be used in the ADES. The specification for the ADES was obtained from an industrial high-speed helium blower system.

Selecting the controller was done by concurrently evaluating the conceptual main controller architectures and proposed system architectures. The system architecture is based on an industrial form factor, called compact peripheral component interconnect (PCI), or cPCI, which is an industrial version of PCI. The architectures were evaluated by performing trade-off studies and by weighing each architecture against a decision matrix, which weighs the architectures according to robustness, efficiency, cost, risk, reliability, flexibility and expandability. The selected system architecture includes a single board computer (SBC) with two PCI mezzanine cards (PMCs); a Virtex®-5 field programmable gate array (FPGA) based PMC module, for scheduling real-time tasks, and a Profibus PMC module, which will be used in future iterations of this project to interface the ADES with a programmable logic controller (PLC).

The specified functions were designed, verified and implemented on the selected controller. The digital control was implemented on the FPGA-embedded PowerPC whereas the communication and filters were implemented on the FPGA. The sensitivity analysis placed the system into zone C, which implies a system normally considered unsatisfactory for long-term continuous operation. The system may operate in this condition for a limited period, until a suitable opportunity arises for remedial action. It was also determined that the system is stable for a step-input added to the reference position. Due to the stability of the control, it was possible to suspend the rotor at its designed rating of 19,000 r/min, but due to the high sensitivity rating, prolonged operation at this speed is not recommended.

The selected architecture is versatile and powerful. The FPGA as a co-processor can be used to alleviate the load on the PowerPC, if additional features are required and not enough clock cycles are left on the PowerPC to implement them. The solution is compact, powerful and robust. These features, together with the industrial-based architecture of the system, make the ADES a suitable controller for controlling AMBs in an industrial system.

Acknowledgements

Great thanks go out to M-Tech Industrial, THRIP and the North-West University. Thank you for funding this project and for the opportunity to further my studies.

I would also like to thank the following people for their contribution, support, knowledge and expertise throughout this project:

- Prof. George van Schoor, my supervisor, for his guidance, support and advice
- Mr. Jacques Jansen van Rensburg, my assistant-supervisor and friend, for his insight, advice and friendship
- Dr. Eugén Ranft, project manager, for managing and driving the project
- Mr. Cobus Potgieter, our industrial partner and consultant, who provided technical expertise, guidance, advice and VHDL training
- Dr. André Niemann, for his invaluable insights, input and for helping to design the filters
- Mr. Roelof Burger, for his design of the user interface
- Mr. Morné Neser, for his input and help regarding object orientated embedded C, as well as helping to solve general embedded C issues
- Ms. Elsebi Gadney, for her friendship over the last couple of years. Without her, this project would have been a whole lot less entertaining.
- My family, for their unwavering love, support, encouragement and prayer
- My girlfriend, Nicolene Swart, for her love, patience and support
- The McTronX research group and in the specific André, Elna, Elsebi, Gordon, and Kristoff, for their friendship and late night coffee breaks

I would like to thank everyone who designed, built or implemented components cardinal to the success of the project. Without your hard work, there would not be a system to control or a rotor to suspend.

Above all, I would like to thank our Heavenly Father, for this amazing opportunity He bestowed upon me. “I can do everything through Him who gives me strength.”

"When you go through deep water and great trouble, I will be with you.
When you go through rivers of difficulty, you will not drown!
When you walk through the fire of oppression, you will not be burned up; the flames will not consume you.
I will be with you…"

Isaiah 43:2 (LB)

*"There is no extra magic inside the FPGA………
Just the usual magic."*

Göran Bilski (Xilinx Employee)

Table of contents

List of figures

List of tables

List of abbreviations and acronyms

(A)GCR	-	(Advanced) Gas Cooled Reactor
ADC	-	Analogue to Digital Conversion
ADC	-	Analogue to Digital Converter
ADE(S)	-	AMB and Drive Electronic (System)
ADEU	-	AMB and Drive Electronic Unit
ADI	-	Analog Devices, Inc.
AIM alliance	-	Apple, IBM and Motorola alliance
ALU	-	Arithmetic Devices and Arithmetic Logic Units
AMB	-	Active Magnetic Bearing
ANSI	-	American National Standards Institute
API	-	Application Programming Interfaces
API	-	Application Programming Interfaces
APU	-	Auxiliary Processor Unit
ARM	-	Acorn/Advanced RISC Machine
B/FLOP	-	Byte per Floating-point OPerataion
BDTI	-	Berkeley design technology, Inc
BPR	-	Bandwidth-to-Processing Ratio
bps	-	bits per second
BRAM	-	Block RAM
CAD	-	Computer Aided Design
CAD	-	Computer-Aided Design
CAN	-	Control Access Network
CAT	-	Computed Axial Tomography
CISC	-	Complex Instruction Set Computer
CLB	-	Complex Logic Block
CMOS	-	Complementary Metal-Oxide Semiconductor
COTS	-	Commercial Off-The-Shelf
cPCI	-	compact PCI
CPLD	-	Complex Programmable Logic Device
CPU	-	Central Processing Unit
DAC	-	Digital to Analogue Converter
dB	-	Decibel
DC	-	Direct Current
DCM	-	Digital Clock Management
DCM	-	Digital Clock Management
DDR SDRAM	-	Double Data Rate Synchronous Dynamic Random

		Access Memory
DIMM	-	Dual In-line Memory Module
DOF	-	Degree Of Freedom
DPR	-	Dual Port Random Access Memory
DSP	-	Digital Signal Processor
EDK	-	Engineering Design Kit
EEMBC	-	EDN Embedded Microprocessor Benchmark Consortium
FCB	-	Fabric Coprocessor Bus
FFT	-	Fast Fourier Transform
FIR	-	Finite Impulse Response
FLOPS	-	FLoating-point Operations Per Second
FPGA	-	Field Programmable Gate Array
FPR	-	Floating-Point Register
FPU	-	Floating-point Unit
FSM	-	Finite State Machine
GB	-	Giga Byte
GFLOPS	-	Giga Floating-point Operations Per Second
GHz	-	Giga Hertz
GMACS	-	Giga Multiplications and Accumulation Cycles per Second
GPIO	-	General Purpose Input/Output
GPR	-	General Purpose Register
GUI	-	Graphical User Interface
HDL	-	Hardware Description Language
I/O	-	Input/Output
IDE	-	Integrated Development Environment
IEEE	-	Institute for Electrical and Electronic Engineers
IIR	-	Infinite Impulse Response
IOB	-	Input/Output Block
IRQ	-	Interrupt ReQuest
JTAG	-	Joint Test Action Group
KB	-	Kilo Byte
LSB	-	Least Significant Bit
LUT	-	Look-Up Table
LVDS	-	Low voltage Differential Signalling
LVTTL	-	Low Voltage Transistor-Transistor Logic
MACS	-	Multiply Accumulation Cycles per Second
MB	-	Mega Byte
MIMO	-	Multiple Input and Multiple Output

MIPS	-	Million Instructions Per Second
MPLB	-	Master Processor Local Bus
MRI	-	Magnetic Resonance Imaging
MSB	-	Most Significant Bit
MSPS	-	Mega Samples Per Second
NWU	-	North-West University
OPS	-	Operations Per Second
OS	-	Operating System
OTP	-	One-Time Programmable
P.O.	-	Percentage Overshoot
PA	-	Power Amplifier
PBMR	-	Pebble Bed Modular Reactor
PC	-	Personal Computer
PCB	-	Printed Circuit Board
PCI	-	Peripheral Component Interconnect
PI control	-	Proportional plus Integral control
PID control	-	Proportional plus Integral plus Derivative control
PLB	-	Processor Local Bus
PLC	-	Programmable Logic Controller
PLL	-	Phase Lock Loop
PMC	-	PCI Mezzanine Card
PMSM	-	Permanent Magnet Synchronous Motor
POWER	-	Performance Optimization With Enhanced RISC
PowerPC	-	POWER Performance Computing/Chip
PWM	-	Pulse Width Modulation
RAM	-	Random Access Memory
RDS	-	Rotor De-levitation System
RISC	-	Reduced Instruction Set Computer
ROM	-	Read Only Memory
RTM	-	Rear Transition Module
RTOS	-	Real-Time Operating System
SBC	-	Single Board Computer
SCADA	-	Supervisory Control And Data Acquisition
SDK	-	Student Development Kit
SharC	-	Super Harvard ARChitecture
SIMD	-	Single Instruction, Multiple Data
SO-DIMM	-	Small Outline Dual In-line Memory Module
SOS	-	Second-Order Section
SPLB	-	Slave Processor Local Bus
SPR	-	Special Purpose Register

SRAM	-	Static Random Access Memory
TBL	-	Time Base Lower
TBU	-	Time Base Upper
TCP/IP	-	Transmission Control Protocol/Internet Protocol
TI	-	Texas Instruments
UPS	-	Uninterrupted Power Supply
USB	-	Universal Serial Bus
V/F	-	Voltage over Frequency
VHDL	-	VHSIC Hardware Description Language
VHSIC	-	Very-High-Speed Integrated Circuits
VLIW	-	Very Long Instruction Word
WiMAX	-	Worldwide Interoperability for Microwave Access

CHAPTER 1

Introduction

This chapter provides background fundamental to understanding various concepts in this dissertation and project. It starts-off by discussing digital controllers in general and the advantages gained from using digital controllers. As short discussion on high-speed helium blowers follows. This is relevant, since the proposed active magnetic bearing and drive electronics system (ADES) should function in high-speed industrial systems such as the helium blowers. The requirement specification of the ADES is also based on a helium blower system.

1.1 Digital controllers

Digital system development has had a profound impact on the lives of people, like no other area of technology [1]. The digital firestorm is driven by [2]:

- **Utility** - Digital data can be replicated exactly and distributed effortlessly, which gives it much more utility
- **Quality** - The quality of digital data exhibits less noise.
- **Affordability** - Digital data manipulation is a great deal less expensive since it is able to influence semiconductor economies of scale. The price of digital data manipulation is dropping every year.

The rapid revolution of digital media and the digitization of consumer products have led to the improvements of existing digital products as well as the creation of various new products. Digital products are synonymous with [2]: quality, speed, accuracy, reliability, power, low cost and superiority.

More and more systems are moving away from analogue communication and control to take advantage of the above-mentioned characteristics. The same applies to the proposed active magnetic bearing and drive electronics system (ADES). The proposed ADES system is a digital system aimed at replacing the current rapid prototyping system with a controller suitable for usage in an industrial environment. By using digital communication between components the noise in the system will be reduced. This will lead to a higher bandwidth and more efficient control.

1.2 Background

Before the proposed system is given, it is first necessary to understand why the ADES should be capable of performing in an industrial environment. The primary intent of the ADES system is to replicate a high-speed helium blower control system typically used in a nuclear environment. The helium blower is just one of many applications for an AMB-system and the ADES will demonstrate AMB-control on an industrial level, regardless of the application. However, since the specification for the ADES is based on a high-speed helium blower, background on helium blowers is given to illustrate the usage of AMBs in an industrial environment.

1.2.1 High speed helium blower

The gas-cooled nuclear reactor (GCR) is just one of the many different types of nuclear reactors. It is a development of the earlier natural uranium Magnox type reactor, which, in various forms was the basis of the first nuclear power programme in the United Kingdom [3].

The Pebble Bed Modular Reactor (PBMR) is a new type of high temperature helium gas-cooled nuclear reactor [4] which falls into generation IV reactors [5], [6] building and advancing on world-wide nuclear operators' experience of older reactor designs. The most remarkable feature of these reactors is that they use attributes inherent in and natural to the processes of nuclear energy generation to enhance safety features.

Figure 1-1 illustrates the flow of helium in the PBMR. The helium gas is passed through the PBMR into the reactor. Situated inside the reactor are the fuel pebbles in which a chain reaction takes place. The helium flows over the fuel pebbles and is heated to a temperature of 900 degrees and the pressure is increased to 69 bars inside the reactor. The helium gas then flows through to the turbine, which drives a generator. The helium gas then goes through to a recuperator which pre- or re-heats the helium. Inside the recuperator, most of its heat is transferred to the helium - which is just about to re-enter the reactor. The lower energy helium gas is then passed to the intercooler, the low pressure compressor as well as the high pressure compressor, before returning to the reactor core at 540 degrees [7].

In order for the preceding process to function and to keep the PBMR safe, two helium blowers are required: one to blow the helium through the system, and one to regulate the temperature of the helium inside the reactor. The position of the helium blower(s) can be seen in Figure 1-1. (The latter blower is used in conjunction with a heat exchanger to perform the mentioned function. The heat exchanger is not depicted in the figure). The blowers depicted are said to be in the primary helium flow cycle – i.e. any cyclic flow of helium where the risk of contamination exists. These blowers should be high speed and should not contaminate the helium in any way. The use of active magnetic bearings (AMBs) is thus essential.

The possibility of helium blowers used outside the primary helium flow cycle also exists. These blowers are said to function in the secondary helium flow cycle; where the contamination of the helium is not critical. These blowers are application dependent, so they are not depicted in the figure.

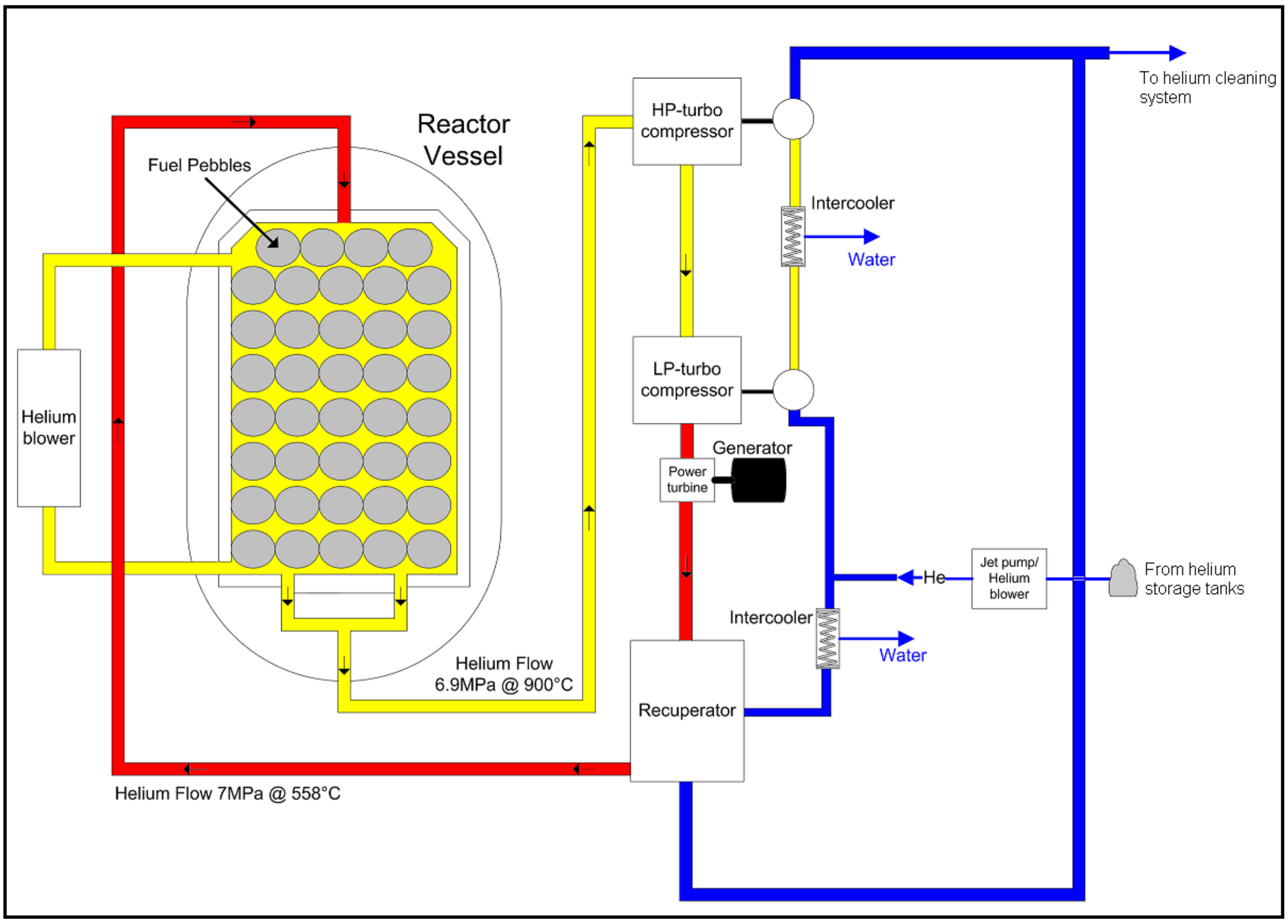


Figure 1-1: PBMR helium path schematic

1.2.2 Active magnetic bearings

AMBs are classified as a mechatronic product, because it combines mechanical, electrical and electronic principles [8], [9]. The term “active magnetic bearing” is due to the fact that the bearings can actively be controlled by means of a feedback loop [9], [10]. Bearing stiffness, damping, rotor position and many other features of the magnetic bearing can be adjusted.

Active magnetic bearings were manufactured due to the limitations of conventional bearings. Whereas conventional bearings are governed by friction and lubrication, AMBs have the ability to function without friction in vacuum, without any contamination due to lubrication. This makes AMBs effective in high-speed applications, as well as applications where there is a risk of contamination such as a high-speed helium blower.

1.2.3 Basic AMB operating principle

Unlike conventional bearings, in AMB systems the rotor is levitated by means of a magnetic field. As the suspension of the rotor is an unstable process, sensors and controllers are needed for stabilization. This implies that dynamic characteristics like stiffness and damping properties of the entire system are influenced by the controller [9], [10].

The functional diagram of a basic AMB system is shown in Figure 1-2. The magnetic field necessary to levitate the rotor is generated by means of an electromagnetic actuator and the displacement of the levitating rotor from the reference position is measured by means of a sensor. Various types of sensors can be used. The most common sensors are eddy current, capacitive, optical, inductive, Hall effect and sensor-less sensors (also known as self-sensing) [11].

A main controller interprets the signal from the sensor. This main controller is responsible for deriving a control signal from the signal delivered by the position sensor. This control signal is sent to a power amplifier (PA) which is used to convert the control signal into a control current necessary to drive the electromagnetic actuator to effectively suspend the rotor at the reference position.

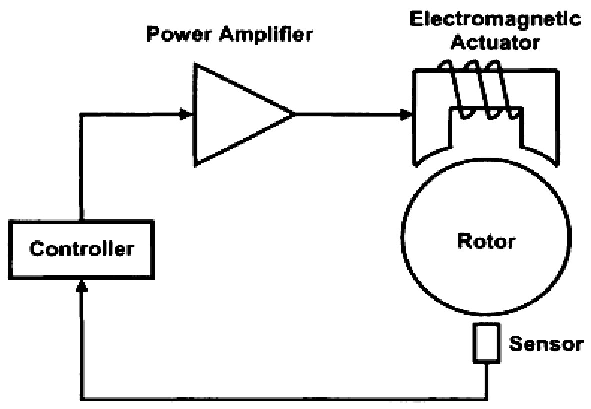


Figure 1-2: Functional diagram of a basic AMB system [9]

1.2.4 Components of an AMB

As can be seen in Figure 1-2, a basic AMB system constitutes the following components:

- A rotor
- Sensor(s)
- A main controller
- Electromagnetic actuators
- Electromagnets

The critical components in the system are the controller and the PAs. This is due to the power requirements for frequencies from DC to the kHz range [12].

The main controller can be divided into a few sub-categories:

- Analogue to digital converters (ADCs), depending on the type of sensors used
- Filters
- The main processor
- A co-processor[1]
- Digital to analogue converters (DACs), if analogue power amplifiers are used.

The main controller can be any of the following:

- Digital signal processor (DSP)
- A field programmable gate array (FPGA)
- A central processing unit (CPU) such as those used in personal computers (PCs)
- General purpose processor (GPP)

1.3 System architecture

The general AMB system layout was given in preceding sections. The subsequent sections will focus on the current AMB system implemented for research purposes.

1.3.1 Current system

The current system implemented for research purposes can be seen in Figure 1-3. Currently, this AMB system is controlled by means of an expensive dSPACE® system. This system incorporates the use of MATLAB® and Simulink® on a dSPACE® peripheral component interconnect (PCI) controller card to control the AMBs as well as a permanent magnet synchronous motor (PMSM). The dSPACE® system includes analogue and digital input and output (I/O) as well as an on-board microcontroller. A current reference is received by the dSPACE® system, which uses voltage over frequency (V/F) control to generate a pulse width modulation (PWM) signal, which in turn drives the three phase motor. The levitation of the AMB is controlled by means of a PID controller

The problem with this system is that, although it is an effective method of controlling the AMB system, it is extremely expensive, which makes it unsuitable for commercial use. A variety of research is underway to reduce the cost of an AMB system. This research include a single board computer (SBC), or a single controller solution as a replacement for the dSPACE®

[1] Application dependent

system. This controller will replace the entire dSPACE® system, which includes MATLAB®, Simulink® and the ControlDesk®.

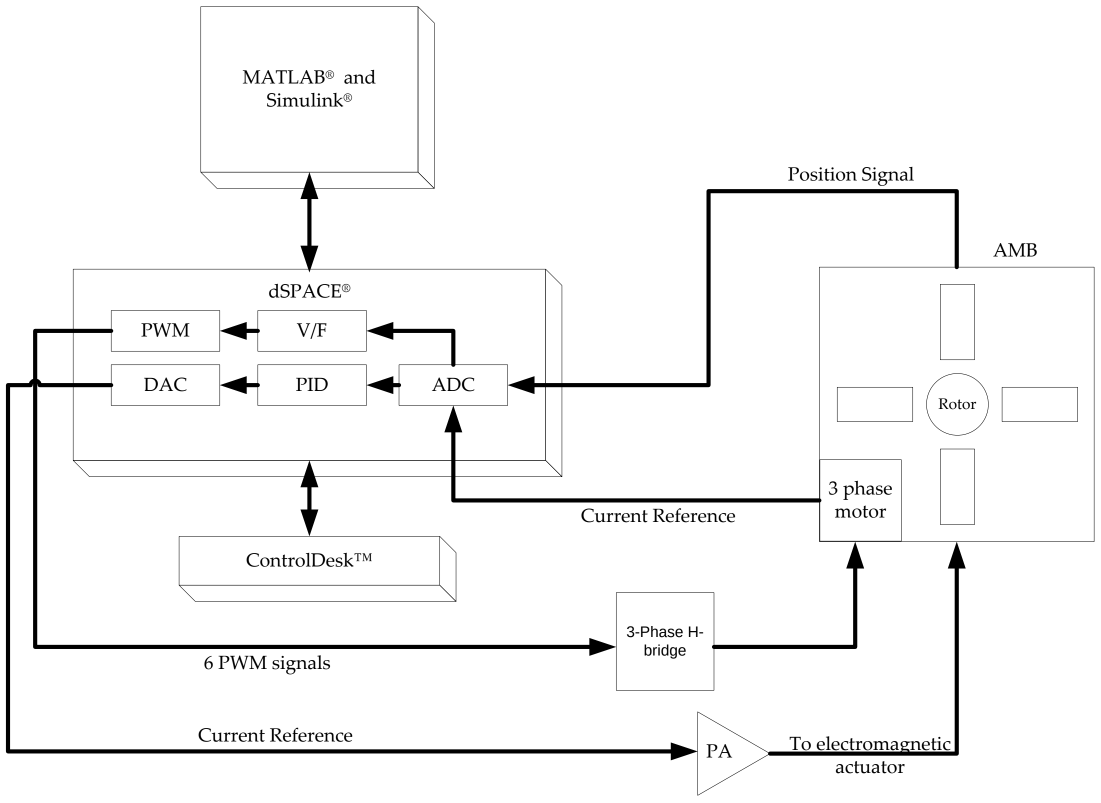


Figure 1-3: Current system used for research

1.3.2 Proposed system

Although an SBC is an effective replacement for the dSPACE® system, it is still an expensive commercial product. In order to realize the cost of the proposed commercial system, it was decided to design an electronic package to replace the dSPACE® system with a cost effective, efficient electronic package to control the AMB system. This electronic package will be called an ADES or AMB and drive electronic system. The proposed system can be seen in Figure 1-4.

Figure 1-5 shows the breakdown of the proposed ADES system. The components are:

- The main controller – which is responsible for the control of the system
- The sensor driver unit – which is responsible for sensing the rotor position
- PA units – which is responsible for driving the electromagnetic actuators
- The motor drive – which is responsible for driving the three phase motor
- Additional I/O unit – for any additional analogue I/O needed on the system

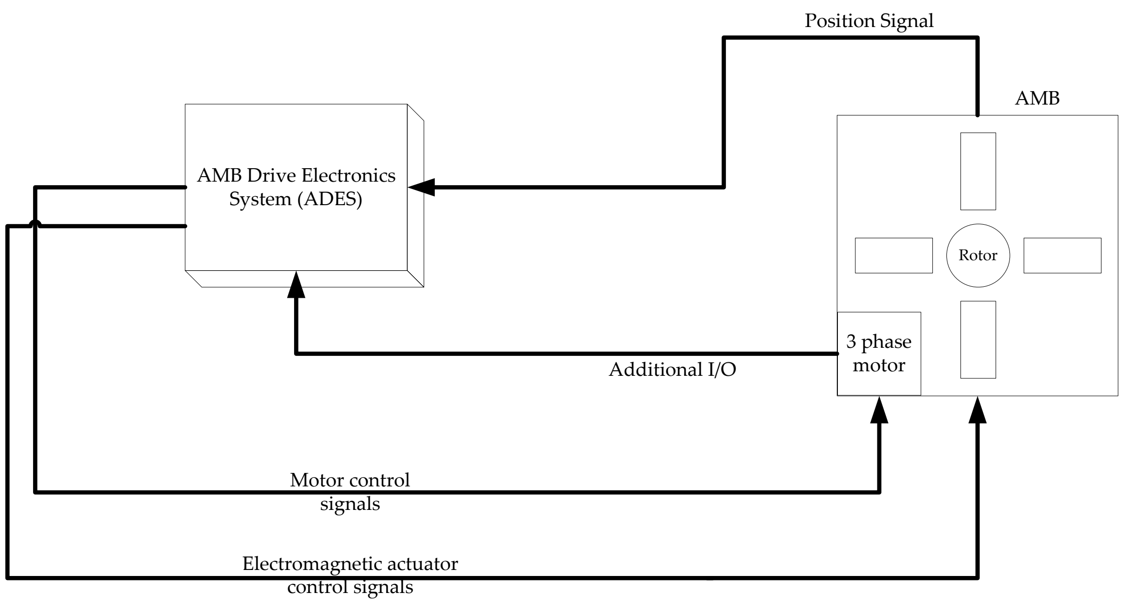

Position Signal
AMB
AMB Drive Electronics System (ADES)
Rotor
3 phase motor
Additional I/O
Motor control signals
Electromagnetic actuator control signals

Figure 1-4: Proposed ADES system

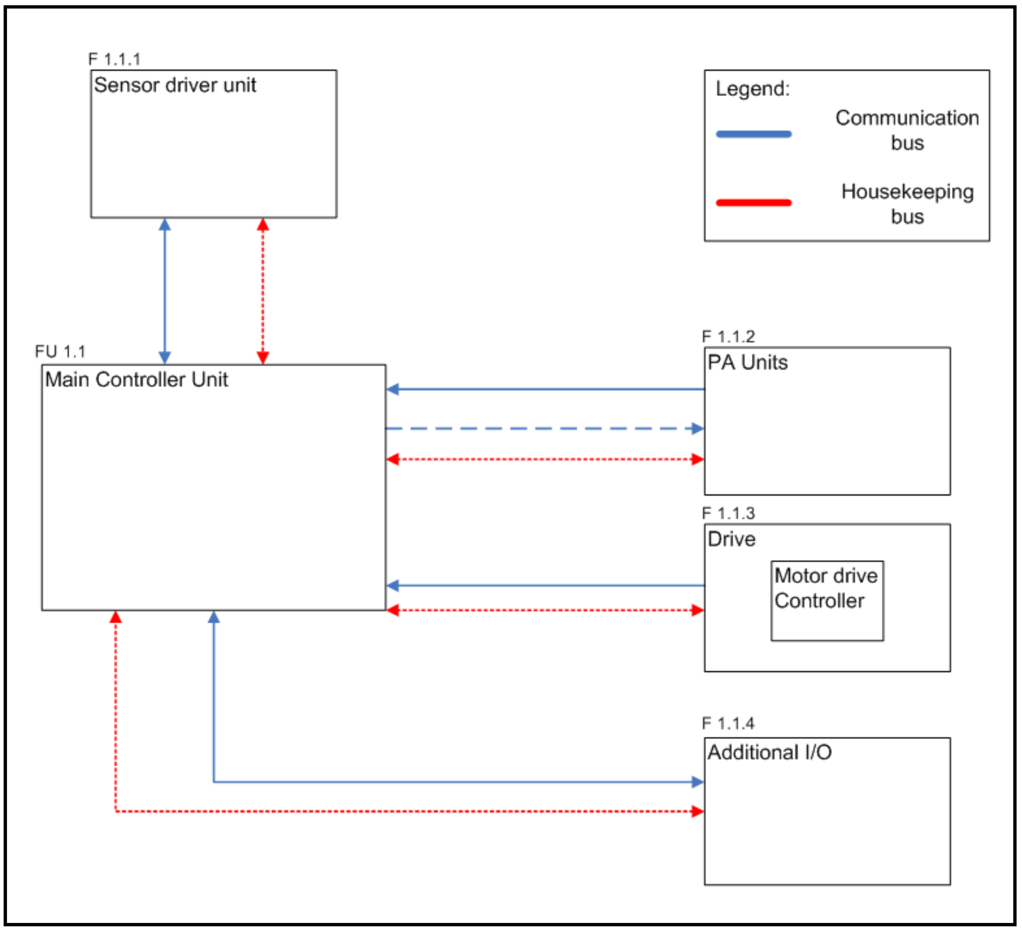


Figure 1-5: ADES breakdown structure

1.4 Problem statement

The purpose of this dissertation is the specification, selection, preliminary design[2], implementation and evaluation of an optimal controller suitable for performing the functions of the main controller of the ADES system. This implies the selection of a controller capable of performing these functions, the preliminary design of the hardware to accompany the selected controller and the outsourcing of the preliminary design to an industry partner for detail design (if necessary) the detail design of the firmware to be implemented in the selected controller, implementing the designed firmware on the developed hardware and finally the testing and evaluation of the design and implementation.

The controller should have the following functionality:

- Control of the AMBs and PMSM
- Condition monitoring of the currents, temperatures, speed of the rotor and status signals of the system
- Give a reference signal to the PAs to drive the electromagnetic actuators
- Perform self-diagnostic testing
- Perform sensitivity analyses
- Interface with memory modules

The aim of the ADES project as a whole is to take the level of control even closer to industrial system specifications and to address the cost-constraints of the system in future iterations in order for the ADES system to be commercially competitive. The primary aim of application is in high-speed systems such as a helium blower.

1.5 Issues to be addressed

This project proposes a few issues to be addressed in order to complete the project successfully. This section discusses these issues.

1.5.1 Conceptual analysis

Various concepts of the proposed system should be analysed. This is necessary in order to fully understand the functionality of the proposed system. The conceptual analysis should include concepts such as: intelligent components, communication speeds and architecture, processor types, practicality of the proposed concept as well as implementation implications.

[2] Only considered necessary if it is decided to develop the controller in-house

1.5.2 Functional analysis and allocation

The concept analysed in the preceding section will be the basis for drawing up numerous functions. Functional units will be allocated to these functions and allocated certain performance characteristics. Hardware will in turn be allocated to the functional units.

1.5.3 Controller selection

Selecting the right controller is a fundamental part of this project. The functions analysed and allocated in the previous section, will aid as a baseline to the controller selection. The controller needs to be selected on grounds of these functions that need to be performed. These functions will require certain performance parameters from the controller, and should also be performed within a specified time frame, which proposes timing issues.

1.5.4 Hardware preliminary design

The primary focus of this dissertation is on the controller selection and the implementation of the firmware running on the controller. The selected controller can either be a commercial product, or be developed in-house. If the latter option is selected, a preliminary design is needed before the hardware could be outsourced for detailed design. If needed, the preliminary design will consist of a B-spec document, which specifies the functionality of the proposed system, the performance characteristics of the system, as well as the hardware necessary to perform these functions.

1.5.5 Hardware outsourcing for detail design

After the preliminary design of the hardware, the hardware should be outsourced for detailed design i.e. the circuit bord layout design. This implies finding a reliable company with the necessary experience and acquaintance in designing printed circuit boards (PCBs), as well as establishing collaboration between the research group and the specified company. If a commercial controller is selected, this issue will not apply and hardware needs to be procured from the supplier.

1.5.6 Training in firmware development

The next issue is training in firmware development. This is due to the lack of experience in certain components which might be used i.e. field programmable gate arrays (FPGAs). This training could include firmware development in general, as well as training in the corresponding programming language for the chosen hardware components; such as VHDL (very high-speed integrated circuits hardware descriptive language) if an FPGA is used. This issue also includes the procurement of development boards, which will allow for parallel development of the firmware and hardware. This will be made possible by the fact that the firmware can be developed on the development boards, while the industry partner is designing the circuit boards or while waiting for the selected hardware to be sourced.

1.5.7 Firmware design and development

After sufficient training in firmware development, the next issue to address is the design and development of the firmware to be implemented on the developed or sourced hardware. This development will take place on the development boards, until the circuit design by the industry partner is complete or until the sourced hardware arrives. The firmware should act as an operating system and should:

- Contain algorithms for controlling the suspension of the AMBs
- Contain algorithms to generate a current reference to drive the PAs
- Contain PMSM drive algorithms
- Contain functions used in condition monitoring
- Contain functions used in self diagnostic testing
- Establish communication to memory
- Schedule events to trigger at a specified time

1.5.8 Implementation

The following issue is the implementation of the developed firmware on the hardware. The firmware designed on the development boards should eventually be implemented on the selected or manufactured hardware. This will occur during the implementation phase. This implies that all the requirements stated in section 1.5.7 and designed on the development board(s), be implemented on the developed or selected hardware.

1.5.9 Testing and evaluation

The final issue is the testing and evaluation of the hardware-firmware combination. During this phase the designed components need to be verified and the specifications of the system will also be validated by comparing the designed system with the system requirements.

1.6 Research methodology

Section 1.5 discussed the issues to be addressed during this project. Section 1.6 will in turn discuss how to address these issues.

1.6.1 Conceptual analysis

Before the conceptual analysis can take place, a literature study is needed to elucidate required concepts of the proposed system. These concepts will include similar systems developed and used in the industry. The requirements and specification of the proposed ADES system will also be based on the systems used in the industry. A detailed study of the industrial systems is thus essential. After defining the specifications and requirements from the literature, conceptual analysis can commence.

The conceptual analysis will include concepts to be analysed. Each concept will be analysed in terms of effectiveness, practicality and cost. Although cost is not a determining factor for the initial concept, it will play an important role in future iterations. By means of trade-off studies, the best concept and architecture will be determined.

1.6.2 Functional analysis and allocation

By analysing the functions of the system as a whole and by dividing the concept evaluated above into numerous sub-functions, the various functions can be allocated to the functional units of the system. This will ensure that the required functions are allocated to certain components.

After allocating the functions, the requirements will be linked to each function and the requirements document compiled.

1.6.3 Controller selection

In order to select the most suitable processor/controller for the ADES project, the performance parameters allocated in the previous section, has to be weighed. Each allocated function requires processing power and must be executed within a certain timeframe. It may also be that certain functions have specific memory requirements.

All these requirements must be brought into consideration when a processor is chosen. This will be done by using a timeline depicting the timeframe in which to complete one control cycle. This timeline will be divided into sections, allocating a part of the timeframe to each function. This, together with an estimate of the length, type and complexity of the instructions needed to be executed on the processor, will be used to determine the speed, processing power and type of the processor.

1.6.4 Hardware preliminary design/sourcing of hardware

If it is decided to develop a controller in-house, a preliminary hardware design is needed for the outsourced company to do detailed design and development. The preliminary design will state the type of processor as well as the necessary hardware required in this project. If it is decided to use a commercial system, the commercial system needs to be procured.

1.6.5 Hardware outsourcing for detail design

The preliminary design will be used by the outsourced company to do a detailed design of the system. The outsourced company can be an industry partner – who already has a relationship with the research group – or any other reliable company. If another industry partner is used, close collaboration should be established via effective and regular correspondence.

1.6.6 Training in firmware development

The firmware training can be categorized into two categories: dependent and independent. The dependent training is any training hosted by an external company whereas independent training is self-training by means of a variety literature on the subject of firmware, or courses hosted internally by experts on the area of firmware and the hardware specific programming language. The dependent training will take place if an external company can be found to host the course(s). The necessary development kits/boards should also be procured.

1.6.7 Firmware design and development

After the training discussed in the previous paragraph, the firmware design and development can commence in parallel with the hardware development or sourcing of commercial hardware, since development kits/boards were procured. The firmware will be developed using an integrated development environment (IDE) package.

An IDE is a software application extensively used in software development and consists of a source code editor, a compiler and/or interpreter, build-tools and usually a debugging facility. It can also incorporate version control software.

Due to the fact that firmware design and development are the main focus of this dissertation, significant time should be spent on the design and development of the firmware.

1.6.8 Implementation

After the development of the firmware and the hardware procured, integration of the software and hardware can commence. This implies the implementation of the firmware on the hardware. The implementation will be made possible by the usage of emulators, programmers as well as IDE packages.

1.6.9 Testing and Evaluation

The following quotes are from the definitions given for “verify”, “valid” and “validate”:

“Verify – Making sure or demonstrate that something is true, accurate of justified”

“Valid – Actually supporting the intended point or claim.”

“Validate – Check or prove the validity”.

Concise Oxford English Dictionary
10th edition 2002

The verification and validation of this project will be based on the above definitions. Verification of the project will be done by determining whether the project satisfies the specifications given. The project will be validated by testing different points and claims made i.e. finding and testing the truth of statements.

The testing, verification and evaluation will consist of:

- Verifying the hardware design by means of design reviews (if the controller is developed in-house)
- Verifying the design of the individual components using simulation packages
- Verifying algorithms against speed specifications or number of clock cycles it takes to execute
- Evaluating system stability by means of step responses
- Sensitivity analyses of the system
- Validating the designed system by comparing it to the specifications given

1.7 Dissertation overview

This dissertation consists of seven chapters. The succeeding chapter layout is as follows:

Chapter 2 discusses the literature surveyed. This literature encapsulates concepts and information essential to completing this project. Examples of concepts discussed are:

- Embedded control
- Microcontrollers and –processors
- Digital signal processing concepts
- Fixed and floating-point number representation
- The various processors and controllers considered for realizing the ADES
- The different design considerations, such as benchmarking and co-processing
- Digital implementations of PID control, IIR and FIR filters
- Object-orientated embedded C

The design considerations discussed in Chapter 2 are used to select a controller and system architecture. The selection process is discussed in Chapter 3. It starts off by giving the system specifications and processor requirements before a controller is selected. The selection process is performed by evaluating the considered architectures against decision matrices and by performing trade-offs between architectures. The system and main controller architectures are evaluated concurrently and the most suitable architecture for both is found.

After selecting the most suitable architecture, the hardware was sourced. The next step is designing the different components necessary for implementation. This is discussed in Chapter 4 which focuses on the design of components such as memory management and allocation, design of the embedded state machine, the design of the digital PID control, filters and interrupts. The implementation of these components are discussed in Chapter 5. Each component is also verified before it is implemented to ensure correct operation before implementation. The setup of a verification-model is also discussed in this chapter. Verification is either done by first simulating the component in MATLAB®, by using ModelSim® or by using system emulation.

Chapter 6 evaluates the main controller by first discussing the performance of the controller. This is done by discussing the execution timing of the controller and by determining whether the controller is capable of performing multiple input and multiple output (MIMO) control. The stability of the system is then determined. This is done by determining the step response of the system. The settling time, percentage overshoot, damping ratio, natural frequency, equivalent stiffness and equivalent damping is then calculated for the x and y-axis of both AMBs. The stability-test is followed by a sensitivity analysis. The sensitivity analysis is used to determine the stability margin of the system according to the ISO CD 14839-3 standard.

The dissertation concludes by giving conclusions and recommendations in Chapter 7. The issues encountered during the project as well as discrepancies in results are discussed. Conclusions are drawn regarding the selected processor and its efficiency in controlling the ADES. Future work is discussed and a closure statement is given.

CHAPTER 2

Literature study

This chapter provides insight to concepts necessary for understanding and completing this dissertation. Since this project is aimed at selecting, developing and implementing an embedded controller, this chapter is initiated with an overview on embedded control, which defines certain concepts and discusses different types of controllers and processors. The design considerations for selecting a controller or processor are also discussed in this chapter. Lastly, the focus is shifted to active magnetic bearing (AMB) control and digital filters.

2.1 Embedded control

This chapter starts-off by discussing embedded control by first giving background on microprocessors and –controllers.

2.1.1 Background

Although microcontrollers and microprocessors have distinct features, as will be discussed in section 2.1.2, microcontrollers have existed ever since the first microprocessors. In November 1971, Intel® released the world's first single chip microprocessor. It was a four bit microprocessor called the 4004. The following years saw the release of the first microcontroller from Texas Instruments® (TI) dubbed the TMS 1000 as well as eight bit microprocessors from Intel® (the 8008 and 8080). The first Intel® microcontroller was released in 1975 and was called the 8048. It featured random access memory (RAM) and read only memory (ROM) inclusive on the chip [13].

Ever since the first microprocessors and microcontrollers arrived on the scene in the 1970's, the world of electronics was changed forever. Numerous companies started developing a large variety of controllers and processors. The first microprocessors from Intel® were mainly used in the first personal computers (PCs), whereas the microcontrollers were used to control electronic apparatus and systems. Table 2-1 depicts a brief summary of the history of microcontrollers and processors. The main focus of this dissertation is on embedded control. This implies that an embedded microcontroller and/or -processor will be utilized in an embedded system. It was considered relevant to give definitions on embedded systems and control, just to clarify these concepts:

- **Embedded system -** "A computer system that is a component of a larger machine or system. Embedded systems can respond to events in real time" [14].
- **Embedded controller -** "A device that performs embedded control" [15].

In this instance, the ADES is classified as an embedded system, whereas the main controller unit is defined as an embedded controller.

Table 2-1: Brief summary of the microcontroller/microprocessor history [16] [17] [18] [19]

Year	Microprocessor/microcontroller	Remark
1971-1972	Intel® 4004, Intel® 4040	4-bit microprocessors
1974	Intel® 8080, TMS 1000	8-bit microprocessor
1975	Motorola® 6800	8-bit microprocessor
1976	MCS-48, Intel® 8085	8-bit microcontroller
1978	8086, Motorola® 68000, Zilog Z-8000	16-bit microprocessors
1979	Intel® 8088	8 bit microcontroller
1980	Intel® 8051	8 bit microcontroller
1982	68010, 6805, 80186, 80188, 80286, 8096 (MCS-96)	16-bit microcontrollers
1984	Motorola® 68020	32-bit microprocessor
1985	Intel® 80386 PIC microcontrollers by Microchip®	32-bit microprocessor 8-bit microcontrollers
1987	Zilog Z280	16-bit microprocessor
1989	Intel® 80386xx, 80486	32-bit microprocessor
1993	Intel® Pentium™	32-bit microprocessor
1995	Intel® Pentium™ Pro	32-bit microprocessor
1997	Atmel® 8-bit AVR family Intel® Pentium™ II and Xeon™	8-bit RISC microcontrollers 32-bit microprocessor
1999	Intel® Pentium™ III, Celeron™, Pentium™ III Xeon™	32-bit microprocessors
2000	Intel® Pentium™ 4	32-bit microprocessor
2003	Intel® Pentium™ M	32-bit microprocessor
2006-2007	Intel® Core™ 2 Duo and Quad	64-bit microprocessor
2008	Intel® Core™ *i*7	64-bit microprocessor

Since the terms “microcontroller” and “microprocessor” is used quite a lot throughout this dissertation, the next section is dedicated to explaining the difference between a microcontroller and microprocessor, as well as defining the terms “microcontroller” and “microprocessor”.

2.1.2 Microprocessor vs. microcontroller

The significant improvements in technology over recent years have caused the separation-line between a microcontroller and microprocessor to fade. Initially a microprocessor was defined as “a central processing unit (CPU) only” [13]. This implies that a microprocessor is simply a unit that performs processing and all the additional peripherals (such as RAM) are separate from the processor. A microcontroller, on the other hand, is a device where some of the peripherals are included onto the chip. Traditionally, these controllers were less powerful than their processor counterparts. However, over the years the processing power on the microcontrollers has increased; so much so, that it is comparable to microprocessors. Modern microprocessors have some peripherals included on chip. A microprocessor is even defined in [18] as “a digital machine capable of executing many different software instructions and controlling a wide variety of electronic devices”. This can be quite confusing since it implies that a processor can be used for control and vice versa.

This fine line between microprocessors and microcontrollers deemed it necessary to define a microprocessor and microcontroller as used in this dissertation. For this dissertation, a microcontroller is defined as a device that performs control, whereas a microprocessor is a device that performs processing. Different microcontrollers and microprocessors from various companies are listed in Table 2-2.

Table 2-2: List of microcontrollers and -processors

Microcontrollers		Microprocessors	
Supplier	**Series**	**Supplier**	**Series**
Texas Instruments®	TMS320C2000	Texas Instruments®	TMS320C6000
Analog Devices®	MicroConverter	Analog Devices®	Blackfin™ SHARC™ TigerSHARC™ ADSP-21xx
Microchip®	PIC dsPIC	Intel®	Various (refer to Table 2-1)

The definitions for embedded controller and embedded processor as used in this dissertation (based on the definitions given above) are as follows: an embedded controller is an embedded system dedicated to control whereas an embedded processor is an embedded system dedicated to processing. The term “embedded” implies a component which is part of a larger system, such as a processor or controller, where the component is able to respond to commands in real-time and is generally not programmable by the end-user [20], [21].

The main controller unit for the ADES is classified as an embedded controller. It is thus an embedded system dedicated to control, regardless of whether a microcontroller or microprocessor is used to establish control.

2.1.3 Digital signal processors and processing

This section discusses digital signal processors and processing. According to [22], digital signal processing is “concerned with the digital representation of signals and the use of digital processors to analyse, modify, or extract information from signals”. Smith [23] defines digital signal processors as “microprocessors specifically designed to handle digital signal processing tasks”. These two definitions define signal processing and digital signal processors respectively.

Most signals are analogue in nature, and a digital signal processor (DSP) can be used to derive a digital representation of the analogue signal. If the signal is already in digital form, a DSP can be used to manipulate, or perform calculations, on the data. As mentioned in 2.1.2, a processor is extremely powerful and can perform a large number of calculations in a short period of time. It can be deducted that a digital signal processor is thus a processor optimized for digital manipulation of data such as [22]:

- Convolution
- Correlation
- Digital filtering
- Discrete transformation
- Modulation

In general, DSPs excel in two broad areas: data manipulation and mathematical calculation, however, it is extremely difficult to design a device that is optimised for both. This is due to hardware considerations, the size of the instruction set and how interrupts are handled [23].

The advantages of using a DSP are given in [22] as:

- The accuracy is guaranteed by the number of bits used
- Performance can be reproduced perfectly, since there are no variations due to component tolerances
- The performance is not influenced by temperature or age

- Flexibility is enhanced since any modification in the system is done in software and not in hardware
- A DSP has increased performance over analogue signal processing since a DSP can perform functions not possible (or difficult) to perform with analogue signal processing

Since DSPs are digital components, the data used by the DSPs should be represented digitally, i.e. by means of binary 0's and 1's. In general, two types of controllers and processors exist: fixed-point and floating-point. The next section is dedicated to discussing the difference between these two number formats. Please note that the subsequent section focuses on the digital signal processing capabilities of either processors of controllers (as defined in section 2.1.2) and not on the physical hardware.

2.1.4 Fixed-point vs. floating-point

DSPs offer two main categories of processors: fixed-point and floating-point. This refers to the numbering format used by the particular processor. The best way of describing fixed-point processors are by means of an example: say for instance that a certain company manufactures a 16-bit fixed-point processor. This implies that 2^{16} = 65,536 different bit patterns can be used to represent a number (in general, an *n*-bit number gives 2^n different bit patterns). The number can be represented in the following formats:

- Unsigned integer – This implies any positive integer from 0 to 65,535.
- Signed integer – Two's complement is used to represent an integer from –32,768 to 32,767.
- Unsigned fraction – The 65,536 different levels are spread between 0 and 1.
- Signed fraction – The 65,536 levels are spread between -1 and 1.

A positive integer value is normally represented by the numerical weight given to a bit position. This can be seen in Figure 2-1 [24].

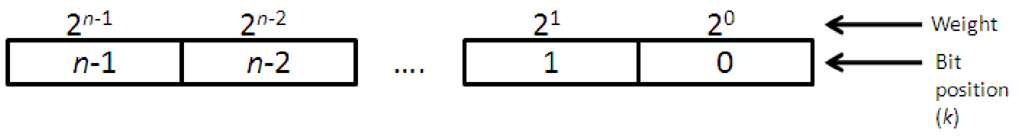


Figure 2-1: Numerical bit-weight for positive integer numbers

The value of the represented value is then given by the sum of the weights:

$$\sum_{k=0}^{n-1} b_k \cdot 2^k$$

where *b* is the bit (0 or 1) in the *k*th position of the bit string. For example, the binary number 101101_2 is represented as depicted in Figure 2-2. It is then summed as:

$$1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 45$$

Thus, $101101_2 = 45_{10}$.

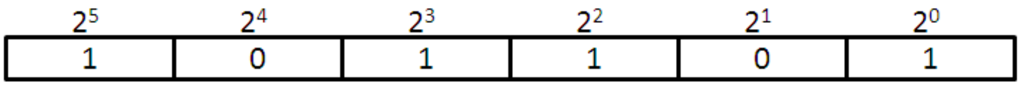


Figure 2-2: Example: 101101_2 as an unsigned integer

A positive fraction is represented in the same way as mentioned above, but with the weights changed as presented in Figure 2-3.

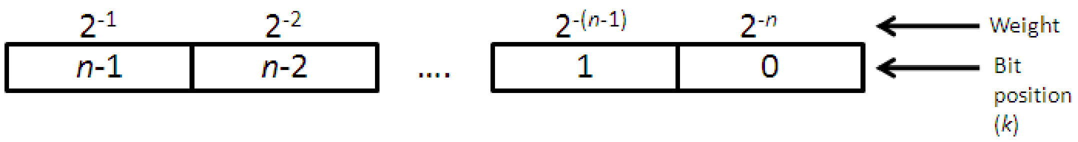


Figure 2-3: Numerical bit-weight for positive fractions

The represented value is then given by:

$$\sum_{k=1}^{n-1} b_k \, . \, 2^{k-n}$$

Figure 2-4 shows an example of a positive fraction: 0.010110_2, which can be summed as follows:

$$0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 1^{-4} + 1 \times 2^{-5} + 0 \times 2^{-6} = 0.343750$$

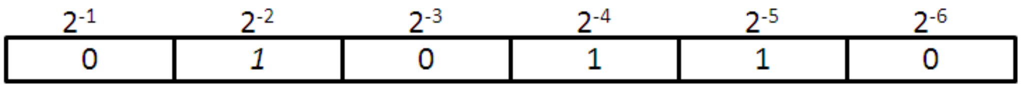


Figure 2-4: Example: 0.010110_2 as an unsigned fraction

To represent the negative values of the two formats in signed notation, two's complement is used. To understand this concept, it is first necessary to understand one's complement. The one's complement of a binary number is obtained by replacing the 1's with 0's and vice versa [25]. Take 45 for example. As discussed previously, it is represented by 101110_2. The one's complement of 101110_2 is 010001_2. To obtain the two's complement, one (1) is added to the one's complement. For example:

$101101_2 \longrightarrow$ Number

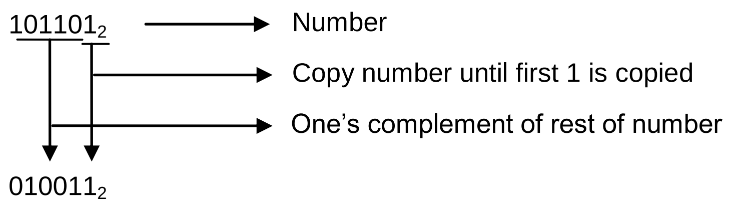
$010010_2 \longrightarrow$ One's complement

$+ \quad 1_2 \longrightarrow$ Add one

$010011_2 \longrightarrow$ Two's complement

The one's complement of 101101_2 (as per example above) is 10010_2. Adding one (1) to this number, two's complement is obtained as: $10010_2 + 1 = 10011_2$, which is the two's complement representation of 45, or -45.

A simplified method for obtaining the two's complement, is to copy the number from right to left until the first one (1) is copied and then to take the one's complement of the rest of the binary numbers [25]. If the first example of 101101_2 is taken:



101101_2
Number
Copy number until first 1 is copied
One's complement of rest of number
010011_2

The same method could also be applied to obtain the negative of the signed fraction notation. The two's complement method is also very useful when two numbers need to be subtracted. This can be done by adding a two's complement of a number to another number to subtract.

These formats (signed/unsigned integer and fraction representations) are collectively called fixed-point representations [24]. The representation of fixed-point numbers is limited to the bit-capability of the processor. Floating-point processors, on the other hand, typically use a minimum of 32-bits to represent a value. This implies that 2^{32} = 4,294,967,296 bit patterns can be used to represent a value. The most important difference between a fixed and floating-point processor are that in floating-point processors, the values are not uniformly spaced. The ANSI/IEEE 754-1985 standard defines the finite positive and negative numbers furthest from zero as $\pm 3.4 \times 10^{38}$ and the positive and negative normalized numbers closest to zero as $\pm 1.2 \times 10^{-38}$. The represented numbers are unevenly spaced between these limits, whereas the spacing between two integer-values is exactly one [23].

The encoding of floating-point numbers is more difficult than for fixed-point numbers. The basic idea behind floating-point number representation is fundamentally the same as scientific notation. In scientific notation, a number is represented as a mantissa multiplied by ten to the power of an exponent. Take the speed of light (*c*) for example. The speed of light is given as 299,792,458 m/s. This can be written as: 2.99792458×10^8, with 2.99792458 the mantissa and 8 the exponent. The fundamental difference between floating-point and scientific number representation is that floating-point numbers are represented in binary notation in respect to the decimal notation used in scientific notation. The ANSI/IEEE 754-1985 standard also defines 32-bit floating-point numbers as single precision and 64-bit floating-point numbers as double precision. For single precision numbers, the first 23 bits (bits 0 to 22) is used to represent the mantissa, bits 23 to 30 represents the exponent and bit 31 represents the sign (positive or negative) of the represented value [23]. This can be seen in Figure 2-5.

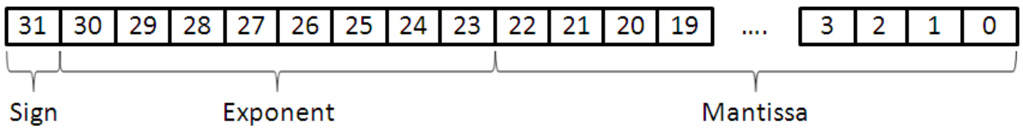


Figure 2-5: Single precision floating-point number representation

This can be converted into a floating-point number v [23]:

$$v = (-1)^S \times M \times 2^{E-127}$$

with S = 0 to represent a positive number and S = 1 to represent a negative number. E is the decimal value of the 8-bit exponent and M is the unsigned fraction represented by the 23-bit mantissa. Since the only non-zero number in binary notation is 1, the leading digit of the mantissa will always be 1 and does not need to be stored as an explicit bit. This improves precision with one bit to 23 bits. The mantissa can thus be calculated as:

$$M = 1 + M_{22}2^{-1} + M_{21}2^{-2} + M_{20}2^{-3} + M_{19}2^{-4} \ldots.$$

For double precision, the 64-bits are divided as follows: the first 52 bits (bits 0 to 51) are used to represent the mantissa, bits 52 to 62 represent the exponent and the last bit, bit 63, represent the sign bit. Double precision floating-point can represent extremely large and small numbers. In some applications the resolution or range of single precision is not sufficient and double precision numbers are then a viable alternative.

All floating-point DSPs have the capability to handle fixed-point as well as floating-point numbers, since counters, loops and timers require fixed-point numbers. However, this does not mean that a floating-point DSP can execute the fixed-point instructions as fast as floating-point instructions, or vice versa. The execution time of a certain instruction, regardless of the number representation, depends on the internal architecture of the DSP [23].

The question could arise that if a fixed-point processor can represent and manipulate integer and fractional numbers, what are the advantages gained by using a floating-point processor? The answer resides in the fact that floating-point processors have a higher precision, higher dynamic range and a higher signal-to-noise ratio. The reason behind the higher sinal-to-noise ratio is due to the spacing between two consecutive values. In a fixed-point processor, the distance (or gap) between two values are exactly one, whereas the distance between two values in a floating-point processor depends on the size of the numbers. The distance between two large numbers, is large. The distance between two small numbers, is small. When a number is stored, it is rounded up or down with a maximum of one half the gap size, thus adding noise to the number. This occurrence is exactly the same when using fixed-point values. The only difference being that the spacing between fixed-point values is larger [23].

Now that a basic idea of what controllers and processors are, how they work and how they represent numbers is formulated, the next section focuses on the different types of controllers.

2.1.5 Controllers

The word “controller” as used in this dissertation is defined in section 2.1.2. The selection of a controller for realising the ADES is not limited to a single DSP, per se, and could also be a combination of controllers, a field programmable gate array (FPGA) or any controller/processor with the necessary processing power and DSP capabilities to perform the required functions of the ADES. This section is dedicated to discussing the different types of controllers.

Digital signal processors (DSPs)

Section 2.1.3 discussed digital processor and digital processing in general and also gave a few advantages of using digital data over analogue data. The focus of this section is on the processors: the hardware, architecture, etc.

DSPs are manufactured by various companies. The two major companies in consideration (due to their powerful range of DSPs) are Analog Devices (ADI) and Texas Instruments (TI). The SharC™ and TigerSHARC™-ranges of DSPs are manufactured by ADI, whereas the TMS-range is manufactured by TI.

SharC™

The SharC™-range is a 32-bit DSP from ADI and is based on the super Harvard architecture (from there the name SharC™ – Super Harvard ARChitecture). It is ADI’s DSP leader in floating-point applications and offers high floating-point performance. The range also has application-specific peripherals included on chip and specifically designed interfaces to reduce cost. The SharC™ range of DSPs provides high-end DSPs which delivers up to 2.4 GFLOPS (Giga floating-point operations per second) of processing power running at 400 MHz [26].

SharC™ processors are ideal for [26]:

- Home theatre audio systems
- Professional audio systems
- Automotive audio systems
- Industrial and instrumentation equipment
- Medical imaging
- Telephony

TigerSHARC™

The TigerSharC offers the industry’s highest performance per watt per square inch of board space [26], [27], [28] and is very effective in demanding signal and image processing applications. The TigerSHARC™ is very effective in multi-processor systems due to its patented link-port technology which provides effortless inter-processor communication between two or more TigerSHARC™s.

The TigerSHARC™ is based on a 128-bit static superscalar architecture and offers native support for both fixed and floating-point computations. It's high I/O bandwidth and high processing power gives the TigerSHARC™ a high I/O bandwidth-to-processing ratio (BPR). A BPR of 1 B/FLOP (byte per floating-point operation) implies a well balanced processor suitable for real time applications. This means a processor is capable of moving 1 byte of data on or off the chip for every floating-point operation it can perform. A BPR of significantly lower than one, implies a processor more suitable for batch processing, whereas a BPR higher than one is better suited for data movement. The ADSP-TS101S TigerSHARC™, for example, has a BPR of 1, making it an effective processor in real-time applications [29].

TigerSHARC™s are ideal for [26]:

- Wireless infrastructure WiMAX (worldwide interoperability for microwave access) applications.
- Floating-point, performance density related systems in both single and multiprocessor systems.
- Medical imaging equipment (such as computed axial tomography or CAT scan, ultrasound and magnetic resonance imaging (MRI)).
- Military equipment.
- Industrial and instrumentation equipment.
- Automated test equipment.

The TigerSHARC™ was awarded numerous titles and nominations including:

- The industry's highest DSP performance [30]
- Top floating-point benchmarking score [31]
- And two nominations for innovation of the year (2002 and 2003) [32], [33]

Although some of these awards and nominations are a bit old, the TigerSHARC™ is still one of the best processors in the market.

Texas instrument processors

The TI range of controllers and processors has a lot to offer in terms of processing power, but since it was decided that the main controller unit should be a floating-point processor (for more information of the choice of processor, refer to Chapter 3), the TI range of DSPs was limited to the TMS320C67x™ DSP series, since this is the only DSP from TI capable of floating-point operations. This series are 32-bit DSPs with advanced very long instruction word (VLIW) architecture. VLIW is an instruction consisting of various sub-instructions executed at once. This series DSPs is also capable of executing eight 32-bit instructions per cycle (by means op pipelining instructions) and has eight functional units and sixty-four 32-bit registers [34].

Applications for the TMS320C67x include [34]:

- Professional audio mixers
- Mixers
- Audio synthesis
- Instrument/amplifier modelling
- Audio conferencing and broadcast
- Biometrical and medical
- Digital imaging
- Speech recognition
- Voice-over packet

Field Programmable Gate Array (FPGA)

The sections above discussed off-the-shelf DSPs, i.e. processors bought from a supplier capable of performing DSP instructions. Control of the ADES system can also be performed using a field programmable gate array (FPGA). FPGA devices were introduced by Xilinx® in the mid 1980s. FPGAs are silicon devices that can be programmed to be almost any kind of digital system. The main difference between the DSPs discussed above and the FPGAs discussed in this section, is that the DSPs of the previous section have fixed hardware architectures, whereas the hardware architecture of an FPGA is described using a hardware descriptive language (HDL) called VHDL. The "V" in VHDL stands for another abbreviation: VHSIC or very high speed integrated circuit, whereas the "HDL" stands for "hardware descriptive language" as mentioned earlier [35], [36]. The basic internal architecture of an FPGA can be seen in Figure 2-6 [37].

As can be seen, an FPGA consists of an array of programmable logic blocks (CLBs) as well as some memory and multiplier blocks. The programmable logic blocks can be programmed to be any type of logic (such as AND, OR, XOR, etc.). The different blocks are interconnected by a programmable routing fabric (or switch matrices) which allows the blocks to be connected in various ways. The I/O blocks in the figure are used to interface the FPGA to the outside world.

The basic internal architecture of a Xilinx® FPGA can be seen in Figure 2-7. As can be seen, the internal architecture of a Xilinx® FPGA is fundamentally the same as the FPGA architecture seen in Figure 2-6. It also consists of a matrix of complex logic blocks connected using switch matrices. The other components in the figure are explained below [38]:

- **IOB – I/O banks**
  Modern FPGAs include several I/O standards. These I/O standards are grouped in banks. Each bank is able to support different I/O standards.

- **DCM – Digital clock management**
  DCM is used in most FPGAs to minimize skew and other issues with designing global FPGA signals.

- **SRAM – Static random access memory**
  The SRAM on the FPGAs are used to store interconnect (switch matrix) configuration [36]. Since SRAM is volatile, this makes the FPGA volatile, although one-time programmable (OTP) FPGAs, using flash, are available.

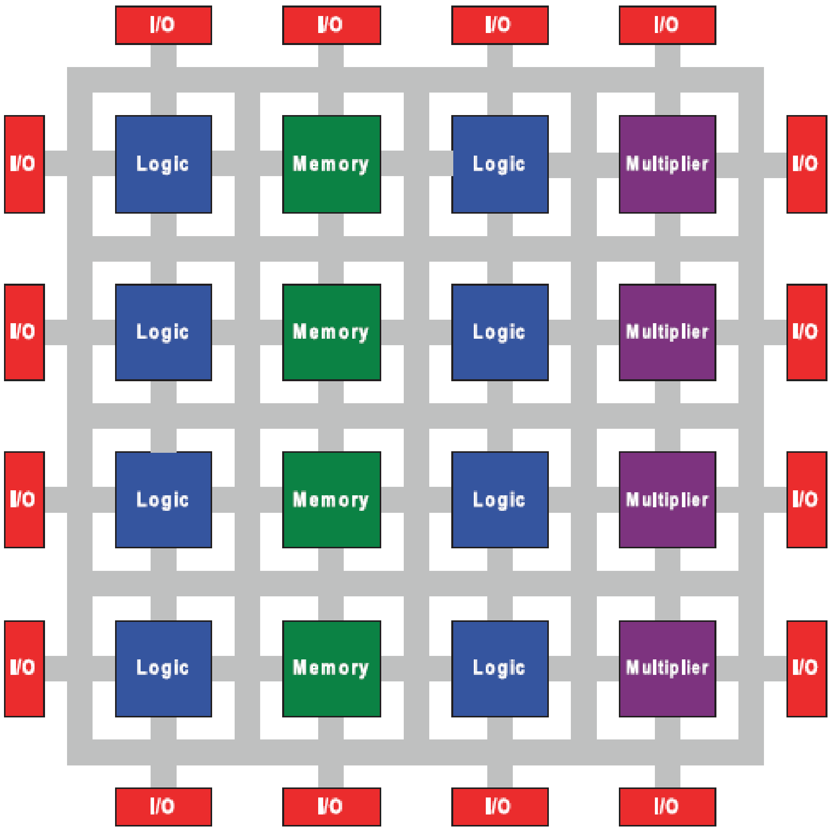


Figure 2-6: Basic FPGA internal architecture

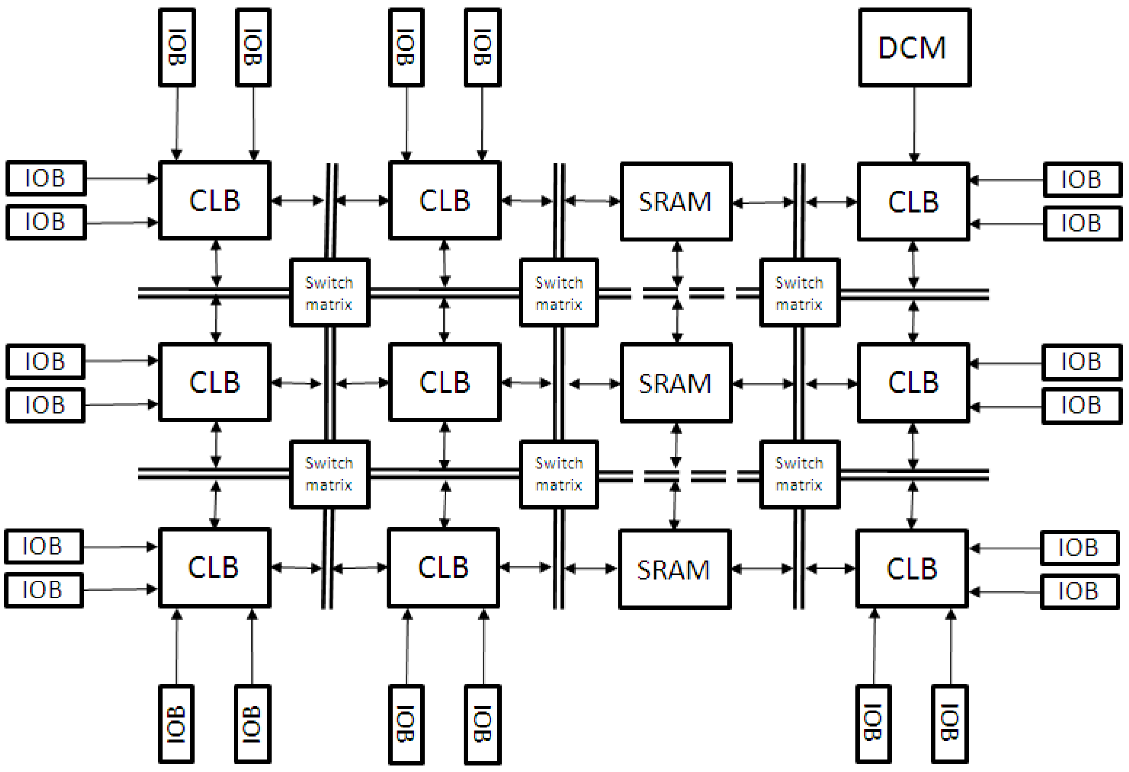


Figure 2-7: Xilinx® specific FPGA architecture [36], [38]

The CLBs are the most basic building blocks of any FPGA. The number and features of the CLBs are device dependent and whereas a programmable logic device (PLD) uses AND gates followed by OR gates, its operation is based on lookup tables (LUT) [36]. The switch matrices (or programmable interconnects) are part of the CLBs. They have 4 or 6 inputs and contain some selection circuitry, like multiplexers, and flip-flops. The switch matrices are modular and can be configured for combinational logic, shift registers or additional RAM [38].

There are five reasons to use FPGAs in DSP applications and for signal processing [39]:

1. **Ability to handle very high computational workloads**
   Due to the fact that FPGAs are capable of executing concurrent instructions, it is possible for the sample rate to match the clock rate. It is thus possible to have performance levels of up to 500 mega samples per second (MSPS). This is ideal for fast single channel systems or multiple slower channel systems.

2. **Reducing computation tasks**
   The computation intensive tasks can be offloaded from the DSP, which leaves more cycles to implement other functions.

3. **Power efficiency**
   FPGAs deliver lower power at higher sampling rates.

4. **The architecture can be customized to suit the ideal algorithm**
   The configurability of the FPGA allows custom architectures to suit a specific algorithm.

5. **Reduce system cost**
   FPGAs allows for the integration of external components into a single FPGA. This reduces the overall system costs. Examples include serial interfaces and PCI express interfaces

Advanced Reduced Instruction Set Computer Machine (ARM)

As seen from the section heading, ARM is short for Advanced RISC machine, where RISC stands for reduced instruction set. Prior to becoming Advanced RISC machine, ARM was known as Acorn RISC machine. ARM processors are 32-bit RISC processors developed by ARM Limited. Advanced RISC machine Ltd. was founded in 1990 and is owned by Acorn, Apple Computers® and VLSI Technology [40]. ARM processors is one of the most licensed and widespread processors in the world [40] and is used in more than 90% of the mobile phone market [41].

Some features of the ARM architecture are [40]:

- It has 37 pieces of 32-bit registers.
- Have three pipelines for pipelining instructions.
- Have a Von Neumann-type bus structure in the ARM7 processors and Harvard architecture in the ARM9 processors.
- 8/16/32-bit data types.
- 7 modes of operation:
- User - Normal program execution state
- Fast interrupt request (FIQ) - Data transfer state (DMA-type transfer)
- Interrupt request (IRQ) - Used for general interrupt services
- Supervisor - Protected mode for operating system support
- Abort mode - Selected when data or instruction fetch is aborted
- System - Operating system ‘privilege’-mode for user
- Undefined - Selected when undefined instruction is fetched
- It has a simple structure, with reasonable good speed, power and compression ratio.

PowerPC

The PowerPC alliance started in September 1991 when IBM®, Apple Computers® and Motorola® (conjointly known as AIM) announced an alliance to develop emerging technologies [42]:

- Object orientated technology
- Multimedia technology
- Interconnectivity and networking
- Open systems environment
- Microprocessor technology

IBM® and Motorola® (now Freescale Semiconductor®) agreed to develop a family of microprocessors based on IBM®'s POWER™ (which is an acronym for: Performance Optimization with Enhanced RISC) architecture. The new family of microprocessors was dubbed the PowerPC (POWER performance computing/chip). The PowerPC processors are based on the PowerPC architecture which is a RISC architecture optimized for diverse computing requirements [42].

The PowerPC architecture defines the following [43]:

- Separate 32-entry registers for integer and floating-point instructions
- Instructions for data movement between the memory system and general purpose registers (GPRs), which holds the data for the integer arithmetic, and floating-point registers (FPRs), which holds the arithmetic for floating-point instructions
- Uniform-length instructions to simplify instruction pipelining and parallel processing instruction dispatch mechanisms
- A precise interrupt model
- Floating-point support (including IEEE-754 floating-point operations)
- Single and double precision floating-point capabilities
- A flexible architecture which allows certain feature to be performed in either hardware or software (with the help of implementation-specific software)
- User-level instructions for storing, retrieving and validating data in the on-chip caches
- Little- and big-endian addressing support
- Harvard architecture and unified cache support

The PowerPC architecture is so modular; PowerPC hardware cores are also included in some FPGAs as an embedded processor to add additional functionality. It is also possible to install an embedded or real-time operating system onto a PowerPC processor [44].

Off the shelf solutions

The preceding sections covered the various DSP processors in consideration for the ADES project. Another possibility for realizing the main controller unit is by purchasing a commercial off-the-shelf (COTS) component. Various options exist and this section discusses these options.

PC/104

As modern-day applications get more computational intensive, the personal computer (PC) became more involved than just for desktop applications. By standardizing desktop computer technology for embedded applications, the development time and cost can be reduced. Since standardized software and hardware are widely available for the PC architecture, even more advantage is gained by using the PC architecture [45].

For the abovementioned reasons, more and more companies tried to reap the benefits of the PC architecture. The main disadvantage of the PC architecture is the bulky backbone necessary for most PC form factors. The solution to this disadvantage was to design a PC directly into the product. This in turn had another disadvantage. By designing a PC, chip-by-chip, into a product borders on the concept of "reinventing the wheel". Management therefore tries to outsource as many components as possible to reduce time and cost of development [45].

A need therefore arose for a more compact form factor for the PC bus. This need should be realised without sacrificing any of the advantages in using the PC architecture. The PC/104 was developed to address this need. It offers full compatibility with the PC architecture, but in a more compact form factor. The issue of a bulky backbone was addressed by producing stackable modules which "piggy back" on one another instead of slotting into a backbone [46].

The PC/104 form factor has produced sub-form factors such as PC/104-Plus, PCI-104, PCI/104-Express, EBX, EPIC and EPIC-Express to address the need of more PC architectures. PC/104 cards exist with DSP-processors and/or FPGAs incorporated onto the board, all of which runs on a standardized PC-bus. This makes the PC/104 a viable option. Figure 2-8 illustrates PC/104 modules stacking on top of one-another [47].

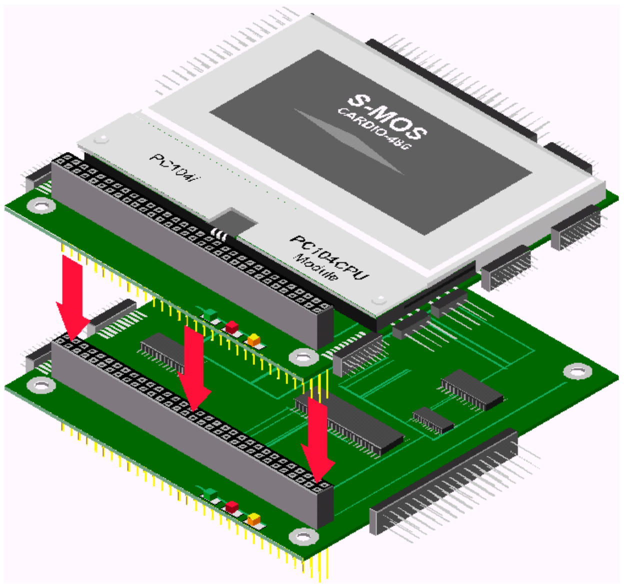


Figure 2-8: Horizontally stacked PC/104 modules

PMC DSP module

The peripheral component interconnect (PCI) is a high speed local bus. It is used by a large variety of processors for communication between components. This communication normally takes place over a backbone. The PCI specification defines a 106.68mm by 312.42mm board that slots perpendicularly into a motherboard [48]. The disadvantage of this method is that these perpendicular boards are not usable in low profile computer applications. Figure 2-9 displays the PCI slots on the motherboard of a desktop PC.

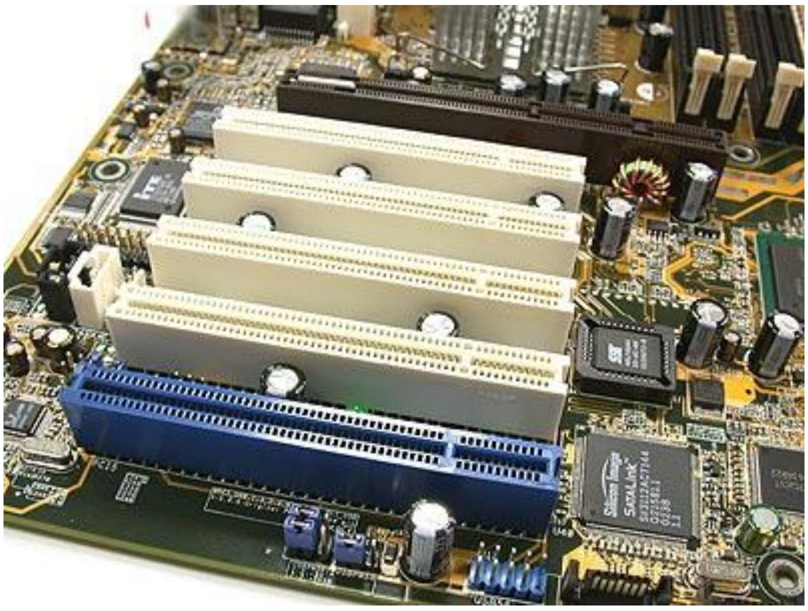


Figure 2-9: PCI slots on a PC motherboard

A slim, modular, parallel mezzanine card was created as a more compact solution to the perpendicular boards where slim, parallel board mounting is required, such as in a single board computer (SBC) solution. These mezzanine cards use the logical and electrical layers of the PCI local bus and are called a PCI mezzanine cards or PMCs. When compared to Figure 2-9, a PMC would slot into similar slot (only smaller) but whereas a traditional PCI card is perpendicular to the motherboard, a PMC is parallel to the board hosting the PMC slot.

An adaptation of the PCI standard, CompactPCI (cPCI), is an industrial version of the PCI standard. It offers a more robust mechanical form factor than PCI and is optimized to function in a more rugged environment [48]. Electrically it utilizes the same standard as PCI and is therefore compatible with the PCI standard.

Just as with the PC/104 form factor, PMCs are available in DSP and/or FPGA modules.

2.1.6 Design considerations

This section discusses the design considerations. The options for realizing the main controller unit of the ADES discussed in the preceding sections should be evaluated against the design considerations in this section to ensure that best possible solution is chosen.

Processor selection

An embedded system is a system targeted at performing a specific task on a specific platform. This limits the tasks of the processor in the embedded system to a specific range. If a correct processor or controller is chosen, it will be possible to optimize the system to perform its specified task(s) efficiently. Although the selection of a controller is a daunting task due to the sheer amount of controllers and processors in the marketplace, the final selection must pass four critical tests [49]:

1. Is it available in a suitable implementation?
2. Is it capable of sufficient performance?
3. Is it supported by a suitable operating system?
4. Is it supported by appropriate and adequate tools?

Each of these tests will now be discussed.

1. In some cases, a commercial system is available with enough processing power to perform the necessary functions and/or calculations of the project. This could reduce the cost of the final system since the hardware development of a system can be very high, thus forcing the choice of a COTS system.

2. This could possibly be the most important test a controller or processor should pass. If a controller is not capable of sufficient performance, there is absolutely no need to purchase such a controller. A controller should be able to execute a process in sufficient time not to inhibit the rest of the system performance. A real-time system should have a controller capable of performing tasks in real-time and a bulk processor should be capable of performing calculations in sufficient time on the bulk of data it receives. As embedded systems become more and more complex, it becomes more difficult to define the tasks a controller should perform and it becomes more difficult to perform the task on time. These tasks could create bottlenecks which limits performance. These tasks should thus be managed and a controller with a internal architecture should be chosen to minimize these bottlenecks.

3. Most of today's microprocessors can be equipped witch real-time operating systems (RTOS) which can be used to simplify the scheduling of real-time tasks. It is easy to see that the use of a RTOS can have obvious advantages. An RTOS can be come kind of Linux, a real-time Linux, third party real-time operating system (OS), or a

vendor created OS. Examples of RTOSs are QNX, RTLinux, VxWorks and Windows CE.

An RTOS can give added advantages in the system. The decision to use an RTOS, or not, as well as which RTOS to use is not a simple decision. Deciding to use an RTOS can induce a steep learning curve to learn the RTOS. This, together with the initial complexity of the system, can induce unnecessary complexity in the system.

Although RTOSs are mostly used in PowerPC architectures, it can also be used in a SBC solution instead of a full operating system. RTOSs can be used where scheduling of tasks are needed, where speed is of the essence and where hardware specifications are limited.

4. The quality and availability of a vendor's tools can determine the success of a project. The types of tools needed for a specific project is also dependent on the type of project. The minimum tools required for a software/firmware project are a good compiler and good debugging support. Emulators and simulators might also be needed.

Although these tests are a good indication on what to look for in selecting a processor, the optimal fit to these criteria is not necessarily the best selection. The final decision in controller also depends on the cost of the project, the time-to-market, reliability and previous experience.

Co-processing

One of the main considerations when selecting a controller is future expandability. It should always be kept in mind that the end product might need expansions in the future. What happens when the selected controller is unable to cope with the expansions? The first way around this problem is to increase the clock speed. This will allow the processor to execute instructions and calculations faster, but what happens when the maximum clock rate is reached? The next option is to replace the controller with a faster one, should one be available in the same architecture, size and socket.

Another option is to add another processor/controller. This implies additional or redesign of the hardware to accommodate the additional controller/processor. This adds additional cost to the project and will require software re-design since the original software was designed and developed for single controller architecture. Even when a multi-core processor is used to improve or expand performance, the code will still have to be re-designed to function on multiple cores in order to obtain additional performance.

Another problem could be that the selected controller is limited to certain functionality. A DSP is mostly limited to signal processing, has limited parallel processing capability and is limited in terms of functions such as timers, analogue to digital converters (ADCs), pulse width modulation (PWM) etc. What happens if a certain function is required which the DSP is unable to perform?

A way to circumvent these problems is to use an FPGA as a co-processor to the DSP [50]. This allows the FPGA to act as an extension to the DSP. Since FPGA are capable of parallel processing it adds extreme processing power to the DSP. The advantage of this approach above the multi-processor or multi-core processor is that there is no need to re-design the code to function on multiple cores. It is only necessary to transfer a certain part of the code to the FPGA for execution since the FPGA accepts the code as-is [50].

This architectural setup also increases the modularity of the system due to the fact that an FPGA is a versatile, configurable device. Whereas the DSP is limited to certain functionality, the FPGA can be programmed to perform virtually any function.

Benchmarking

What is a benchmark? According to Weiss [51], "benchmarking is a method of measuring performance against a standard, or given set of standards." He also states that these "standards" originate over time or is purposely developed for a specific application.

Benchmarking a controller or processor, or even when just comparing two controllers, proposes a few issues. For many engineers and scientists, benchmarking is synonymous with million instructions per second (MIPS), Dhrystones and Whetstones. As more and more floating-point and complex processors saw the light of day, benchmarks such as floating-point operations per second (FLOPS) and multiply accumulation cycles per second (MACS) were created. This section is dedicated to discussing the various different benchmarks available for processors.

Million Instructions per Second (MIPS)

If a processor (*A*) benchmarks against another processor (*B*) and processor *A* benchmarks at 2 MIPS while processor *B* benchmarks at half the MIPS of *A*, it can be considered that processor *A* is a better choice than processor *B* because it is twice as fast as processor *B*. This assumption is not necessarily true, unless both processors have the same internal architecture.

The term "MIPS" was originally defined in terms of the VAX 11/780 minicomputer. The VAX was a range of computers released in the 70's, and the VAX 11/780 minicomputer was the first machine that could run one million fixed-point instructions per second. This minicomputer was thus defined as a 1 MIPS machine [49]. The instructions used in these machines were one-dimensional metrics and had no reference to the internal architectures of the different VAX machines.

The MIPS benchmark is used today as an indication to how many times faster a certain processor or controller can execute fixed-point instructions than the original VAX machine. A processor with a benchmark of 1800 MIPS implies that the processor is 1800 times faster than the original VAX machine. This, however, is in the ideal case in a specific setup. Modern benchmarking techniques are optimized to reach extremely high MIPS (or FLOPS etc. as discussed later). This could either be done by benchmarking the processor executing simple instructions or by benchmarking the processor while it executes instructions its internal architecture was optimized to do.

The problem with this approach of benchmarking is that the ideal case does not apply to real world applications. Another aspect to consider is that RISC (reduced instruction set computer) and CISC (complex instruction set computer) machines differ in the way they execute instructions. So, although MIPS are a good indication of the processing speed of a controller/processor, unless two VAX computers are compared, MIPS is not a good choice of benchmark.

Dhrystones and Whetstones

The Dhrystone benchmark was created to overcome the shortcomings of the MIPS benchmark. It is based on a simple C-program that, when compiled, is about 2000 lines of assembly code. The 1 MIPS VAX 11/70 machine discussed previously was benchmarked using the Dhrystone and it was determined that the VAX machine could execute 1757 loops in one second. 1757 thus became 1 Dhrystone [49].

Since Dhrystones is a benchmark for fixed-point processors and controllers, the name “Dhrystone” was chosen as a parody to the floating-point benchmark of that time called the Whetstone (originally developed in Whetstone, Leicestershire in England). The current version on Dhrystone is version 2.1, which was developed in 1988 [51].

The problem with the Dhrystone benchmark is that it does not take into account the following [51]:

- RISC computing
- Availability of sophisticated floating-point processor units inside processors
- Single instruction, multiple data (SIMD) vector processors inside the main processor
- Superscalar RISC designs (multiple execution inside a single processor)
- Very long instruction word (VLIW) processors
- Optimizing controllers
- Large memory subsystems, including processors and subsystems with level 1, 2 and 3 (L1, L2 and L3 respectively) caches

- RTOS with sophisticated application programming interfaces (APIs), transmission control protocol/internet protocol (TCP/IP) functionality and graphical user interfaces (GUIs)
- Graphics, multimedia and communication-intensive applications

The same applies to the Dhrystone (and Whetstone) benchmarks as with MIPS: it is possible for a good programmer to optimize code to blast through the code, reaching a high Dhrystone (or Whetstone) benchmark, which in real-world applications will never be reached. The shortcomings of these Benchmarks are thus evident and another, better way of determining the performance of a processor or controller should be found.

EDN Embedded Microprocessor Benchmark Consortium (EEMBC)

To address the short-comings of the previously discussed benchmarks, the EDN Embedded Microprocessor Benchmark Consortium, or EEMBC[3] was founded. The difference between the EEMBC benchmark and the benchmarks discussed previously is that the EEMBC is based on industry-specific tests. These tests are divided into specific suites such as [49]:

1. Automotive/Industrial
2. Consumer
3. Networking
4. Office Automation
5. Telecommunications

As already mentioned, each of these suites is subdivided into numerous tests relevant to the application. For example, the automotive/industrial suite is divided into:

- Angle-to-time conversion
- Basic floating-point
- Bit manipulation
- Inverse discrete cosine transform
- Inverse Fast-Fourier transform (FFT) filter
- Matrix arithmetic
- Cache buster
- Pointer chasing
- Control access network (CAN) remote data request
- PWM modulation
- FFT
- Road speed calculation
- Finite impulse response (FIR) filter

[3] Pronounced “embassy”

- Table lookup and interpolation
- Tooth-to-spark calculation

The EEMBC benchmark uses statistics on the number of times per second the algorithm executes and the size of the compiled code and since the compiler used can optimize code to be executed – which could influence the benchmark, each suite of the EEMBC benchmark contains information on the compiler used.

The main disadvantage of using the EEMBC benchmark is that each suite should be purchased per application and most manufactures of processors and controllers see this as an unnecessary expense. The result is that the EEMBC benchmark for most of the processors in consideration is unavailable, making EEMBC an inefficient parameter for comparing controllers or processors.

Multiply Accumulate Cycles per Second (MACS)

Although MACS are not a benchmark in the true sense of the word, MACS are normally used in conjunction with another benchmark (mostly MIPS or FLOPS) to indicate the number of accumulation and multiplication cycles a processor or controller can execute in one second.

Since DSP operations mostly contain multiplication followed by a summation, most processors and controllers capable of DSP operations contain MAC units. These MAC units are capable of performing a certain amount of multiplications and additions in one cycle. The more cycles the processor or controller can execute in one second, the higher the MACS. A MAC instruction is given by the following equation:

$$x = a_1 \times b_1 + a_2 \times b_2 + a_3 \times b_3 + \cdots + a_n \times b_n$$

Floating-point operations per second (FLOPS)

FLOPS are equivalent to MIPS, but whereas MIPS are mostly used to measure the performance of a fixed-point processor or controller, FLOPS are used to measure the performance of a floating-point processor or controller. For the same reasons as MIPS, FLOPS are also an ineffective method of determining the performance of a processor or controller.

Operations per second (OPS)

An operation per second, or OPS, is simply an indication of the number of operations a processor can execute in a second. These operations could either be fixed or floating-point operations. Just as with MIPS and FLOPS, this benchmark is dependent on the internal architecture of the specific controller/processor and does not give an accurate measure to compare controllers/processors.

Berkeley design technology, Inc (BDTI)

Berkeley design technology, Inc. (BDTI) is a company founded in 1991 and provides resources for [52]:

- Independent benchmarking and competitive analysis
- Advice on creating credible, compelling marketing
- Guidance for confident technology and business decisions
- Expert product development advice
- Seminars and publications on technologies and trends

BDTI realized the shortcomings of other benchmarking techniques and created the BDTImark™ benchmarking technique in 1997. The BDTImark™ is a composite signal processing speed metric based on BDTI's suite of signal processing algorithm kernel benchmarks called the BDTI Benchmarks™ [53]. This suite as well as the BDTImark™ was updated in 1999 to better reflect more modern signal processing applications. The updated BDTImark™ was called the BDTImark2000™. For a list on the functions included by the BDTImark™ (and the BDTImark2000™) refer to Table A-1 in the Appendix.

Most of BDTI's benchmarking scores are verified on actual silicon chips, but in the cases where it is not feasible or possible to verify the scores on actual silicon, they have created the BDTIsimMark2000™. This is equivalent to the BDTImark2000™, but is only verified in simulation. Care should be taken when comparing the BDTIsimMark2000™ to the BDTImark2000™ since the speed on the actual silicon may vary and depends on the fabrication of the chip.

The BDTI benchmarking technique has a few limitations [53]:

- The BDTImark2000™ is a composite metric based on different signal processing algorithms. It is thus only an indication of signal processing speed and not for a specific application
- A processors aptitude for a specific application (such as co-processing) is not taken into consideration by the BDTImark2000™
- The BDTImark2000™ is only an indication of signal processing. Other factors such as power consumption, price and on-chip integration are not taken into consideration
- It is based on a processors native arithmetic format. Fixed-point processors should thus be compared to fixed-point processors and vice versa

Despite the limitations of the BDTImark2000™, many companies have adopted the BDTImark2000™ as another method of benchmarking their processors and/or controllers.

Refer to Figure A-1: BDTI speed per dollar ratios for floating-point processors and Figure A-2 in the Appendix for the BDTImark2000™ of the TigerSHARC™ and TMS320C67x.

For instance, the BDTImark2000 for the TigerSHARC™ and TMS320C67x is available at [28]. More specific BDTImark2000™ tests and scores are also available should the developing companies feel the need to benchmark their processors and controllers in other areas as well (refer toFigure A-3 in the Appendix for an example how speed per millwatt is benchmarked).

2.2 Active magnetic bearing control

Since control of an active magnetic bearing (AMB) constitutes a major part of this project, this section is dedicated to discussing control of an AMB.

Figure 2-10 shows a control diagram for a one degree of freedom (DOF) AMB [55]. A sensor measures the position of the rotor between the AMBs. This position value is converted into a current value which is subtracted from a reference value. This reference value is the required position of the rotor and the difference between the reference and actual value gives an error value. This is the value used by the main controller in a position control algorithm to determine the current reference (i_{ref}). This current reference is then sent to the two power amplifiers which convert the current reference into a pulse width modulation (PWM) signal to drive the electromagnetic actuators.
The position control algorithm typically used in AMB control is proportional plus integral plus derivative (PID) control. The subsequent section is dedicated to discussing PID control.

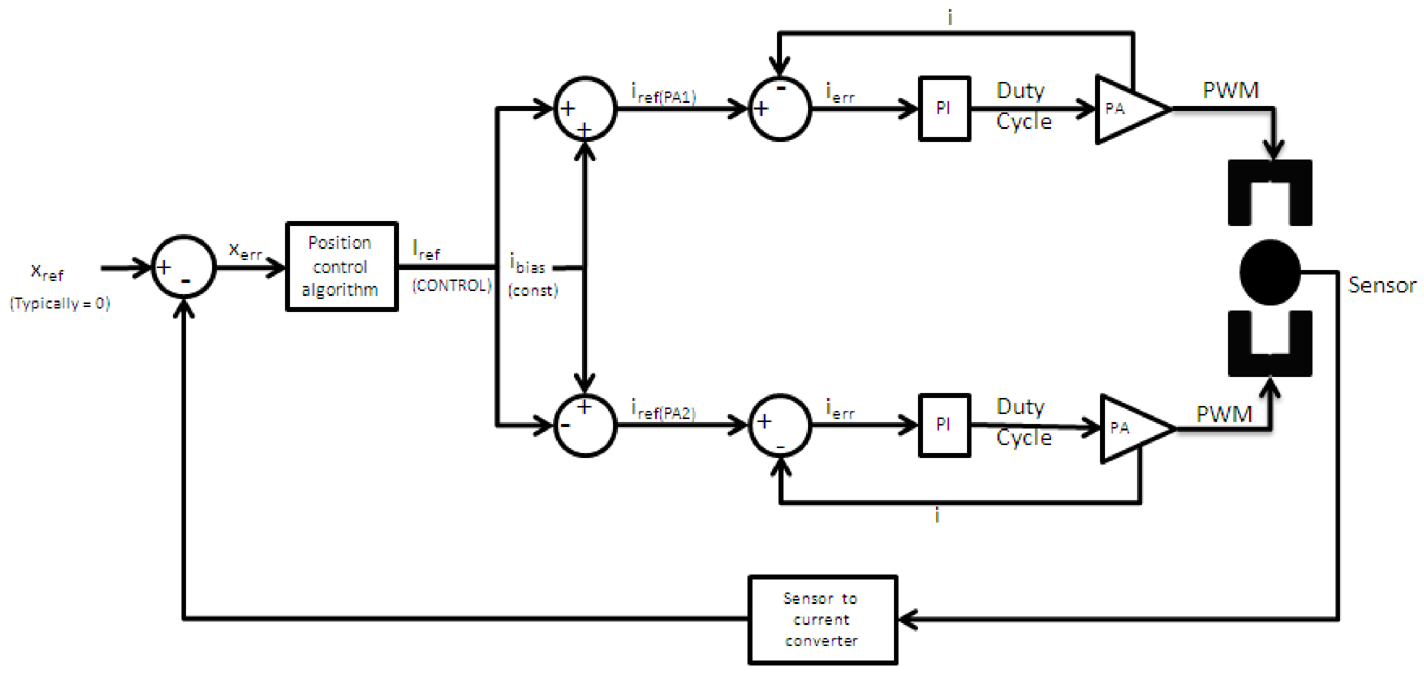


Figure 2-10: Control diagram for the suspension of a rotor in one degree of freedom

2.2.1 PID controllers

The block diagram of a discrete (thus digital) PID controller can be seen in Figure 2-11. The transfer function of a PID controller is [56]:

$$D\ \ w\ = K_P + \frac{K_I}{w} + K_D w \tag{2.1}$$

with K_P the gain in the proportional path, K_I the gain in the integral path and K_D the gain in the derivative path.

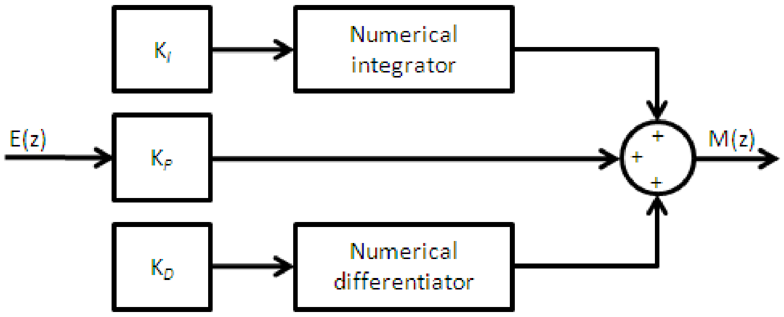


Figure 2-11: Discrete PID controller block diagram

Since ω is a function of frequency, it is easy to see that the gain of the PID controller increases as frequency increase and is thus not practical. To circumvent this problem, a pole, or poles, must be added to the derivative term. This results in the following transfer function [56]:

$$D\ \ w\ = K_P + \frac{K_I}{w} + \frac{K_D w}{\prod(1+\frac{w}{\omega_{wpi}})} \tag{2.2}$$

with ω_{wpi} the pole added at a certain frequency. The norm, however, is only to add a single pole, thus giving:

$$D\ \ w\ = K_P + \frac{K_I}{w} + \frac{K_D w}{(1+\frac{w}{\omega_{wp}})} \tag{2.3}$$

with ω_{wp} (the pole) chosen at:

$$\omega_{wp} = -\frac{2}{T} = -\frac{\omega_s}{\pi} \tag{2.4}$$

with T the sampling interval and $\omega_s = \frac{2\pi}{T}$, or the sampling frequency. This pole is chosen out of the system bandwidth and will have no effect on the system response.

A practical PID transfer function can then be obtained by transforming equation (2.3) to the z-plane. The resulting equation is then:

$$D\;\;w\;\;=K_P+K_I\frac{T}{2}\left[\frac{z+1}{z-1}\right]+K_D\left[\frac{z-1}{Tz}\right] \tag{2.5}$$

The first practical method of employing a digital PID controller is by using numerical integration and differentiation.

Equation (2.3) can also be written as a second order transfer function (2.6) and be implemented using several second order implementation methods [56]. One method is shown in Figure 2-12 [56].

$$\frac{\left(K_P+\frac{K_IT}{2}+\frac{K_D}{T}\right)+\left(-K_P+\frac{K_IT}{2}-\frac{2K_D}{T}\right)z^{-1}+\left(\frac{K_D}{T}\right)z^{-2}}{1+\;\;-1\;\;z^{-1}} \tag{2.6}$$

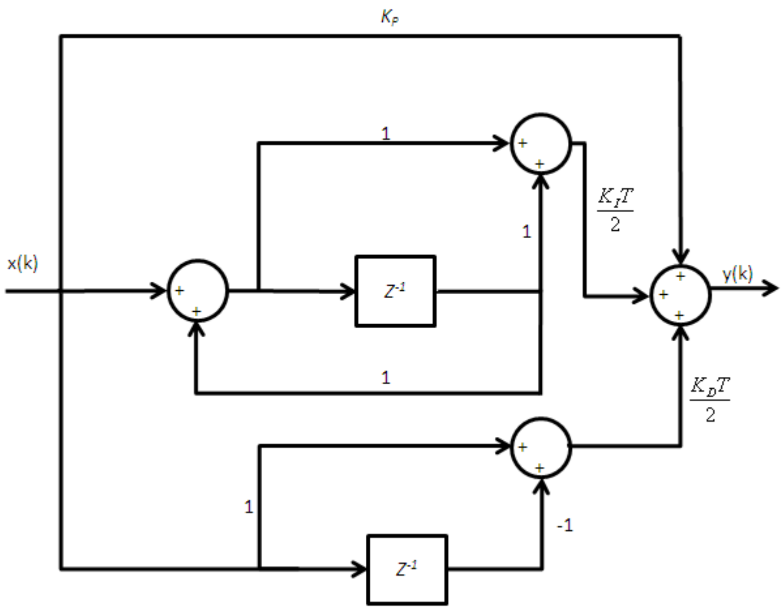


Figure 2-12: PID controller

2.3 Digital filters

Since digital filters will play a major role in this project, it was deemed necessary to discuss digital filter concepts.

According to [22], “a filter is essentially a system or network that selectively changes the wave shape, amplitude-frequency and/or phase-frequency characteristics of a signal in a desired manner”. The changing of signal characteristics could either be to improve quality, to reduce the noise, to remove a certain frequency (or frequencies) or to extract certain information from the signal. Whereas analogue filters normally consists of active components (such as operational amplifiers (op-amps)), or passive components (such as resistors, capacitors and/or inductors), digital filters are mathematical algorithms implemented in either hardware and/or software. These digital filters receive a digital signal and produce a filtered digital output.

Digital filters have the following advantages [22]:

- Digital filters have certain characteristics which are not possible with analogue filters
- The performance of digital filters does not vary with environmental changes
- The response of digital filters can easily be adjusted if the filter is implemented in software
- A single digital filter can take multiple inputs and can be used to filter several signals without replacing hardware
- Filtered and unfiltered data can be saved for future use
- Advances in very-large-scale integration (VLSI) can be used to create small, low power, cheap filters
- High precision is achievable
- Digital filters have repeatable performance
- Can be used at low frequencies

Digital filters also have certain disadvantages [22]:

- There is a limitation on the speed digital filters can handle. The maximum bandwidth digital filters can handle is lower than that of analogue filters. The speed of the digital filter is limited by the ADC and DAC speed as well as the processing speed of the microcontroller/-processor.
- Quantization of a continuous signal induces noise. Digital filters are subject to this noise.
- The design and development time of a digital filter can be longer than that of an analogue filter.

Two types of digital filters exist [22]: finite impulse response (FIR) and infinite impulse response (IIR). Each of these filters is a product of their respective impulse response sequence. This is depicted in Figure 2-13.

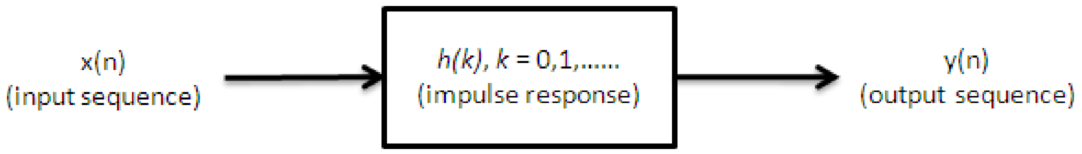


Figure 2-13: Conceptual representation of a digital filter

The output of the digital filter is a convolution sum. This can be seen from the following equations [22]:

$$y\ \ n\ = \sum_{k=0}^{\infty} h\ \ k\ \ x(n-k) \tag{2.7}$$

$$y\ \ n\ = \sum_{k=0}^{N-1} h\ \ k\ \ x(n-k) \tag{2.8}$$

Equation (2.7) represents the IIR filter and (2.8) represents the FIR filter, with *h(k)* as defined in Figure 2-13. As can be seen, the IIR filter's response is calculated until infinity. In practice, however, this is unfeasible. Instead, the IIR filter's response is rather calculated in recursive form [22]:

$$y\ \ n\ = \sum_{k=0}^{\infty} h\ \ k\ \ x(n-k) = \sum_{k=0}^{N} b_k x\ \ n-k\ \ - \sum_{k=1}^{M} a_k y(n-k) \tag{2.9}$$

where a_k and b_k are the IIR filter coefficients. These values, together with *h(k)*, are essential parameters in digital filter design.

Another crucial difference between IIR and FIR filters is IIR filters (according to (2.9)) are a function of previous outputs as well as present and past inputs, whereas the output sample of FIR filters (according to (2.7)) are only a function of present and past input values.

The choice between an FIR and IIR depends on [22]:

- FIR filters are capable of having an exact linear phase response
- FIR filters are more stable since they are realized non-recursively
- Round-off noise and coefficient quantization errors are reduced when a limited number of bits are used to implement a filter such as in a FIR filter
- FIR filters require more coefficients than IIR to achieve a sharp cut-off
- Unlike FIR filters, IIR filters have an analogue counterpart which makes it easy to transform analogue filters into equivalent IIR filters
- It is simple to synthesize filters of arbitrary response with FIR filters
- FIR filters are algebraically more difficult to synthesize without computer aided design (CAD) tools

2.4 Object orientated embedded C

One of the key aspects of good software engineering is the ability to write robust, reusable code which can easily be reproduced [57], [58]. In an object-orientated environment such as C++, objects are represented as classes. These classes consist of attributes and behaviours which make part of the function's variables and methods. Object-orientated programming for embedded systems is limited by intensive platform dependent hardware as well as real-time specifications. This is due to the fact that the most popular programming language for embedded systems, C, is not inherently object-orientated, regularly rely on global variables and lacks classes and inheritance [57].

Neser proposes an object-orientated approach to ANS-C in [57], which can be followed for real-time embedded applications. By using this approach in this project, the code will be made reusable throughout the development process. Another key feature of using this approach is maintainability and modularity of the code, since future improvement and expansion of the code is imminent.

An object consists of data in a set of member variables and methods, which is relevant of the data. Although objects and classes are not part of the C programming language, this functionality can be emulated by using structures and methods could be emulated by using function pointers. Classes in C are thus defined as structures with all their member variables and function pointers to all their methods. Each class definition and declaration is done in their own respective header and C-files, named according to the class name.

As an example, the following example of a class in C is given:

```
// Example: Class in C

#include “predef.h”

#ifndef example_h
#define example_h

typedef struct
{
        Int IntVariable ;
        float FloatVariable ;
}



#endif
```

Figure 2-14: Example: Embedded C-class

2.5 Critical evaluation of literature

The literature survey done was to get a general overview on the discussed subjects. Although most of the microcontrollers and microprocessors discussed in section 2.1.1 and 2.1.2 are predominately used in personal computers (PCs), it is invaluable to understanding the concepts “microcontroller” and “microprocessor”. The aim of this dissertation is to select an embedded controller for an AMB and drive electronics system; thus implying an embedded unit to perform control. This control can be performed using a microcontroller and/or microprocessor. Since the line between microcontrollers and microprocessors are fading as more and more processors are acquiring functions normally included in microcontrollers and microcontrollers are becoming increasingly more powerful, it was necessary to define the terms “controller” and “processor” from the literature; a “microcontroller” is a device that performs control, whereas a “microprocessor” is a device that performs processing.

A wide variety of processors and controllers were discussed in this chapter. In general, it can be divided into two sections: standalone and as part of an off-the-shelf system. It was also investigated as such, since the off-the-shelf systems contain the standalone controllers investigated. The standalone controllers investigated are all under consideration for realizing the main controller unit of the ADES. According to specification, the selected controller should be capable of performing floating-point calculations, thus all the controllers considered are capable of floating–point number representation.

The digital signal processors (DSPs) investigated are the SharC™ and TigerSHARC™ DSPs from Analog Devices®, as well the TMS320C67x-range DSPs from Texas Instruments®. The SharC™-range contain a variety of DSPs ranging from entry-level to high-end. The TigerSHARC™-processors are the most powerful processors in the SharC™-range. The TMS-range from TI® includes microcontrollers and microprocessors ranging from entry-level to high-end and even multimedia-processors. The reason why the TMS320C67x-range is considered is due to their floating-point capability.

Floating-point number representation is important due to the higher resolution obtained when using floating-point calculations. A higher resolution will lead to better control of the system. Even though a good resolution can be obtained using fixed-point number representation by scaling the floating-point number to retain the decimal numbers, it adds unnecessary complexity to the system. A floating-point processor is also capable of handling true floating-point numbers and calculations. In a fixed-point processor, this process should either be emulated, or the floating-point number should be interpreted as two fixed-point values. This also adds unnecessary complexity and overhead, which inhibits the performance of the processor.

It was mentioned that a processor should pass the following tests in order to determine the best controller for the main controller unit:

1. **Is it available in a suitable implementation?**
   Two implementation options are available: designing a main controller in-house using the best controller from the options listed in this chapter, or as an off-the-shelf component in PC/104 or cPCI form factor.

2. **Is it capable of sufficient performance?**
   Quantifying “sufficient performance” is difficult. As mentioned, performance of a microcontroller or microprocessor not only depends on the number of MIPS, MACS, FLOPS etc., it also depends on the internal architecture of the integrated circuit (IC), the application in which it is used and the instructions executed. An efficient processor must be selected for the ADES control. Due to the sheer complexity of benchmarking a controller or processor and even knowing beforehand how powerful a processor is, it is important to know how the different developers benchmark their product. Developers can give processor/controller performance in MIPS, FLOPS, MACS, DMIPS, OPS or BDTI mark. These concepts were discussed in this chapter to better understand these concepts.

3. **It is supported by a suitable operating system?**
   Some processors discussed in this chapter are capable of handling an operating system. This might lead to an advantage in certain architectures. This should be kept in mind when a processor is selected.

4. **Is it supported by appropriate and adequate tools?**
   Although this is not the most important factor in selecting a processor, the adequacy (or lack there-of) can have an enormous influence on the development time, difficulty and success of the system.

After determining the most suitable controller for controlling the ADES, the control should be implemented on the selected controller. This includes the filter(s) to remove noise from the control-band as well as the PID control. All the components in the control loop will be implemented digitally. Object orientated embedded C will provide a robust and reusable implementation of the ADES control. The aim of the ADES is to develop a system to be used in an industrial environment. Robustness of the written code is thus of great importance. It might also be necessary to improve or expand the written code. The object orientated approach will ensure efficient hand-over of the code to another programmer.

Now that the literature essential for the completion of this dissertation was discussed, focus can now be shifted to selecting a sufficient controller.

CHAPTER 3

Controller hardware selection

This chapter discusses the process of selecting a suitable controller based on the requirements of the ADES. The chapter therefore starts off by first giving an overview of the general ADES system and then continues by giving the requirements of the main controller unit. The focus is then shifted to the process of assigning functions to the main controller unit and then to the selection process.

3.1 System specification

As stated, the main controller unit's specification is related to the ADES system specification, therefore, a quick system specification of the ADES is given. The complete ADES system requirement specification is available on the data disc. Refer to Appendix F for the location of the file.

3.1.1 ADE system introduction

As discussed in chapter 1, the primary intent of the ADES is to replace the dSPACE® system (seen in Figure 1-3) with a compact, reliable and low cost system which is on an industrial standard. The proposed system is shown yet again in Figure 3-1 for clarity.

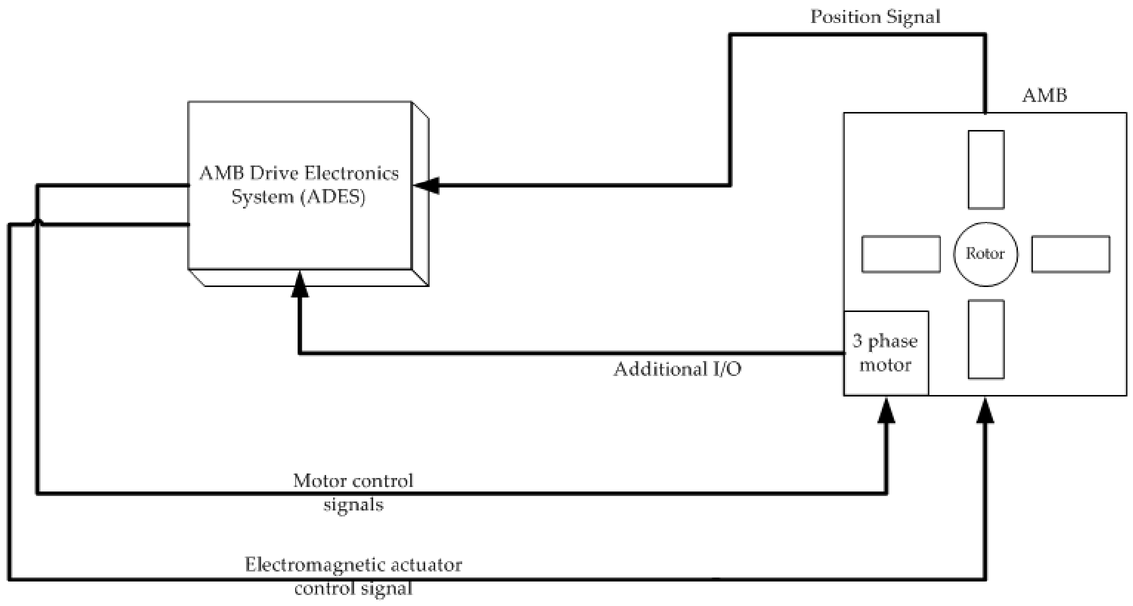

Position Signal
AMB
AMB Drive Electronics System (ADES)
Rotor
3 phase motor
Additional I/O
Motor control signals
Electromagnetic actuator control signal

Figure 3-1: Proposed system

Although the ADES comprises of individual sub-components, the focus of this dissertation is on the main controller unit seen in Figure 1-5. The ADES with its functional units are shown in Figure 3-2.

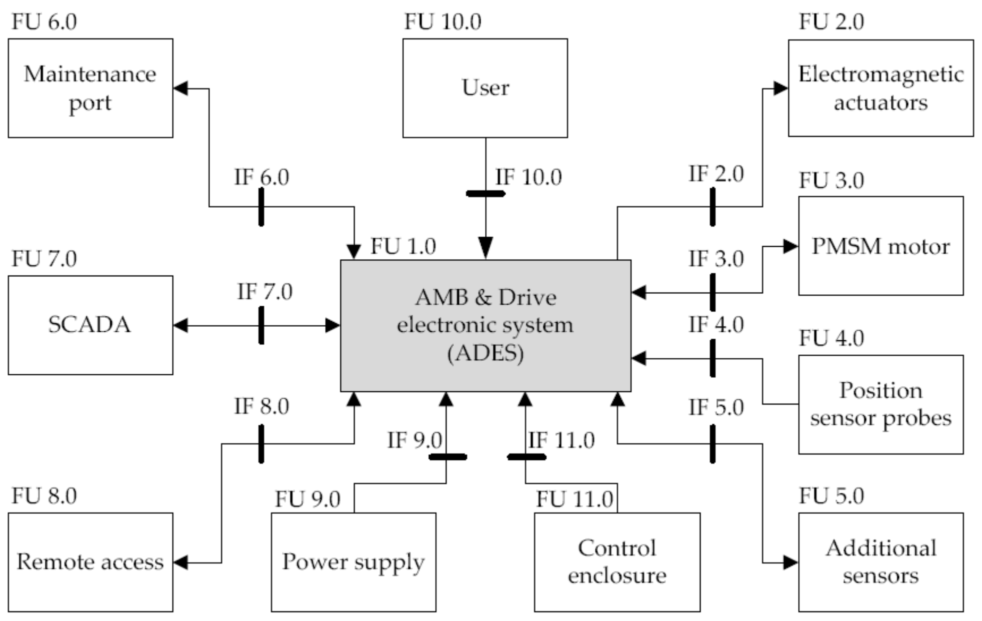


Figure 3-2: Functional units of the ADES

3.1.2 Embedded controller system specification

As given and discussed in the system requirement specification, the main controller unit should be capable of performing the following functions:

- Controlling the AMBs (F1.1)
  - Calculate control response (F1.1.2)
- Monitoring system (F1.3)
  - Run self-diagnostic test (F1.3.1)
  - Monitor conditions (F1.3.2)
  - Sensitivity analysis (F1.3.3)
- Communicate (F1.4)
  - Transmit (F1.4.1)
  - Receive (F1.4.2)

Each of these functions will now be discussed:

Controlling the AMBs

The primary function of the main controller unit is controlling the AMBs. As seen in Figure 2-10, the AMBs are controlled by means of a current reference generated by a position control algorithm. It was decided that the position control algorithm for this project is PID control.

Two options exist for implementing the control diagram: closing the current and position control loops on the main controller unit, or moving the current loop to the power amplifiers. The second option of implementation was selected, since this will improve communication speed due to less data needed to be transferred between components. Figure 2-10 can be divided into three sections, seen in Figure 3-3:

1. Main controller section - which is responsible for generating the current reference by means of the PID algorithm
2. Power Amplifier 1 section - receives the current reference and generates a pulse width modulation (PWM) signal by using a proportional-integral (PI) control algorithm
3. Power Amplifier 2 section – which inherently performs the same functionality as power amplifier 1, only in the opposite direction

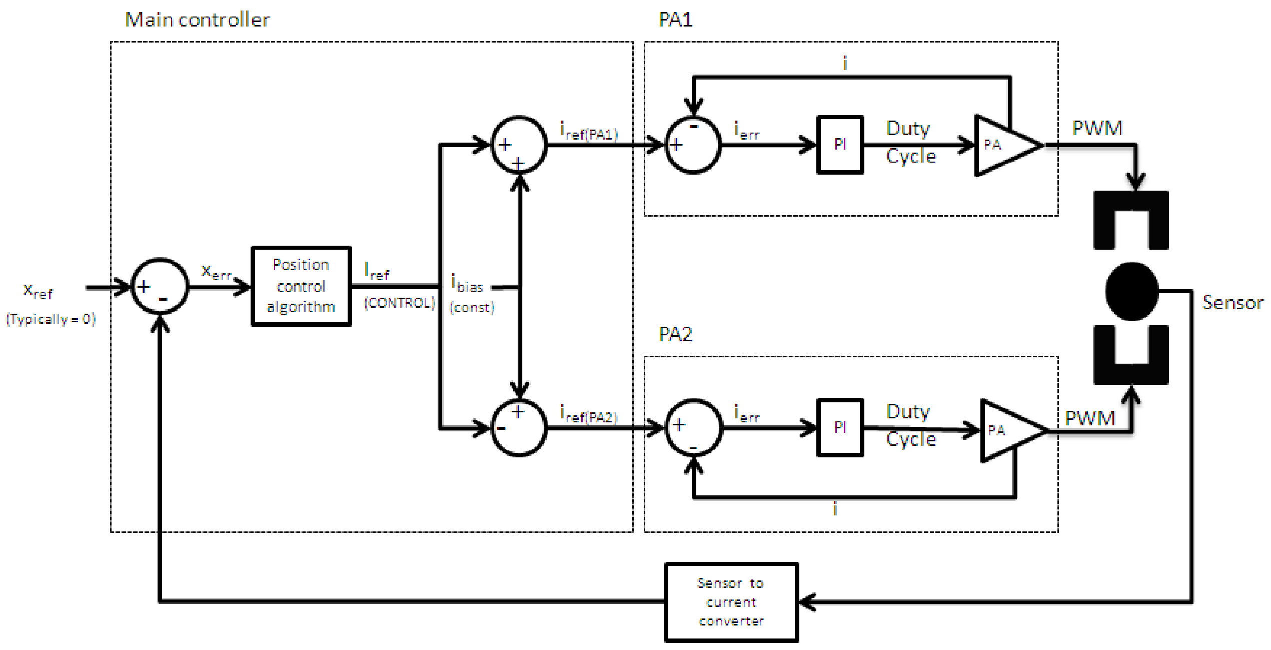


Figure 3-3: Control diagram divided into three sections

Monitoring the system

The main controller should be capable of monitoring the system. This implies performing a self-diagnostic test at start-up, performing condition monitoring and performing a sensitivity analysis at rotor standstill and at nominal speed. Although complex condition monitoring, such as monitoring by means of a neural network or statistical analysis, is out of scope for this dissertation, the selected controller should be capable of general condition monitoring. General condition monitoring includes monitoring the data sent and received and ensuring it is valid, logging data and/or displaying data on a user interface.

Communicate

High speed, reliable communication is required between the different components of the ADES. Despite the fact that a designated communication unit is responsible for the communication between the ADES components, the main controller unit should still be capable of receiving and interpreting data. It might also be necessary for inter-processor communication, should a processor and a co-processor be chosen as the main controller unit.

3.1.3 Specified ADES system states

Another important specification of the system is the states in which the system can reside. This is due to the fact that the system states influence the possible states of the main controller unit. The specified state diagram can be seen in

Figure 3-4. The transitions to and from each stage is also indicated in the figure. A brief description of each state is given:

1. **Power down**
   In this mode the ADES is disconnected from the power supply and the controller is non-operational. The interrogation signal used in the sensitivity analysis is disabled and the motor speed is zero. Physical maintenance, such as replacements, may be performed in this state.

2. **Standby**
   The ADES is connected to the power supply and the controller is operational. The AMBs, however, are non-operational and both the interrogation and control signals are disabled. The motor speed is still zero.

3. **Programming**
   The power supply is on and the controller is in a configuration state where the firmware may be upgraded or altered. The AMBs are non-operational and the interrogation and control signals are disabled.

4. **AMBs operational**
   The power supply is on and the controller is operational. The AMBs are operational which implies that the rotor is levitated. The interrogation signal is disabled and motor speed is zero.

5. **Sensitivity analysis at standstill**
   This state flows from the previous state where both the controller and AMBs are operational. The interrogation signal is enabled and the sensitivity analysis is performed according to the ISO 14839 standard.

6. **Normal operation**
   In this mode the power supply is on, the AMBs are operational and the interrogation signal is disabled. The motor is running at nominal speed.

7. **Sensitivity analysis at nominal speed**
   This mode is used to perform sensitivity analysis on the system while the system is operational. This implies that the AMBs are operational and the motor is running at nominal speed.

8. **Maintenance**

Although this state is not included in Figure 3-4, a maintenance state is required to allow the user to perform maintenance on the system.

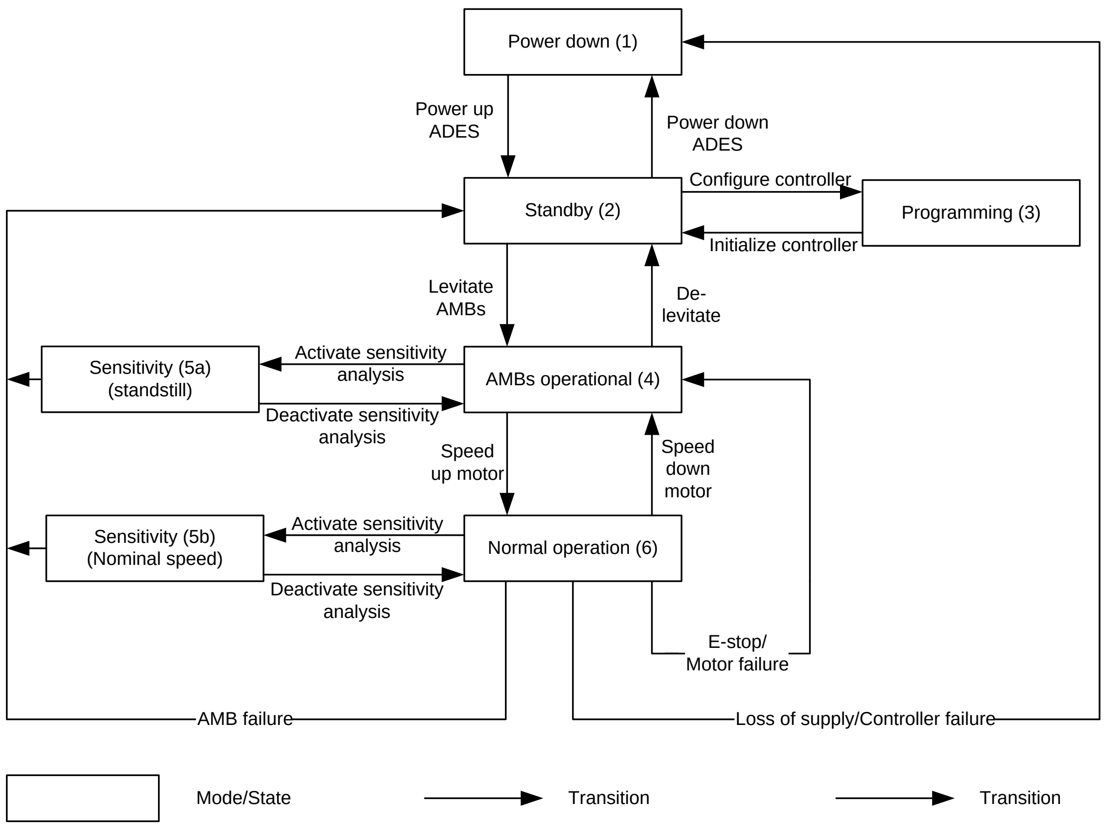


Figure 3-4: Specified system states and transitions[4]

[4] As specified in the system requirement specification, which can be found on the data disc

3.2 Control estimation calculations

Before it can be determined what processor to use in the system it is necessary to estimate the number of instructions required to preform specific tasks. This estimation is based on the requirements of previous projects as well as the number of instructions from a TMS320F/C24x DSP from Texas Instruments (used in a previous project to control an AMB system). The first section gives the requirements of the PID control.

3.2.1 PID control algorithm requirement estimation

The block diagram of a PID control algorithm implemented in a previous project can be seen in Figure 3-5.

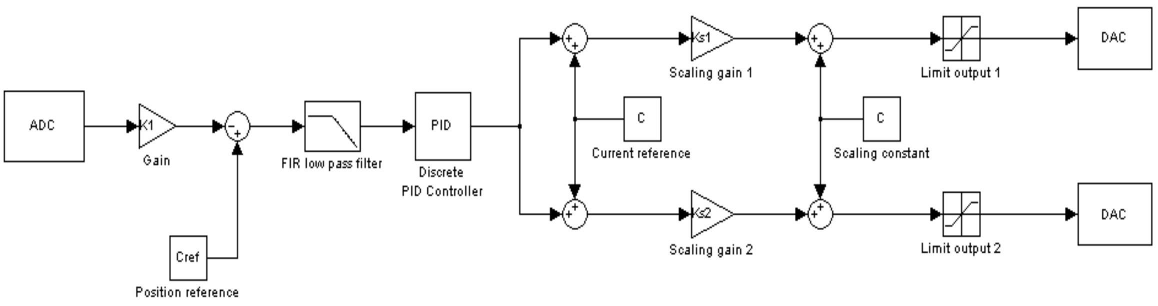


Figure 3-5: PID control block diagram

To implement the control algorithm on a DSP, the block diagram has to be simplified to:

$$\frac{\left(K_P + \frac{K_I T}{2} + \frac{K_D}{T}\right) + \left(-K_P + \frac{K_I T}{2} - \frac{2K_D}{T}\right) z^{-1} + \left(\frac{K_D}{T}\right) z^{-2}}{1 + \ -1 \ z^{-1} + 0} \tag{3.1}$$

which can be seen in Figure 3-6.

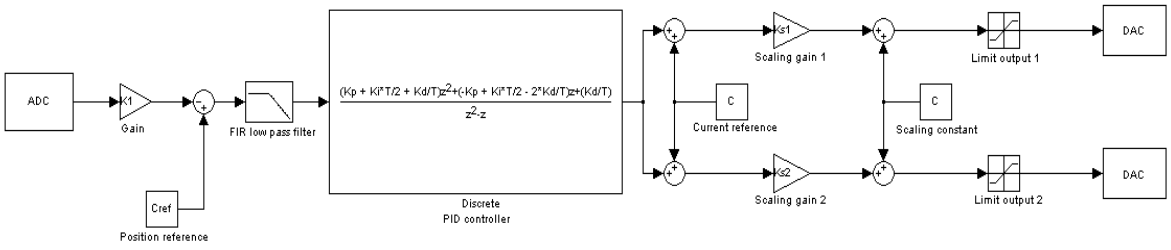


Figure 3-6: Simplified PID controller

This block diagram is then expanded into its sub-components and the number of cycles necessary to execute each block. This is calculated in Table 3-1 [59].

From the table it can be seen that the number of cycles required for filtering the input (by means of a low pass filter) and then to control the system by means of PID is estimated to be 1757 for one degree of freedom (DOF). For five degrees of freedom, this value should be multiplied by 5, thus $1757 \times 5 = 8785$. The chosen sample rate for the ADES system is 20 kHz. This implies that the system performs $1757 \times 5 \times 20 \times 10^{3} = 175.7 \approx 176 \text{ MFLOPS}$. Thus, in one second, the controller/processor should be capable of 176 million floating-point operations per second.

The control cycle of the proposed system is 50 µs. It was decided that the main controller should be capable of calculating the control response in roughly half the control cycle – thus 25 µs – to enable future expansions. Thus, the speed calculated above should be doubled to 352 MFLOPS.

Table 3-1: Estimated PID control requirements for one DOF

Instruction	**Mnemonic**	**No. of executions**	**Cycles**
FIR filter before PID (50th order, 50 samples)		1	1438
FIR filter after PID (50th order, 1 sample)		1	118
Read the ADC	IN	2	4
Multiply with scaling factor	MAC	1	3
Subtract reference	SUB	1	2
Calculate PID			
Summations	MAC	1	14
Divisions	SUBC	1	96
Multiplications	MAC	1	42
Add to i_{ref} for upper AMB	ADD	1	2
Subtract from i_{ref} for bottom AMB	SUB	1	2
Multiply scaling factor for top AMB	MAC	1	3
Multiply scaling factor for bottom AMB	MAC	1	3
Add biasing constant	ADD	1	4
Check range limit for upper AMB	BCND	1	8
Check range limit for lower AMB	BCND	1	8
Set limit	LAR	2	4
Write to DAC	OUT	2	6
		Total	1757

3.2.2 Estimated MIMO control requirements

At this stage it is also necessary to quantify the control response. Estimation to the number of accumulations per cycle and multiplications is shown in

Table 3-2. Also indicated are the cycle time, numerical accuracy and number representation for effective MIMO control. This is significant since the processor selected will be used in future iterations of this project and even though multiple input and multiple output (MIMO) control is not implemented during this iteration of the system, it will be investigated in the future.

Table 3-2: Control response estimation

Description	Specification
Cycle rate	20 kHz
Type	MIMO
Accumulations/cycle	304
Multiplications	484
Numerical accuracy	64-bit[5]
Number representation	Floating-point

According to the estimation given, the number of multiplications and accumulations per cycle for the controller is then calculated for a worst case as:

$$\frac{\text{Accumulations}}{\text{cycle}} + \frac{\text{Multiplications}}{\text{cycle}} = 304 + 484 = 788 \qquad (3.2)$$

The number of MACS is then:

$$\frac{788}{50\,\mu\text{s}} \times \frac{\frac{1}{50\,\mu\text{s}}}{\frac{1}{50\,\mu\text{s}}} = 1400\ \text{MACS} \qquad (3.3)$$

By leaving room for error, the total number of MACS is estimated as 1500 MACS.

[5] Recommended numerical accuracy. 32-bit is also sufficient.

3.2.3 Other estimated control requirements

This section discusses the other estimated control requirements such as:

- Monitoring conditions - Although complex condition monitoring is out of scope for this project, enough room on the processor should be provided to implement advanced condition monitoring at a later stage
- Sensitivity analysis
- Transmitting and receiving data

These estimations are based on diagrams and estimations on the number instructions it will take. Estimation on the number of instructions necessary to perform the abovementioned functions can be seen in the Table 3-3 and Table 3-4.

Table 3-3: Estimated number of instructions for monitoring conditions

F 1.3.2 Monitor conditions		
	F 1.3.2.1 Rotor position	25
	F 1.3.2.9 Cooling gas temperatures	10
	F 1.3.2.10 Rotor temperature	5
	F 1.3.2.11 Status signals	50
	F 1.3.2.4 Pressures and flow rates	50
	F 1.3.2.2 EM coil currents	50
	F 1.3.2.3 Rotational speed	10
	F 1.3.2.5 AMB coil temperatures	100
	F 1.3.2.7 PMSM winding temperatures	30
Total number of instructions		330

Table 3-4: Estimated number of instructions for sensitivity analysis

F 1.3.3 Sensitivity analysis		
	Sine wave function:	50
	Store data to memory:	10
Total number of instructions		60

Although the communication is not part of the scope of this dissertation, the communication still influences the specification on the main controller unit, if it is implemented on the same processor.

The estimated number of instructions can be seen in Table 3-5. The “?” in the estimation of the SCADA (supervisory control and data acquisition) and maintenance port are due to the number of unknowns for these two functions, which complicates the estimation to the number of clock cycles required.

The total number of instructions needed to be executed during one control cycle for monitoring the conditions, performing the sensitivity analysis and communicating between components are estimated as:

$$330+60+1925=2315 \tag{3.4}$$

It is thus, 2315 floating-point operations per cycle. Converting this number to FLOPS:

$$\text{Operations per cycle} = \frac{2315}{50\,\mu\text{s}} \tag{3.5}$$

$$\text{FLOPS} = \frac{2315}{50\,\mu\text{s}} \times \frac{\dfrac{1}{50\,\mu\text{s}}}{\dfrac{1}{50\,\mu\text{s}}} = 46,3\ \text{MFLOPS} \tag{3.6}$$

To compensate for any possible errors, this estimated number of FLOPS is rounded to 50 MFLOPS.

Table 3-5: Estimated number of instructions for communication

F1.4 Communicate				
	F 1.4.1 Transmit			
		F 1.4.1.18 Tx: current reference x10		
			Generate error code	500
			Pack data into buffers	50
		F 1.4.1.19 Tx: PMSM speed reference x1		
			Generate error code	50
			Pack data into buffers	5
		SCADA		?
		Maintenance port		?
	F 1.4.2 Receive			
		F 1.4.2.9 Rx: current values x10		
		F 1.4.2.10 Rx: status		
			Pack data from buffer	50
			Run error algorithm	500
			Update status registers	50
		F 1.4.2.11 Rx: Rotor position values x5		
		F 1.4.2.12 Rx: status of sensor driver unit		
			Pack data from buffer	25
			Run error algorithm	50
			Update status registers	5
		F 1.4.2.13 Rx: PMSM speed value x 1		
		F 1.4.2.14 Rx: status		
			Pack data from buffer	5
			Run error algorithm	50
			Update status registers	5
		F 1.4.2.15 Rx: Rotor temperature 1x		
		F 1.4.2.16 Rx: Cooling gas temperature 2x		
		F 1.4.2.17 Rx: AMB coil temperature 10x		
		F 1.4.2.18 Rx: PMSM winding temperatures 3x		
		F 1.4.2 19 Rx: Pressures and flow rates 10x		
		F 1.4.2.19 Rx: status 1x		
			Pack data from buffer	160
			Run error algorithm	50
			Update status registers	5
		SCADA		
		Maintenance port	Store data to DPR	25
			Retrieve Command data from DPR	50
			Analyse data	100
Total number of instructions for communication				1925

3.3 Estimated signal processing requirements

The total number of FLOPS for the processor/controller can then be estimated from section 3.2:

$$352\ \text{MFLOPS} + 50\ \text{MFLOPS} = 402\ \text{MFLOPS} \tag{3.7}$$

To compensate for errors in the calculations and to ensure that the processor adheres to the required speed, the number of FLOPS is multiplied by 2:

$$402\ \text{MFLOPS} \times 2 = 803\ \text{MFLOPS} \tag{3.8}$$

Since the control algorithm requires 803 MFLOPS of processing power, it is estimated that at least 1 GFLOPS of processing power is required to realize the ADES. The specifications of the processor can be seen in Table 3-6.

Table 3-6: Estimated processor specifications

Cycle rate	20 kHz
Number of MACS	1000
Numerical accuracy	32/64 bit
Number type	Floating-point
FLOPS	1 GFLOPS

3.4 Decision matrices

The selection of the main controller will be evaluated according to certain criteria. Each criterion will be given a weighted value. The higher the weighted value, the more important the criterion is.

This section gives and discusses the decision matrix, shown in Table 3-7. This matrix depicts the criteria and weight against which the system and its individual components should be measured. This matrix will be used throughout the selection process to perform trade-offs.

Table 3-7: System evaluation criteria

Criteria	Weight	Discussion
Robustness	0.2	Measurement of performance. An algorithm is robust if it continues to operate despite abnormalities in input, calculations, etc. Hardware is robust if is environmentally sound.
Efficiency	0.2	Measurement of performance. An indication to the ability to control the system and perform the required tasks.
Cost	0.3	This is the cost of the implied system or component.
Risk	0.05	The technical risk of the project. This implies the technical skill necessary to design the hardware and develop firmware.
Reliability	0.2	The reliability of the hardware to perform for extended periods of time without failure.
Flexibility and Expandability	0.05	The ability to expand the firmware should more complex algorithms or additional functionality be required, or if the addition of hardware requires a software change.

Each criterion will now be discussed:

- **Robustness –** Since the ADES should adhere to industrial standards, the robustness of the implied system is essential. An industrial environment places additional strain on a system, due to its harsh conditions. The ADES should thus be environmentally sound and should be able to operate despite minuscule abnormalities in the input. The component selection should reflect the robustness requirements of the ADES.

- **Efficiency –** The hardware and architecture chosen should be capable of performing the specified requirements. The main controller should be capable of proficient control and perform additional tasks specified.

- **Cost –** Although cost is not the primary drive behind the first iteration of the ADES, the main aim behind the development of the ADES is to develop a system capable of competing with industrial systems. Consequently, although cost is not a primary factor, it should be kept in mind for later iterations.

- **Risk –** The research group has no experience in developing systems with real time operating systems (such as embedded Linux) or FPGAs with DSP or PowerPC IP cores. The learning curve and technical skill required for development should be kept in mind when selecting an architecture and components.

- **Reliability –** The requirements of the ADES specifies an MTBF of 10 years. This implies that the system should be highly reliable. The selection of the hardware to realize the system should reflect the required MTBF.

- **Flexibility and expandability –** Since this is the first iteration of the ADES, future iterations of this system require that the system be flexible and expandable, to provide for additional functionality, expansions in firmware and the capability to adapt the firmware should additional hardware be added.

3.5 Architectural evaluation

The main controller unit can either be bought as a commercial off-the-shelf (COTS) product, or be designed in-house and outsourced to be developed. The latter option has the benefit of establishing additional expertise in the research group. The disadvantages are, that it will take significantly longer, is a high-risk path and the controller will only be on an industrial level after a few iterations. This makes the COTS product a more viable option, but before this can be determined, the architecture of the system and main controller should be evaluated.

The system and main controller architectures are discussed in this section. Due to the direct correlation between the two architectures, both were evaluated simultaneously. This section discusses the two architectures by starting with the controller architecture and then the system architecture. Each architecture will be evaluated according to the decision matrix. However, since the above decision matrix includes criteria difficult to measure in a conceptual architecture, a second decision matrix was set up to evaluate the architectures. This matrix can be seen in Table 3-8. As each architecture is discussed, it will also be evaluated against this matrix. The higher the score given to each criterion, the more positive the criterion is.

Table 3-8: Controller architectural evaluation criteria

Efficiency	0.3	Measurement of performance. An indication to the ability to control the system and perform the required tasks.
Cost	0.2	This is the cost of the implied system
Risk	0.2	The technical risk of the project. This implies the technical skill necessary to design the hardware and develop firmware.
Flexibility and Expandability	0.3	The ability to expand the firmware should more complex algorithms or additional functionality be required, or if the addition of hardware requires a software change.

3.5.1 Main controller architecture

Although numerous variations of the main controller architecture exist, the fundamental ideas behind all the architectures are similar: a processor is needed to perform the control algorithm and for the scheduling of the real-time tasks and events. The processor can be combined with a co-processor, such as an FPGA, to improve performance and flexibility. The advantages of this architecture is discussed in: [39], [50], [60] and [61]. The usage of an FPGA as a co-processor expands the capability of the main controller without the need to design software for two controllers. It enables the code designed for a single controller, to be transferred to the FPGA for execution, with the minimal amount of changes. This will reduce the load on the main controller. An FPGA is also much more versatile than a single DSP. Whereas the DSP uses pipelining techniques to establish “parallel” execution, an FPGA is a true parallel device and has the ability to execute commands in parallel as well as sequentially. A DSP has a limited number of user-definable input/output (I/O) ports. An FPGA has a tremendous amount of user-definable I/O ports, when compared to a DSP. This makes the FPGA very versatile and can be configured to incorporate more UARTs for example.

A method is also needed to establish communication with the external components. This can either be established using a separated processor with several I/O ports, such as an FPGA, or incorporated into the main controller or co-processor, depending on the processor(s) used. The communication controller will have to have at least six UARTs and should be able to function concurrently. The purpose of each UART is shown as the architectures are evaluated. Also consistent throughout the evaluated architectures is non-volatile memory and an Ethernet controller. Both these components could also be included in a processor/controller.

Despite the numerous variations available, only the architectures discussed in this section were considered for the ADES. It is also possible to combine features from different architectures to define the best architecture.

Concept architecture 1

The first evaluated main controller architecture can be seen in Figure 3-7. It consists of a single master controller responsible for all real-time scheduling and for performing the position control. This architecture is weighed in Table 3-9.

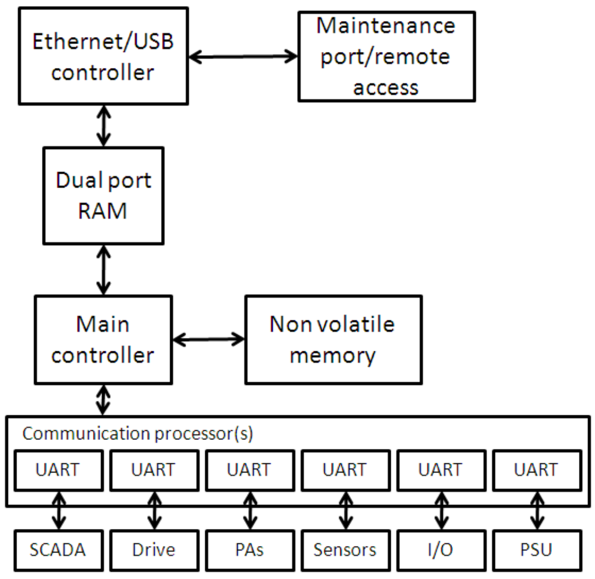


Figure 3-7: Main controller concept architecture 1

Table 3-9: Main controller architecture 1 decision matrix

	Score	**Weight**	**Weighted Score**	**Discussion**
Efficiency	40	0.3	12	Single main controller limits future expansion, flexibility and expandability. Thus lowers efficiency
Cost	60	0.2	12	Processor dependent. Developments tools are available for TI's DSPs, and the Spartan 3 FPGA.
Risk	60	0.2	15	Below average risk since two processor architectures should be learned and previous FPGA experience exist in the group.
Flexibility and Expandability	20	0.3	6	Expandability and flexibility are governed by the limitations of the main controller.
Total			45	

Concept architecture 2

Architecture 2 improves on architecture 1, due to the field programmable gate array (FPGA) used as a co-processor. Two variations on this architecture exist: with a communication line between the main controller and communication processor(s) and one with a communication line between the FPGA and communication processor(s). This is depicted in Figure 3-8 (a) and (b) respectively. Architecture 2 (b) will most likely be used if an FPGA is used as a communication processor. This is due to the fact that FPGAs have native communication standards such as low voltage differential signalling or LVDS which could drastically improve communication speed between processors. Table 3-10 evaluates this architecture according to the decision matrix.

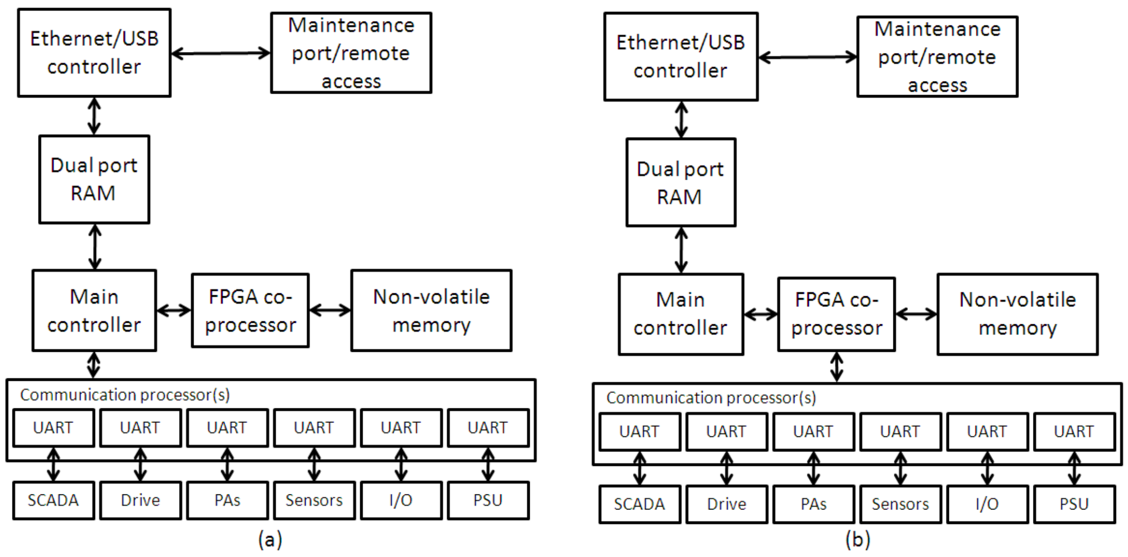


Figure 3-8: Main controller concept architecture 2: with the (a) main controller and (b) the FPGA communicating with the communication processor(s).

Table 3-10: Main controller architecture 2 decision matrix

	Score	**Weight**	**Weighted Score**	**Discussion**
Efficiency	50	0.3	15	Expansion on architecture 1. FPGA as co-processor improves efficiency. Code can be off-loaded to the FPGA for added performance.
Cost	40	0.2	8	Higher cost than architecture 1 due to the added FPGA.
Risk	50	0.2	10	Previous experience exists in the group with this type of architecture, thus average risk. Separate components can induce timing issues.
Flexibility and Expandability	70	0.3	21	Added FPGA improves system flexibility and expandability.
Total			54	

Concept architecture 3

Concept architecture 3, shown in Figure 3-9, assumes a controller or processor with included Ethernet and/or USB capabilities can be used as the master processor. This architecture is evaluated in Table 3-11.

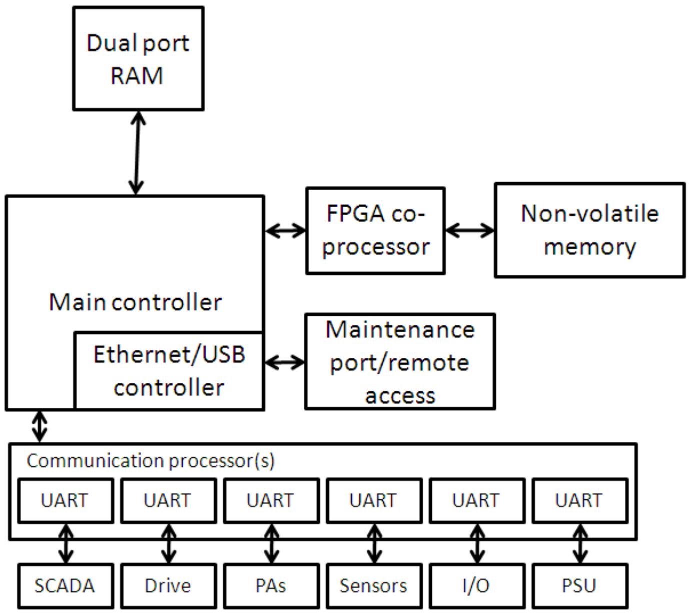


Figure 3-9: Main controller concept architecture 3

Table 3-11: Main controller architecture 3 decision matrix

	Score	**Weight**	**Weighted Score**	**Discussion**
Efficiency	40	0.3	12	Lower efficiency than architecture 2, since more strain is placed on the main controller due to the included Ethernet/USB controller.
Cost	50	0.2	10	Average cost, since a processor should be selected capable of performing both the control and Ethernet/USB
Risk	70	0.2	14	Depends on the amount and type of processor(s) used. A single processor has lower risk. An FPGA as co-processor results in average risk.
Flexibility and Expandability	50	0.3	15	Without the co-processor, the flexibility and expandability is very low, due to the added stain on the main controller. Adding a co-processor will result in high flexibility and expandability.
Total			51	

Concept architecture 4

Architecture 4, shown in Figure 3-10, takes advantage of the versatility of an FPGA by instantiating an Ethernet/USB controller on it. By instantiating an Ethernet or USB core on an FPGA relinquishes the need for another controller. The main processor still handles the AMB control algorithm and overall scheduling functions. The FPGA now incorporates all communication control and possible other co-processing functions.

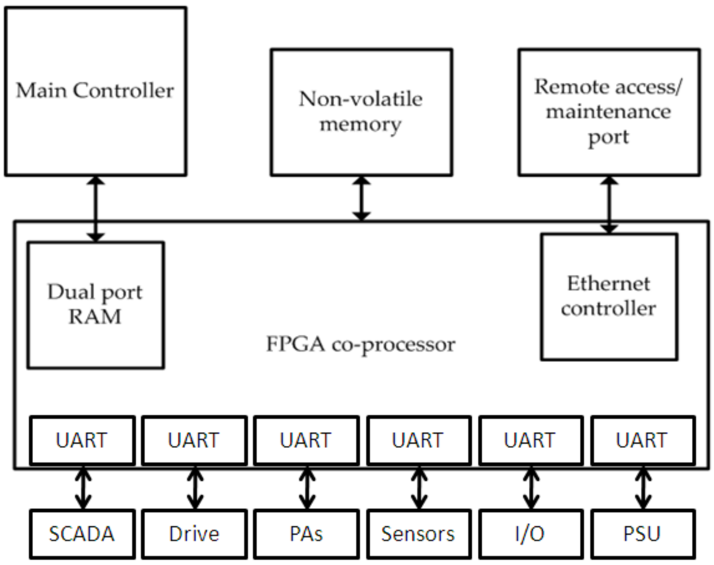


Figure 3-10: Main controller concept architecture 4

Table 3-12: Main controller architecture 4 decision matrix

	Score	**Weight**	**Weighted Score**	**Discussion**
Efficiency	60	0.3	18	Moving Ethernet/USB control from main controller to FPGA makes this architecture efficient.
Cost	20	0.2	4	An FPGA large enough to perform the required functions should be selected, thus increases cost.
Risk	70	0.2	14	Below average risk due to previous experience in this type of architecture.
Flexibility and Expandability	70	0.3	21	High flexibility and expandability because of the FPGA.
Total			57	

Concept architecture 5

It is possible to incorporate the control and communication on a single FPGA. This architecture is an expansion on architecture 2. This setup gives architecture 5 seen in Figure 3-11 and evaluated in Table 3-13.

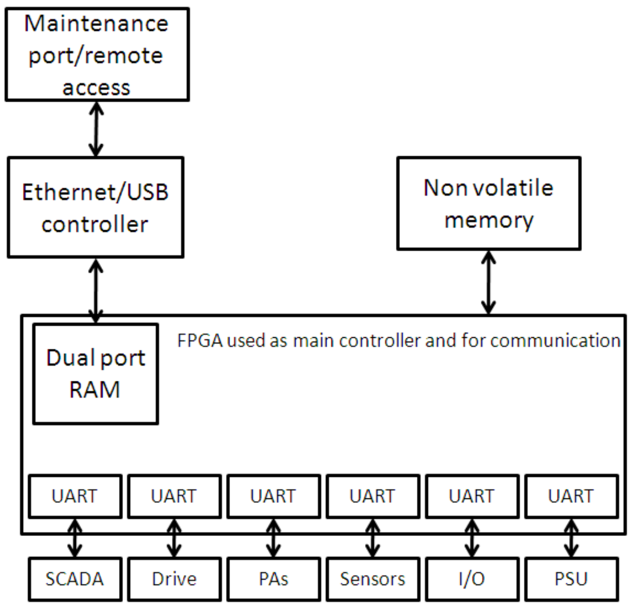


Figure 3-11: Concept architecture 5

Table 3-13: Main controller architecture 5 decision matrix

	Score	**Weight**	**Weighted Score**	**Discussion**
Efficiency	50	0.3	15	Even thought efficiency is above average when a powerful FPGA is selected, it is impeded by the fact that the control is now limited to fixed-point, unless it is emulated.
Cost	40	0.2	8	An FPGA large enough to perform the required functions should be selected, but the absence of another processor reduces cost somewhat.
Risk	50	0.2	10	Average risk due to learning curve in single FPGA architecture.
Flexibility and Expandability	60	0.3	18	FPGA induces high flexibility and expandability.
Total			51	

Concept architecture 6

This architecture expands on architecture 5 by incorporating all the functional requirements into a single FPGA. It also includes a PowerPC embedded processor and/or DSP core. The embedded PowerPC is a hardware core included in some of Xilinx$^{®}$'s top-range FPGAs. This processor will enable floating-point calculations. It is also possible to instantiate a DSP core on the FPGA, either by a purchased, or user written core.

This provides the ultimate in flexibility and performance and eliminates bottle-neck and timing problems that might arise in multi-processor architectures. Powerful FPGAs and IP cores are available that can happily perform all the functions required.

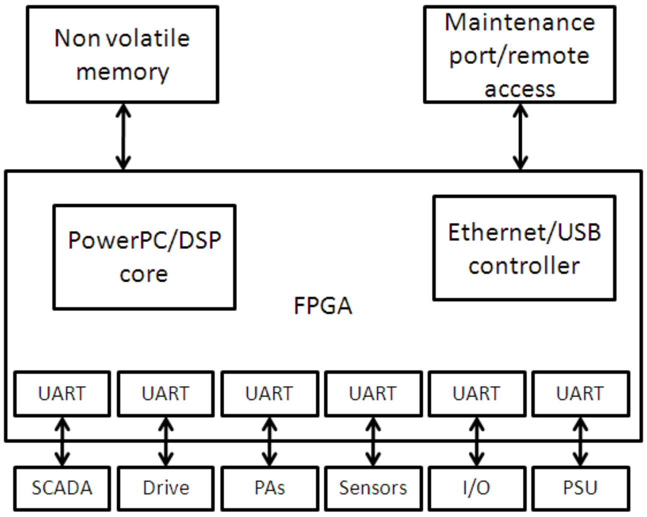


Figure 3-12: Main controller concept architecture 6

Table 3-14: Main controller architecture 6 decision matrix

	Score	**Weight**	**Weighted Score**	**Discussion**
Efficiency	70	0.3	21	High efficiency if a large FPGA is selected. PowerPC/DSP core enables floating-point capability.
Cost	60	0.2	12	Fewer components imply lower cost.
Risk	20	0.2	4	High risk due to FPGA, PowerPC/DSP learning curve.
Flexibility and Expandability	80	0.3	24	High flexibility and expandability due to large FPGA.
Total			61	

Architecture evaluation

As each architecture is evaluated against the decision matrix, it can be seen in Figure 3-13 that the top three architectures are:

- Architecture 2
- Architecture 4
- Architecture 6

Architecture 2 and 4 are suitable architectures due to the FPGA co-processor making the architectures extremely versatile. These architectures take advantage of an FPGA as a co-processor by improving the expandability of the system. Whereas a DSP is designed for performing digital signal processing and includes various built-in functionality (such as timers, PWM generators etc.), an FPGA is a more versatile device and configurable for performing virtually any function.

For instance, an IP (intellectual property) core can be instantiated on an FPGA to enable PowerPC capabilities and the pins of an FPGA can be configured for virtually any input/output such as low voltage differential signalling (LVDS), complementary metal-oxide semiconductor (CMOS) and low voltage transistor-transistor logic (LVTTL). Some FPGAs even have hardware PowerPCs instantiated. These two architectures are also relatively low risk due to previous experience in the research group with these two architectures.

Although architecture 6 is an elegant implementation of the architecture, its drawback is the high risk. The learning curve is also relatively steep when compared to architecture 2 and 4. A drawback in using an FPGA is that an FPGA is a raw silicon device and does not include built-in functions such as a DSP. The programming structure on a DSP is much easier to implement than on an FPGA. Implementing floating-point calculations on an FPGA is more complex than on a floating-point DSP, so if a single FPGA solution is used, a PowerPC or DSP core must be included.

These architectures were reviewed, and it was decided that despite architecture 6 being high risk, its elegancy and low-cost makes this architecture the most viable architecture. Section 3.6 discusses the correlation between the system and main controller architecture.

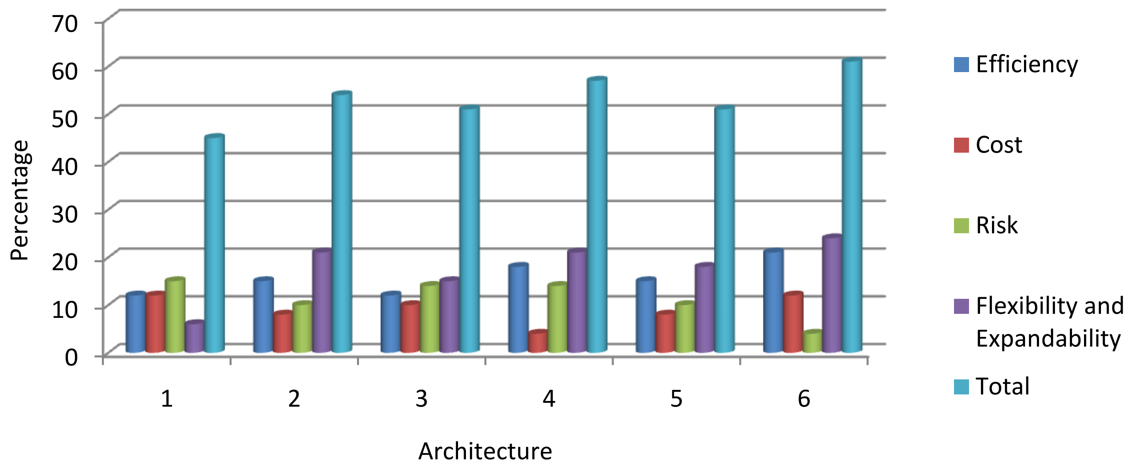


Figure 3-13: Main controller architectural comparison

3.5.2 ADES architecture

The ADES is a system that needs to adhere to industrial standards. Compact PCI (cPCI) is a more robust version of compact PCI which is used in industrial environments. cPCI thus meets the industrial requirements of the ADES and will be used as a backbone of the system. This backbone will consist of a few basic components:

- **A cPCI single board computer (SBC)**
  The SBC can be used to schedule real and/or non-real time tasks, depending on the architecture and configuration. The processor situated on the SBC can be a cental processing unit (CPU) such as a Core™ 2 Duo from Intel®, a DSP, an FPGA or a PowerPC. If a CPU or PowerPC (and certain DSPs) is used, the SBC can either run a non-real time operating system (OS) such as Windows or Linux, or a real-time OS such as RTL Linux, Vxworks or Windows CE. If a real-time OS is used, the SBC can be used to schedule both real-time and non real-time tasks. However, if a non-real time OS is used, the real-time tasks, such as PID control, should be performed by an external controller. It is also possible not to use an OS of any sort. In this case, the real-time scheduling is done using the supplied processor.

- **Communication interface**
  The communication interface can either be situated on the SBC (if enough drivers and I/O ports are available), or on a separate module. For the cPCI architecture, this separate module is a PCI mezzanine card (PMC) which slots into a PMC slot on either the SBC or a cPCI carrier card, if insufficient slots are available.

- **Profibus interface**
  A Profibus option is included in each architecture evaluated. The Profibus standard will enable the ADES system to communicate with a programmable logic controller (PLC). The Profibus option could either be a PMC module such as mentioned above, or an intellectual property (IP) core instantiated on an FPGA.

The architecture of the proposed cPCI system is shown in Figure 3-14. It depicts an SBC with two PMC sites which seats two PMC modules. The SBC slots into a cPCI backplane. Two or more of these systems can be connected via the backplane in a master-slave configuration. The rear transition module shown in the figure is, as the name suggests, a transition module which slots into the back of the cPCI backplane. It provides additional functionality and can provide additional I/O ports.

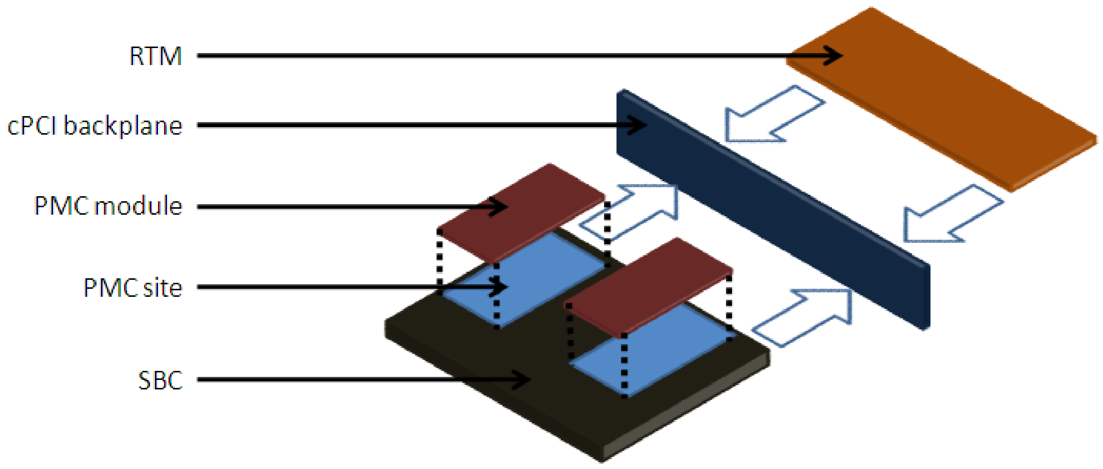
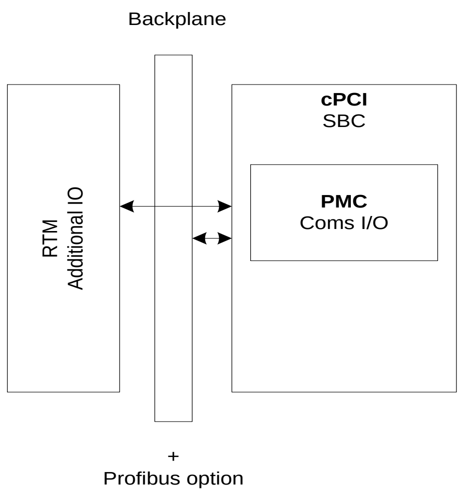


Figure 3-14: cPCI architectural layout

Just as with the architecture for the main controller, the ADES as a system also have a few architectures in consideration. These architectures are discussed in this section. The different options are numbered according to architecture and then according to variation. For example system architecture 1.2 is architecture 1, variation 2.

System architecture 1.1

The first variation of the system architecture is seen in Figure 3-15 with its characteristics. This system architecture in its simplest form implies a main controller unit as seen in section 3.5.1, architecture 1, with a single main controller. The exact architecture of the system could influence the architecture and decision matrix of the main controller architecture. For instance, an SBC with a main controller and FPGA co-processor is selected. Then system architecture 1.1 implies main controller architecture 2. The pros and cons of this architecture are given in Table 3-15.



- SBCs with a wide variety of processors are available.
- Processors include: Intel® processors, DSPs from TI®, PowerPCs from IBM®, etc.
- Real-time OS and user interface running on SBC.
- Data logging on SBC.
- SBC handles real-time and non-real time events.
- PMC mode needed for communication, due to insufficient amount of drivers and I/O ports on SBC.
- System architecture influences main controller architecture.

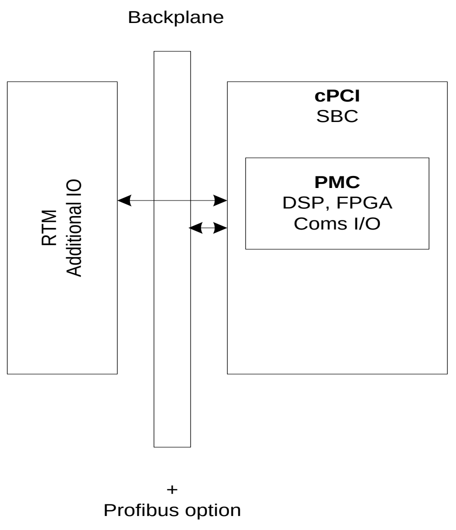
Figure 3-15: System architecture 1.1

Table 3-15: Pros and cons of system architecture 1.1

Pros	Cons
Fewer cards thus is the system cheaper.	Requires real time operating system skills.
Fewer cards imply simpler system integration.	Steep learning curve.
Compact and elegant.	
No different clock domains.	

System architecture 1.2

In this variation of system architecture 1, seen in Table 3-16. The pros and cons of this architecture can be seen in Table 3-16.



- SBC used to schedule non-real time tasks,user interface and data logging
- Real-time tasks scheduled on PMC.
- Processors on PMCs include DSPs, FPGAs and PowerPCs.
- Control algorithm executed on PMC.
- PMC module responsible for communication.
- User interface with data logging on SBC.

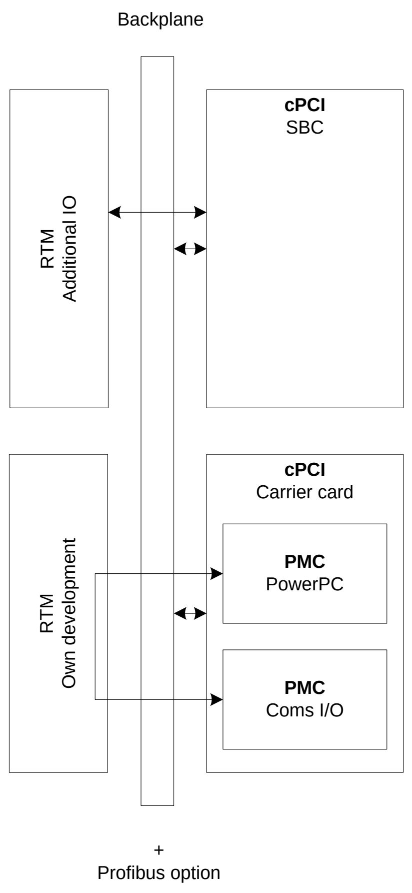
Figure 3-16: System architecture 1.2

Table 3-16: Pros and cons of system architecture 1.2

Pros	Cons
Fewer cards thus is the system cheaper.	Less elegant solution than system architecture 1.1.
Fewer cards imply simpler system integration.	Little to none PMC modules with DSP, FPGA and necessary amount of UART drivers.
Operating system can be non-real time, such as Windows®.	

System architecture 2.1

The realization of system architecture 1.2 relies heavily on the fact that an SBC can be found with 2 PMC sites and that a PMC module with a DSP and an FPGA can be found. As can be seen from Table 3-16, these PMC modules are difficult to find. Architecture 2.1 and 2.2 compensates for this by including a separate PMC module with floating-point capabilities (in this case, a PMC module with a PowerPC). The communication PMC in this architecture is assumed to be an FPGA. System architecture 2.1 can be seen in Figure 3-17 and the pros and cons of this architecture can be seen in Table 3-17.



- SBC used to schedule non-real time tasks.
- Insufficient PMC sites on SBC; if the Profibus option is a PMC module, three is needed and the most PMC sites per board are two.
- Separate carrier card is needed to site PMCs.
- Real-time tasks scheduled on PMC with floating-point-capable processor.
- PowerPC on PMC module.
- Insufficient communication drivers on PowerPC PMC, thus is a separate communication PMC needed.
- To bypass the cPCI backplane, a RTM needs to be developed to transmit data from the PowerPC PMC to the external components via the communication PMC.
- User interface with data logging still on SBC.
- cPCI backplane used to interface the PowerPC PMC with the SBC.
- cPCI master/slave configuration of backplane.

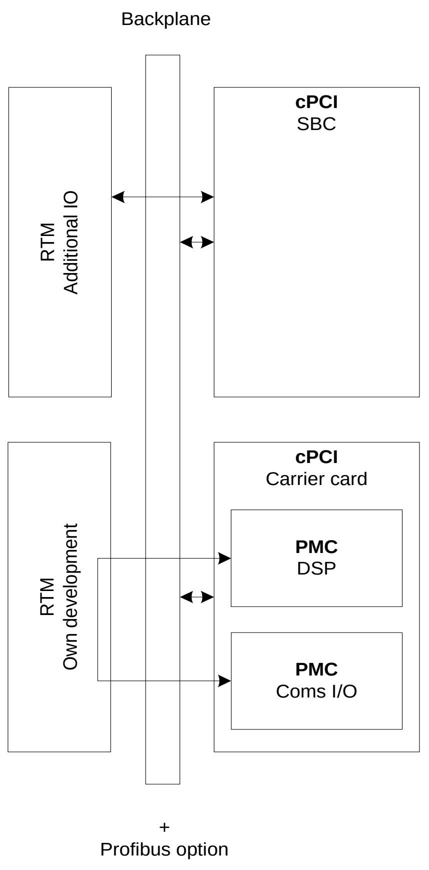
Figure 3-17: System architecture 2.1

Table 3-17: Pros and cons of system architecture 2.1

Pros	Cons
Control algorithm development separate from SBC software development.	More expensive.
Operating system on SBC can be Windows®.	Requires development of custom RTM card.
PowerPCs have multiple manufacturers.	

System architecture 2.2

The basic idea behind system architecture 2.2, shown in Figure 3-18 is the same as architecture 2.1. The only difference being the controller PMC used is in this case equipped with a DSP (such as one from TI®). The pros and cons of this architecture are shown in Table 3-18.



- SBC used to schedule non-real time tasks.
- Insufficient PMC sites on SBC; if the Profibus option is a PMC module, three is needed and the most PMC sites per board are two.
- Separate carrier card is needed to site PMCs.
- Real-time tasks scheduled on PMC with floating-point-capable processor.
- DSP on PMC module.
- Insufficient communication drivers on PowerPC PMC, thus is a separate communication PMC needed.
- To bypass the cPCI backplane, a RTM needs to be developed to transmit data from the PowerPC PMC to the external components via the communication PMC.
- User interface with data logging still on SBC.
- cPCI backplane used to interface the PowerPC PMC with the SBC.
- cPCI master/slave configuration of backplane.

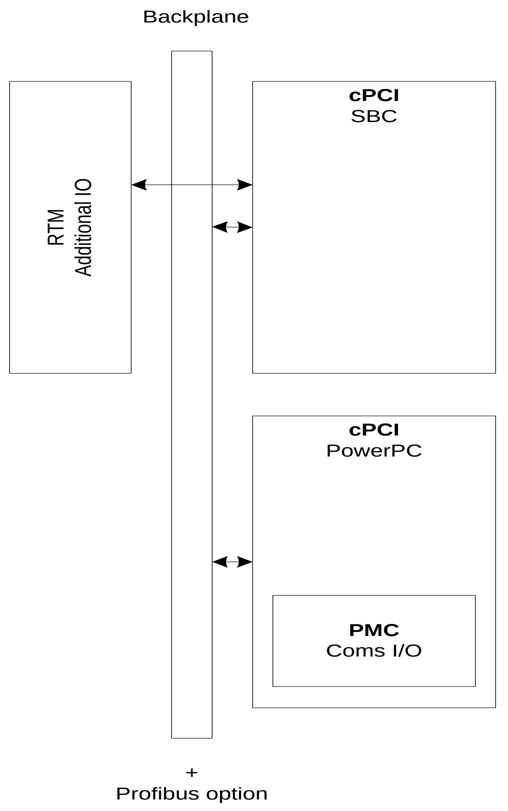
Figure 3-18: System architecture 2.2

Table 3-18: Pros and cons system of architecture 2.2

Pros	Cons
Control algorithm development separate from SBC software development.	More expensive.
Operating system on SBC can be Windows®.	Requires development of custom RTM card.
	Little to none PMCs available with floating-point capable DSPs.

System architecture 3.1

While researching different system architectures, numerous PowerPC and DSP SBC's with PMC sites were found which also slots into a cPCI backplane. Since the PowerPC is an embedded processor capable of taking a real-time OS, it is possible to use these SBCs as in system architecture 1.1, but the same issues exist as in system architecture 1.1. However, it is also possible to use a PowerPC without a real-time OS as a standalone processor and system architecture 3.1 takes advantage of this configuration. This architecture is seen in Figure 3-19 and the pros and cons of this architecture are seen in Figure 3-19.



- SBC used to schedule non-real time tasks.
- Separate cPCI with PowerPC processor.
- Communication processor situated on PowerPC cPCI board.
- Numerous PowerPC cPCI boards available.
- PowerPC cPCI board can be used as in system architecture 1.1.
- Data logging and user interface on SBC.
- cPCI backplane master/slave control difficult.
- SBC used for non-real time scheduling.
- PowerPC cPCI board used for real-time scheduling with no OS.

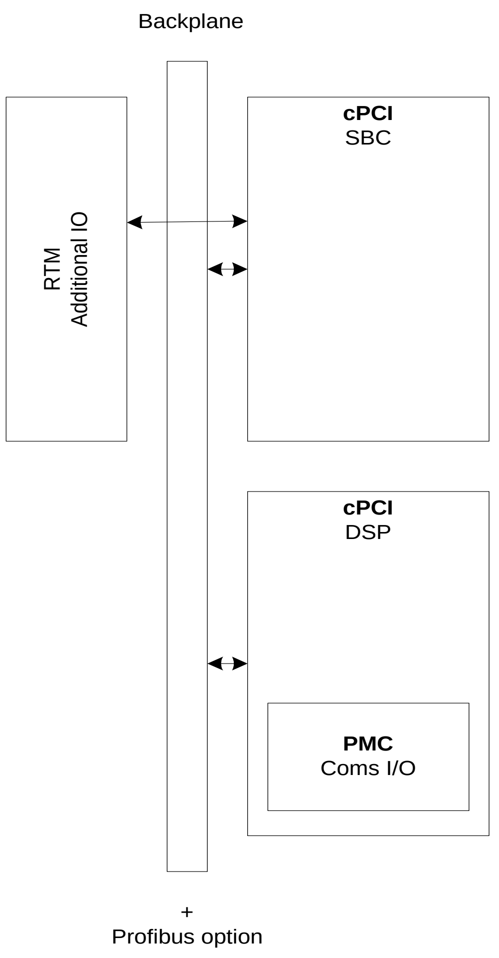
Figure 3-19: System architecture 3.1

Table 3-19: Pros and cons of system architecture 3.1

Pros	Cons
Control algorithm development separate from SBC software development.	More expensive.
Operating system on SBC can be Windows®.	PowerPCs are an unknown platform in the research group.
PowerPC cards are available and have multiple suppliers.	cPCI Master/slave scheduling of the SBCs.

System architecture 3.2

As mentioned above, a few SBCs are found with DSPs as their main processor and functionally this is the only difference between system architecture 3.1 and 3.2. This can be seen in Figure 3-20 and its pros and cons can be seen in Table 3-20.



- SBC used to schedule non-real time tasks.
- Separate DSP cPCI board.
- Communication processor situated on DSP cPCI board.
- Few cPCI boards available with floating-point DSPs.
- DSP cPCI board can be used as in system architecture 1.1.
- Data logging and user interface on SBC.
- cPCI backplane master/slave control difficult.
- SBC used for non-real time scheduling.
- DSP cPCI board used for real-time scheduling with no OS.

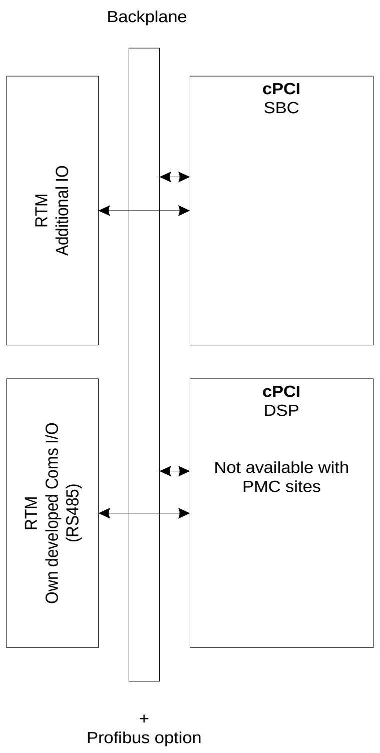
Figure 3-20: System architecture 3.2

Table 3-20: Pros and cons of system architecture 3.2

Pros	Cons
Control algorithm development separate from SBC software development.	More expensive.
Operating system on SBC can be Windows®.	cPCI DSP cards found in investigations have multiple small floating-point DSPs, which increase implementation complexity of MIMO algorithms, or only fixed-point DSPs.
DSP cards are available.	cPCI Master/slave scheduling of the SBCs.

System architecture 4.1

This architecture investigates an architecture similar to the previous two architectures, but with either no PMC sites on the control DSP SBC. In this case it is necessary to develop an RTM module to encapsulate the communication drivers and an FPGA used for communication. This can be seen in Figure 3-21. Its pros and cons can be seen in Table 3-21.



- Separate cPCI board used with no PMC sites used for scheduling of real-time tasks.
- cPCI SBC still used for non-real time taks.
- Profibus card can be situated on the SBC.
- An RTM should be developed to house the communication I/O since a direct interface is required from the DSP cPCI board.
- Master/slave PCI scheduling difficult between the SBC and DSP cPCI boards.
- Data logging and user interface still on SBC.

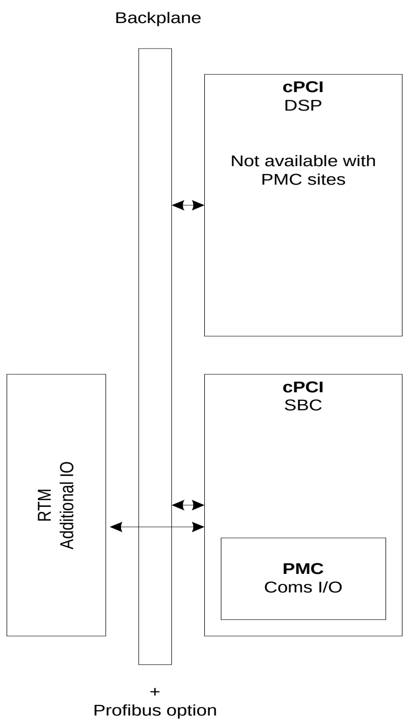
Figure 3-21: System architecture 4.1

Table 3-21: Pros and cons of system architecture 4.1

Pros	Cons
Control algorithm development separate from SBC software development.	More expensive.
Operating system on SBC can be Windows®.	cPCI DSP cards found in investigations have multiple small floating-point DSPs, which increase implementation complexity of MIMO algorithms, or only fixed-point DSPs.
DSP cards are available.	cPCI Master/slave scheduling of the SBCs.
	DSP on cards only have one manufacturer.

System architecture 4.2

In this variation of system architecture 4, the roles of the two SBC and DSP boards are inverted. It investigates the possibility to use a DSP-based SBC as the master controller of the cPCI-bus, if the DSP SBC has no PMC sites and the communication PMC module has to be situated on the Windows®-based SBC. The architecture can be seen Figure 3-22 and its pros and cons can be seen in Table 3-22.



- Roles of SBC and DSP are inverted.
- A cPCI DSP board is used without any PMC sites.
- The real-time tasks are scheduled on the cPCI DSP.
- The SBC is responsible for the scheduling of the non-real-time tasks.
- SBC used to site PMC modules due to the lack of PMC sites on the DSP.
- Roles of SBC and DSP are inverted.
- DSP board is master on the cPCI backplane.
- SBC needs to be configured as an cPCI slave.
- Communication PMC situated on cPCI SBC.
- Control data needs to be sent to SBC.

Figure 3-22: System architecture 4.2

Table 3-22: Pros and cons of system architecture 4.2

Pros	Cons
Control algorithm development separate from SBC software development.	More expensive.
Operating system on SBC can be Windows®.	cPCI DSP cards found in investigations have multiple small floating-point DSPs, which increase implementation complexity of MIMO algorithms, or only fixed-point DSPs.
DSP cards are available.	DSP as PCI master setup might not be possible.
	DSP on cards only have one manufacturer.

3.6 Selected architecture discussion

Initially the idea was to develop a main controller from scratch i.e. to determine the components necessary and then to contract an industrial partner to develop the controller. This proposes some issues. For instance, enough time should be allocated to allow the first versions of the developed boards to be iterated to eliminate potential problems. This proposes a high-risk path. In terms of cost, the in-house development of such a controller is also quite high and a COTS product was selected. This proposes an industrial architecture as discussed in the previous section.

From the previous section it is evident that only some configurations of the system architecture are readily available, while some architectures are unnecessary bulky. The aim is to have a cheaper, more effective industrial controller. After intensive research, numerous discussions and endless emails to suppliers, the options regarding system architecture were limited to two variations of architecture 1.2, as seen in Table 3-23.

The components of the two options were chosen on their specification, architecture, performance and availability (preferably form a local supplier). The following sections discuss the options for the two different options.

Table 3-23: Option for system architecture 1.2

Option 1		Option 2	
SBC	T2-6U-cPCI	**SBC**	PP41x/03x
Manufacturer	Bittware®	**Manufacturer**	Concurrent Technologies®
RTM	N/A	**RTM**	AD PP5/002
Manufacturer	N/A	**Manufacturer**	Concurrent Technologies
Communication PMC	Various PMCs are available from Acromag®. Own developed PMC I/O module.	**Communication PMC**	PMC-VFX70T
Manufacturer	Acromag®	**Manufacturer**	Acromag®

3.6.1 Option 1

Option 1 is the T2-6U-cPCI option form Bittware®. It is also based on the 6U cPCI form factor. The board consists of four ADSP-TS201 TigerSHARC™ DSPs which runs at up to 600 MHz. Also included are two Xilinx® Virtex®-II Pro FPGAs for I/O interfacing and co-processing.

T2-6U-cPCI

Although greatly over powered, the T2-6U-cPCI is one of the few cPCI boards available with floating-point processors as well as FPGAs. At this stage, no other cPCI board could be found with both a floating-point DSP and FPGA. This architecture encapsulates the advantages of main controller architecture 1 and 4. The board also has two PMC sites and could easily site both a communication and Profibus PMC. If this board is selected, the system architecture will be as in Figure 3-23.

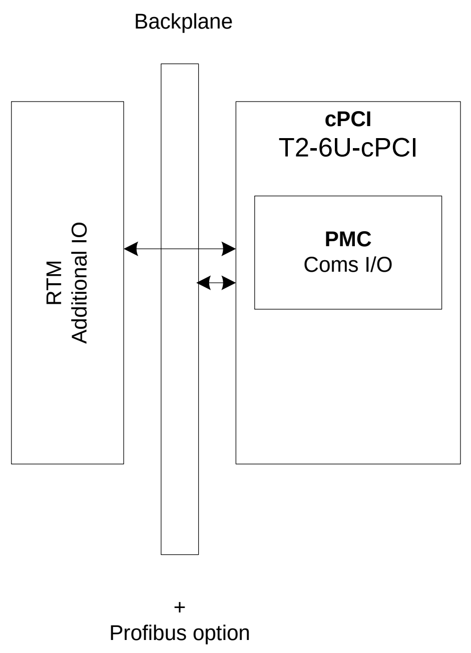


Figure 3-23: System architecture with the T2-6U-cPCI board

Since the T2-6U-cPCI board has no I/O to the external components except via the PMC sites, the communication module has to be situated on a PMC module. Various PMC modules are available from Acromag® with a wide range of FPGAs. It is also possible to develop a PMC to attain the requirements of the system. This PMC could include its own processor, but since the two FPGAs situated on the board is used for I/O interfacing, they can be routed to the external components via the PMC site using a customized PMC module.

The features of this board include:

- Two clusters of four ADSP-TS201 TigerSHARC™ DSPs which runs at up to 600 MHz.
- 28.8 GFLOPS or 3.6 GFLOPS per DSP.
- 24 Mbits of on-chip RAM.
- 16 link ports of up to 1 GByte per second each.
- Two Xilinx® Virtex®-II Pro FPGAs for I/O interfacing and co-processing.
- 128 high-performance DIO (single-ended and/or LVDS).
- 16 channels RocktIO™ high-speed serial transceivers.
- Static superscalar architecture.
- ATLANTiS™ framework featuring 6 GBytes/s of external I/O throughput via Virtex®-II Pro routing.

The board can be seen in Figure 3-24.

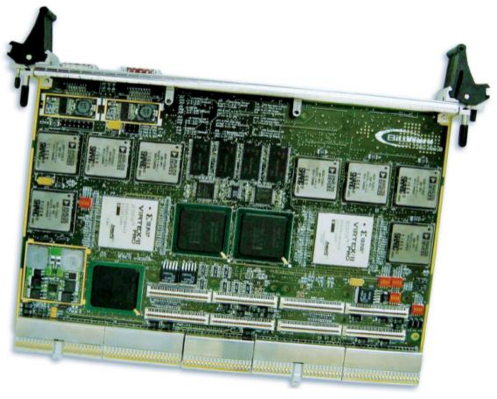

.

Figure 3-24: T2-6U-cPCI TigerSHARC™ board

3.6.2 Option 2

Option 2 is the PP 41x/03x SBC supplied by Concurrent Technologies®. It is also based on the 6U cPCI form factor. This section is dedicated to discussing this board and the complementary hardware.

PP 41x/03x

The PP 41x/03x is an Intel® Core™ 2 Duo-processor based cPCI board with two PMC sites. Although the dual core processor is not essential, the price difference between the single and dual core processors were minimal. Various configurations of this board exist, but the general features of the board include:

- 2.16 GHz Intel® Core™ 2 Duo processor T7400
- Up to 4 GB of dual channel DDR2 400 MHz[6]
- 3x 10/100/1000 Mbps Ethernet interfaces
- 4x Universal serial bus (USB) 2.0 interfaces (up to 3 if RTM modules are used, 1 via front panel). The RTM complementing this SBC, is the AD PP5/002 RTM module.
- Keyboard and mouse interface via front panel
- 32-64 bit 33/66/100 MHz cPCI interface
- Support for Linux®, Windows® 2000, XP and Server 2003, Windows® XP Embedded, QNX® and VXWorks®.

The two PMC sites will be used to site the Profibus and communication PMCs. This reflects an architecture similar to system architecture 1.2[7]. However, since it was impossible to track down

[6] The selected configuration features 1 GB RAM

a PMC module with a DSP, FPGA and enough UART drivers, this architecture needed to be modified. Should this architecture be selected, it will be necessary to select a PMC module with an FPGA and enough drivers connected to an external I/O port, to be used for implementing the communication. The FPGA on this module should either have an embedded PowerPC, or enough resources to implement a DSP core. This will enable floating-point PID control to be implemented on the module. The PMC selected for control and communication is the PMC-VFX70T module from Acromag®. This unit is discussed in the following section.

This architecture can be seen in Figure 3-25. This system architecture then encapsulates the advantages and disadvantages of architecture 6, with the exception of the Ethernet/USB controller moved from the FPGA, since this SBC's RTM module has built-in Ethernet and both SBC and RTM has USB interfaces.

Figure 3-26 shows an image of the PP 41x/03x board.

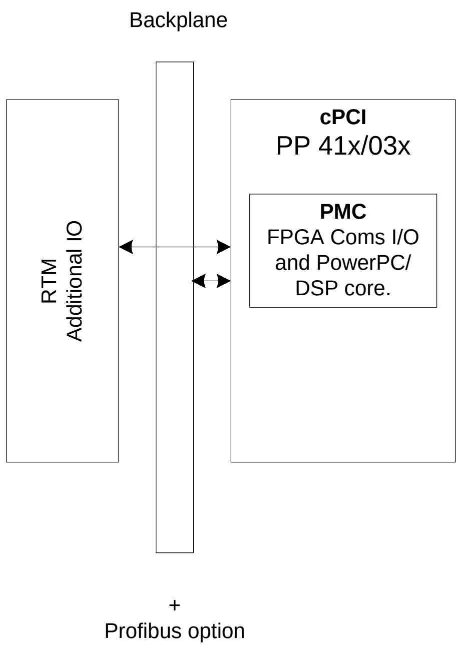


Figure 3-25: System architecture with the PP 41x/03x board

[7] A real-time OS can also be installed on the PP 41x/03x board, enabling migration to system architecture 1.1.

Figure 3-26: PP 41x/03x board

PMC-VFX70T

As already mentioned, the communication and control PMC selected to complement the PP 41x/03x SBC is the PMC-VFX70T from Acromag®. It was selected on the following criteria:

1. It has multiple I/O pins available for the communication. The number and type of I/O available depends on the front I/O transition module used. Various front I/O modules are available[8]. The one selected to be used with the PMC-VFX70T PMC is the AXM-D03 I/O module. It has 16 digital and 22 differential channels available.

2. The FPGA on this module (the VFX70T) is one of the top-level FPGAs available from Xilinx® and includes a hardware based PowerPC on chip. As discussed in the previous section, the included PowerPC is essential if the PP 41x/03x board is selected.

3. The supplier in South Africa is the same supplier as the PP 41x/03x SBC[9].

The PMC-VFX card can be seen in Figure 3-27.

[8] Other digital communication options available: AXM-D02 with 30 RS-485 differential I/O channels and AXM-D04 with 30 LVDS I/O channels.

[9] Redlinx® is the supplier of Concurrent Technologies®-products in South Africa.

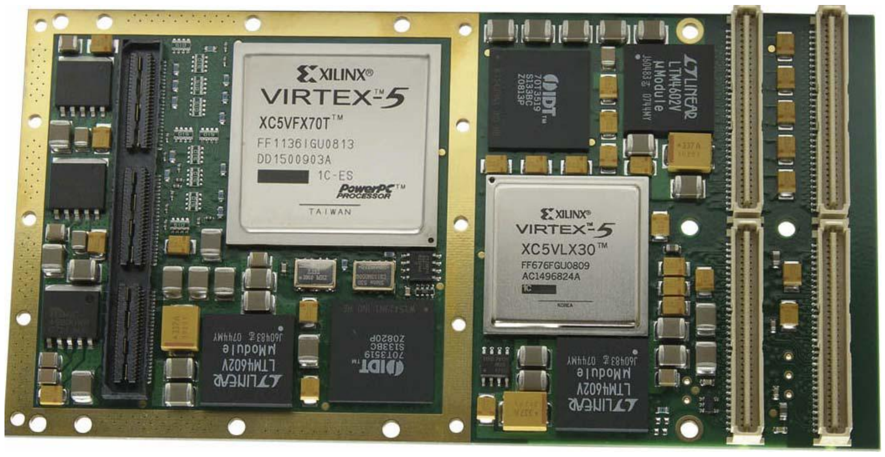


Figure 3-27: PMC-VFX70T PMC card

3.6.3 Selected architecture discussion

From the previous section, it can be seen that the system and controller architecture selection narrows down to two options. The other architectures were eliminated due to cost, availability, implementation, risk and functionality. However, the two proposed architectures have their advantages and disadvantages. This is shown in Table 3-24.

Table 3-24: Comparison between the two proposed system architectures

T2-6U-cPCI		PP 41x/03x and PMC-VFX70T	
Advantages	**Disadvantages**	**Advantages**	**Disadvantages**
Powerful architecture.	Overpowered.	Sufficient number of built-in communication channels.	Steep learning curve due to single FPGA architecture.
Previous architectural experience.	Two development environments need to be learned	Possible to migrate to architecture 1.1 which is a more elegant solution.	Possibility that the embedded PowerPC is not powerful enough for the application.
Communication will either be established via an RTM, or a PMC option.	Very expensive.	Lower cost.	

After intensive debate and a lengthy discussion, it was decided that the single FPGA implementation of the control and communication is the best alternative, although this is a higher risk option than the TigerSHARC™ DSP and FPGA board. This decision was made primarily on the difference in cost between the two systems. The T2-6U-cPCI board is extremely expensive when compared to the PP 41x/03x SBC and PMC-VFT70T. The price comparison between the two architectures is given in Table 3-25[10]

Table 3-25: Price comparison between the two architectures

Option 1		**Option 2**	
T2-6U-cPCI	R229,700.00	**PP41x/03x**	R 31,221.43
RTM	N/A	**RTM**	R 4,782.86
PMC module[11]	R54,331.25	**PMC-VFX70T**	R54,331.25
		AXM-D03	R9,343.75
Development tools	R85,900.00	**Development tools**	R6,417.93
Total cost	R369,931.25	**Total cost**	R106,097.22

3.6.4 Conclusion

After discussing the requirements of the ADES main controller and estimating the number of clock cycles required, the best architecture for the main controller was determined. Since the main controller architecture cannot exist separately from the system architecture, both architectures were evaluated simultaneously. After determining the most suitable architectures, a survey had to be done to determine whether or not the selected architectures are available in a suitable form factor and configuration. It was decided to use a commercial off-the-shelf controller to minimize risks and to reduce development time.

Based on the commercial products available in the selected architectures and form factor, the options were considered. Both options reflect system architecture 1.2. The first option is the T2-6U-cPCI TigerSHARC™ option from Bittware®. This option reflects main controller architecture 2, due to the DSP/FPGA combination on the board.

[10] Note that the price comparison only includes the hardware discussed in this section. For a comprehensive exposition of costs, refer to Appendix B.

[11] The assumption is made that the selected PMC is also a top-of the range commercial PMC module. The own development option will be substantially more expensive on the first iteration.

The second option is the PP41x/03x option. Since the PMC module in this architecture will be used to implement the real-time control of the system and the fact that a hardware PowerPC 440 core is implemented in the FPGA, this system architecture reflects main controller architecture 6, with the exception that the Ethernet and USB interface(s) are handled by the SBC and not the FPGA.

Choosing between these two architectures was difficult. The T2-6U-cPCI option is the lower risk option due to previous experience in the research group in this type of architecture. This architecture is also extremely versatile due to the added FPGA co-processor. The PP41x/03x option is an elegant solution and just as versatile as the T2-6U-cPCI option, due to the hardware PowerPC included. The lack of experience in VHDL and this type of architecture, increases the risk, should this option be selected.

Eventually, the selection between these to options boiled down to price. The PP41x/03x option is about 3.5 time more expensive than the T2-6U-cPCI. Even though price isn’t a major factor in the first iteration of the project, it should be kept in mind that the ADES is an industrial-based system and future iterations of this project should be as low cost as possible. This makes the second option the more viable alternative, despite the high risk involved. The risks involved in selecting the second architecture should be controlled and managed in order to be minimized. This can be done by scheduling VHDL courses and training in this type of architecture.

CHAPTER 4

Controller design

Although the Acromag® PMC module is accompanied by an example design featuring the setup of certain components such as clock management, PCI-bus interfaces and example code on how to access the front and rear-I/O modules, certain hardware components should be added. After setting up the hardware, the code that runs on the hardware can be written. This code can be divided into two sections: software and firmware[12]. This chapter is dedicated to the design of the necessary hardware and software components needed. The implementation of these components is discussed in chapter 5.

4.1 Software control flow

Before any hardware can be designed, it is first necessary to understand and analyse the software control flow. The software control flow depicts the flow of the control through the system. Figure 4-1 shows the software design flow. From the control flow, the required hardware can be deduced.

The units depicted are labelled as follows: The second number indicates functionality whereas the first number indicates the number of occurrences or instantiations. The best example is the dual port RAMs (DPRs). DPR Ux.6 will contain the sensor position data and will be instantiated 6 times. U1.6 will contain the x-position from the first sensor. U2.6 will contain the y-position and U3.6 will contain the z-position. U4.6 to U6.6 will contain the same data as U1.6 to U3.6, but only for the second sensor.

The focus of this dissertation is on the main controller unit. This includes the control of the ADES as well as the interfaces to the ADES components part of the control loop. This implies the DPRs containing the position and pre-filtered data as well as the DPRs housing the data to be transmitted to the power amplifiers. It also includes the filters and PCI memory interface. A short review will be given of the data flow through the system.

[12] Firmware is defined as the VHDL code running on the FPGA, whereas software is defined as the C code running on the PowerPC.

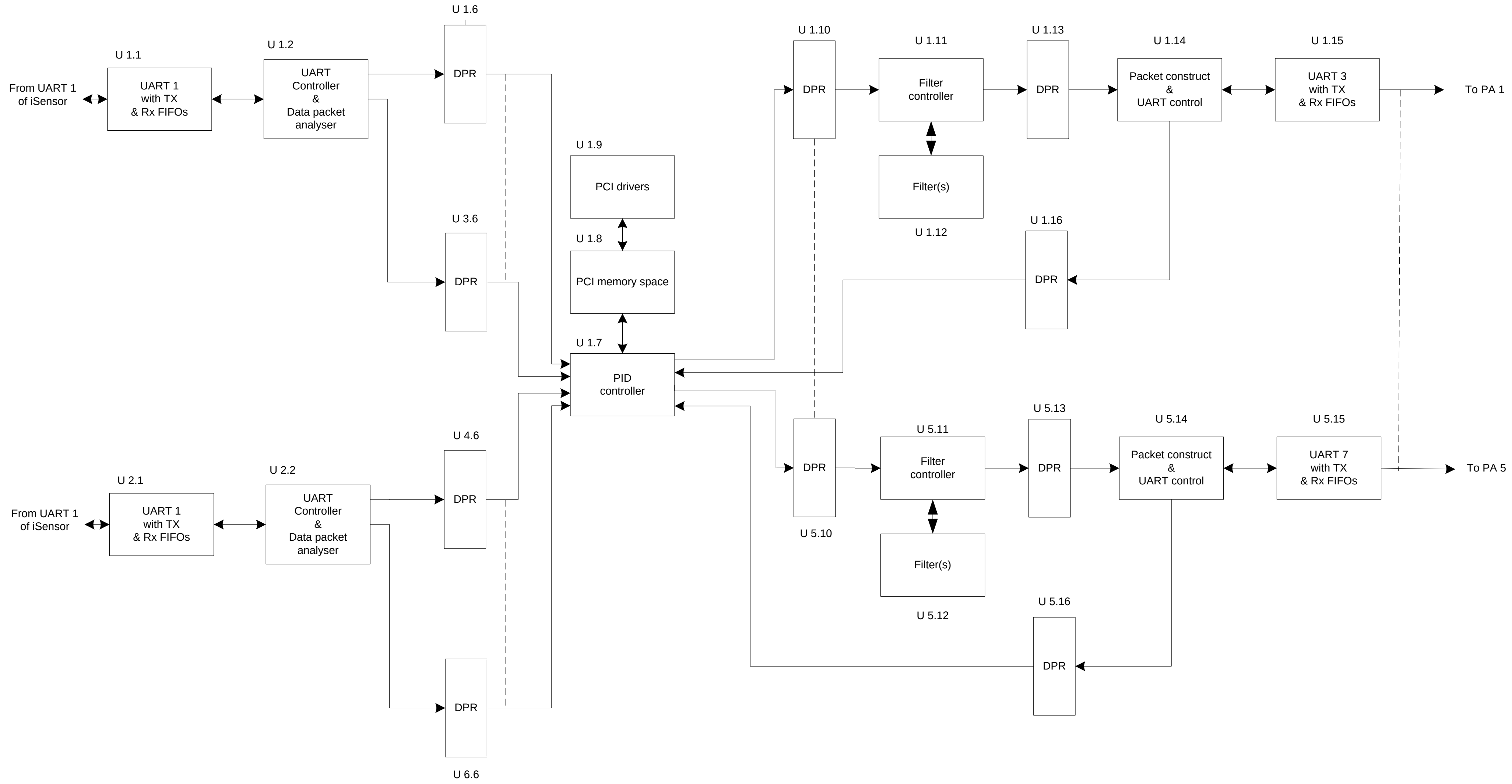


Figure 4-1: Software control flow

As discussed in the mentioned example, units 1.6 to 6.6 hold the position data from the sensors. This data is received from the UART controllers (U1.2 and U2.2). This data will be used by the PID controller (U1.7) situated on the PowerPC. The PCI memory space will serve as the interface between the PowerPC and the SBC's high-level user interface. This interface is mostly registers used by the driver software (U1.9) on the SBC to send and receive data over the PCI bus.

After the PID control has taken place, the resulting current references are written into DPR units 1.10 to 5.10. These current references will then be filtered to remove frequencies which could excite the critical frequencies of the mechanical axis resulting in an unstable and/or noisy system. Lastly, the filtered values are again saved into DPRs (U1.13 to U5.13) from where it is retrieved and sent to the power amplifiers (PA1 to PA5).

4.2 PowerPC hardware design

The PowerPC hardware design section is divided into two sections. The first section discusses the hardware and software setup of the PowerPC. The second section discusses the memory management and assignment.

4.2.1 Hardware

From the software design flow in Figure 4-1, it is easy to see that the following components are necessary for the main controller unit:

- Dual port RAM
- Filter controllers
- Filters
- An interface with the SBC

The motivation behind instantiating parallel components in VHDL for each software thread is to take advantage of the FPGA's concurrent execution capabilities, which greatly improves speed. This will ensure the only bottleneck being the PowerPC interface, since the PowerPC is a sequential processor. It is also evident that an interface of some sort is necessary for the PowerPC to communicate with the external components.

The foundation of the PowerPC hardware is the example design included with the Acromag® PMC module. This example includes the basic setup of the PowerPC, the firmware necessary for transferring data between the PCI bus and the PowerPC, the firmware used to communicate with the various front I/O modules available for the PMC module as well as the instantiation for the PowerPC as a component in VHDL. It also includes the digital clock management (DCM) blocks needed to generate the required frequencies throughout the system. The configuration of the PowerPC as set up in the example design is given in Table 4-1.

Table 4-1: PowerPC initial configuration

Reference clock frequency	200 MHz
Processor clock frequency	125 MHz
Bus clock frequency	125 MHz
Reset polarity	Active low
Floating-point unit	Not included
Hardware components	2x UARTs (for RS232 interface)
	DDR memory interface
	64 kB internal block RAM (BRAM)
	BRAM interface controller
	PCI bus interface controller (as a pcore) Master processor local bus (MPLB) controller Slave processor local bus (SPLB)

Additional components needed include:

- Multiple memory interface controllers to interface with the external DPRs.
- General purpose I/O modules, one for input and one for output.
- Interrupt controller, to connect and manage interrupts.
- A floating-point unit (FPU), since it is not included in the initial configuration.
- A fabric coprocessor bus (FCB) to connect the FPU to the processor core.
- A second PLB bus, since a PLB bus can only connect to 17 components and all the PowerPC components already instantiated along with the new IPs needed, exceed the limit.
- A PLB bridge controller to bridge the second PLB to the PLB already instantiated.

The floating-point unit mentioned above is necessary since, as with the case of most embedded processors, a PowerPC is not capable of true floating-point calculations. When a floating-point calculation is performed, it emulates the floating-point capability, which takes a significant amount of time when compared to fixed-point calculations. As can be inferred, it is possible to add a floating-point unit to the PowerPC as an auxiliary processor unit (APU). This makes it possible for the PowerPC to perform true floating-point calculations, thus greatly improving speed.

Each of the mentioned components need to be connected to a specialized bus in order to be accessible by the processor core. This implies each IP needs to have an address assigned to it. The following section discusses memory space allocation.

4.2.2 Memory space allocation

The memory space of the main controller is distributed over many different components. Eighteen blocks of dual port RAM (DPR) instantiated in VHDL, external DDR RAM and most hardware instantiated by using the PowerPC IP cores, all constitute the memory space used by the main controller unit. It is thus evident that the memory space should be designed and managed thoroughly

Dual Port RAM

Each of the eighteen instantiated DPRs' layout is exactly the same. This simplifies the VHDL design since only one component needed to be designed and instantiated numerous times. A basic DPR block can be seen in Figure 4-2 and the basic DPR data layout as used in this project is seen in Figure 4-3.

As can be seen from the figure, the memory space allocation has two view-points. When the DPR is viewed from the FPGA, each memory position is referenced by using a numbered value. For instance, PA true current 2 is referenced using the value 3 (or 0x0000003). When the DPR is viewed from the PowerPC side, the same position has to be referenced using a physical address: 0x0000000C. This is due to the fact that the port connected to the PowerPC is controlled by means of a memory controller. This memory controller is connected to the PLB, which addresses each component situated on the PLB by means of a physical address.

The CLK pin on the DPR block is used to clock data into the DPR synchronously and the RESET pin is used to clear the status of the DPR ports. The ADD- pins on the DPR points to the address where DIN is written or where DOUT is read. For this project, both data and addresses are 32-bits long. This simplifies connectivity to the PowerPC memory since it also uses 32-bit data and address busses. The data is written when WE-pins are high and the chip is enabled (CE is asserted).

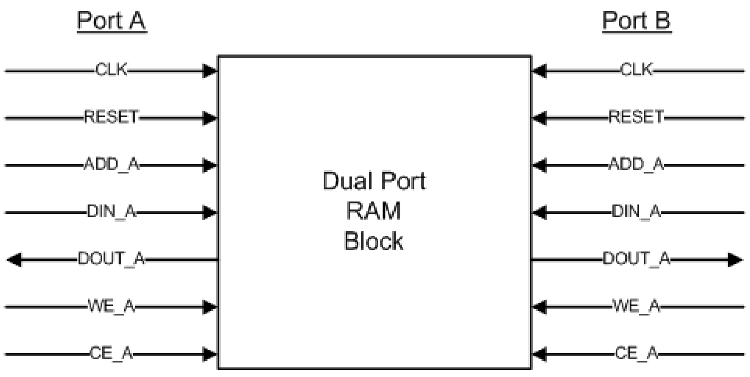


Figure 4-2: DPR block

An issue experienced connecting one port of the DPR to the PowerPC was the fact that each address from the PowerPC is 32-bits long and each consecutive PowerPC memory entry, points to every fourth entry in die DPR. The solution was to shift the PowerPC addresses so that the least significant bits were ignored. For example (when only looking at the eight least significant bits (LSBs) of the memory):

Hex value	Binary number	Binary number (LSB ignored)	Memory entry
0x0	00000000_2	000000_2	0_{10}
0x4	00000100_2	000001_2	1_{10}
0x8	00001000_2	000010_2	2_{10}
0xC	00001100_2	000011_2	3_{10}
0x10	00010000_2	000100_2	4_{10}

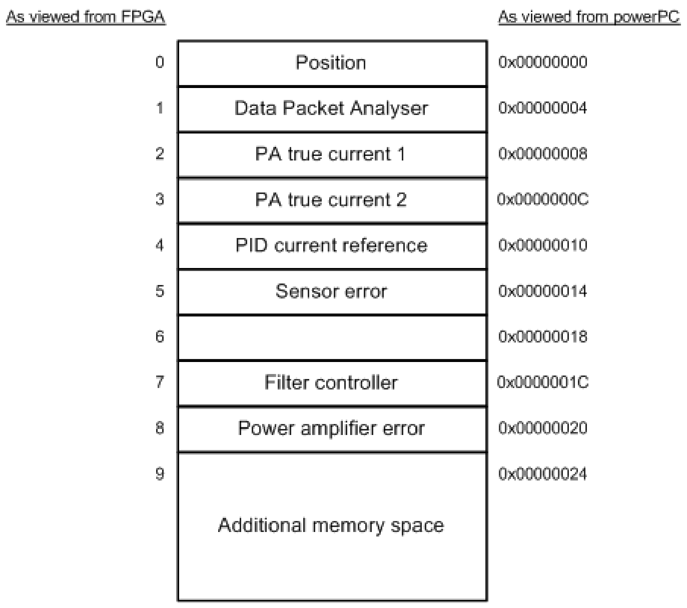


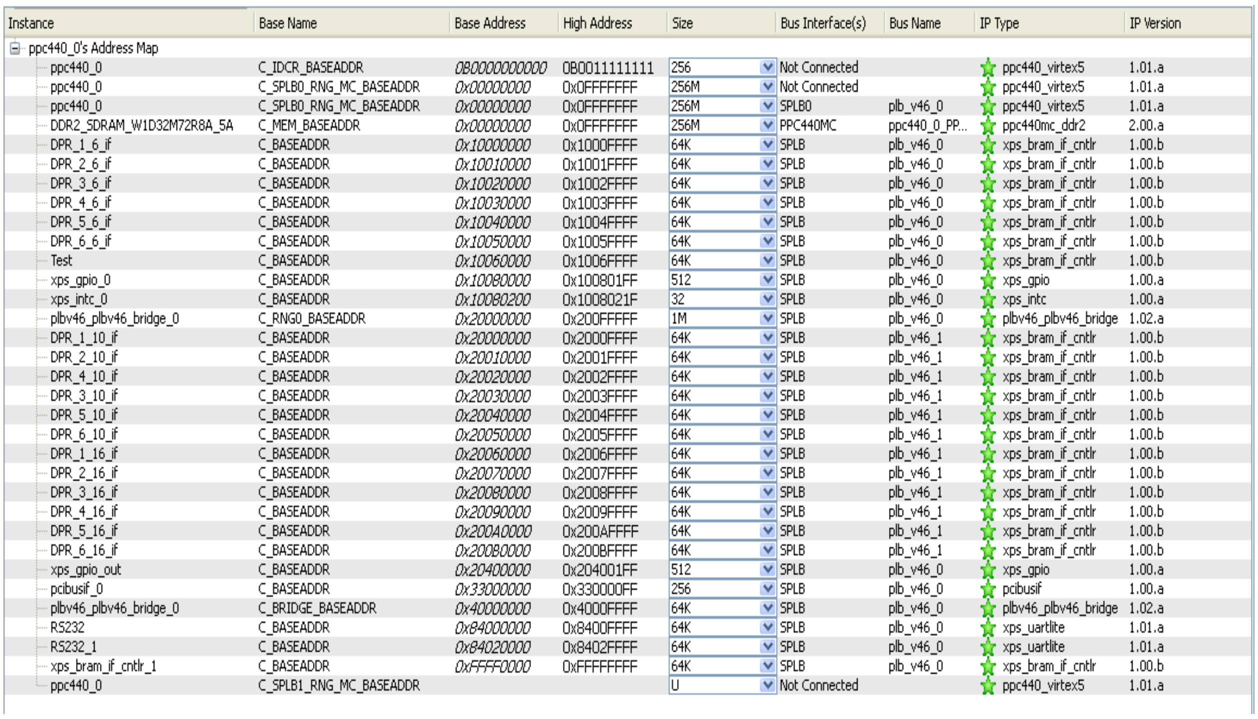
Figure 4-3: DPR memory space allocation

Double data rate (DDR) synchronous dynamic random access memory (SDRAM)

The interface between the SBC and the embedded system is the DDR RAM. This is due to the DDR being directly accessible from the PowerPC. The DDR is an external 256 MB small outline in-line memory module (SO-DIMM) package situated on the SBC. Since the DDR is mostly used by Windows® XP running on the SBC, it was decided to use the upper half of the DDR to store the data used by the embedded system and GUI. The memory layout can be seen in Figure 4-5.

PowerPC hardware memory map

As discussed earlier, the PowerPC hardware is mostly expanded by using IP cores designed to perform a specific task. Each IP core added to the PowerPC hardware connects to the processor core by means of a specialised bus. This implies that most IP cores need some form of addressing. Figure 4-4 shows the designed address map of the PowerPC hardware. The base address is the starting address of the particular piece of hardware and the bus interface shows the bus interfacing the piece of hardware to the processor core. The IP type and version is also given in the figure.



Instance	Base Name	Base Address	High Address	Size	Bus Interface(s)	Bus Name	IP Type	IP Version
ppc440_0's Address Map								
ppc440_0	C_IDCR_BASEADDR	0B0000000000	0B0011111111	256	Not Connected		ppc440_virtex5	1.01.a
ppc440_0	C_SPLB0_RNG_MC_BASEADDR	0x00000000	0x0FFFFFFF	256M	Not Connected		ppc440_virtex5	1.01.a
ppc440_0	C_SPLB0_RNG_MC_BASEADDR	0x00000000	0x0FFFFFFF	256M	SPLB0	plb_v46_0	ppc440_virtex5	1.01.a
DDR2_SDRAM_W1D32M72R8A_5A	C_MEM_BASEADDR	0x00000000	0x0FFFFFFF	256M	PPC440MC	ppc440_0_PP...	ppc440mc_ddr2	2.00.a
DPR_1_6_if	C_BASEADDR	0x10000000	0x1000FFFF	64K	SPLB	plb_v46_0	xps_bram_if_cntlr	1.00.b
DPR_2_6_if	C_BASEADDR	0x10010000	0x1001FFFF	64K	SPLB	plb_v46_0	xps_bram_if_cntlr	1.00.b
DPR_3_6_if	C_BASEADDR	0x10020000	0x1002FFFF	64K	SPLB	plb_v46_0	xps_bram_if_cntlr	1.00.b
DPR_4_6_if	C_BASEADDR	0x10030000	0x1003FFFF	64K	SPLB	plb_v46_0	xps_bram_if_cntlr	1.00.b
DPR_5_6_if	C_BASEADDR	0x10040000	0x1004FFFF	64K	SPLB	plb_v46_0	xps_bram_if_cntlr	1.00.b
DPR_6_6_if	C_BASEADDR	0x10050000	0x1005FFFF	64K	SPLB	plb_v46_0	xps_bram_if_cntlr	1.00.b
Test	C_BASEADDR	0x10060000	0x1006FFFF	64K	SPLB	plb_v46_0	xps_bram_if_cntlr	1.00.b
xps_gpio_0	C_BASEADDR	0x10080000	0x100801FF	512	SPLB	plb_v46_0	xps_gpio	1.00.a
xps_intc_0	C_BASEADDR	0x10080200	0x1008021F	32	SPLB	plb_v46_0	xps_intc	1.00.a
plbv46_plbv46_bridge_0	C_RNG0_BASEADDR	0x20000000	0x200FFFFF	1M	SPLB	plb_v46_0	plbv46_plbv46_bridge	1.02.a
DPR_1_10_if	C_BASEADDR	0x20000000	0x2000FFFF	64K	SPLB	plb_v46_1	xps_bram_if_cntlr	1.00.b
DPR_2_10_if	C_BASEADDR	0x20010000	0x2001FFFF	64K	SPLB	plb_v46_1	xps_bram_if_cntlr	1.00.b
DPR_4_10_if	C_BASEADDR	0x20020000	0x2002FFFF	64K	SPLB	plb_v46_1	xps_bram_if_cntlr	1.00.b
DPR_3_10_if	C_BASEADDR	0x20030000	0x2003FFFF	64K	SPLB	plb_v46_1	xps_bram_if_cntlr	1.00.b
DPR_5_10_if	C_BASEADDR	0x20040000	0x2004FFFF	64K	SPLB	plb_v46_1	xps_bram_if_cntlr	1.00.b
DPR_6_10_if	C_BASEADDR	0x20050000	0x2005FFFF	64K	SPLB	plb_v46_1	xps_bram_if_cntlr	1.00.b
DPR_1_16_if	C_BASEADDR	0x20060000	0x2006FFFF	64K	SPLB	plb_v46_1	xps_bram_if_cntlr	1.00.b
DPR_2_16_if	C_BASEADDR	0x20070000	0x2007FFFF	64K	SPLB	plb_v46_1	xps_bram_if_cntlr	1.00.b
DPR_3_16_if	C_BASEADDR	0x20080000	0x2008FFFF	64K	SPLB	plb_v46_1	xps_bram_if_cntlr	1.00.b
DPR_4_16_if	C_BASEADDR	0x20090000	0x2009FFFF	64K	SPLB	plb_v46_1	xps_bram_if_cntlr	1.00.b
DPR_5_16_if	C_BASEADDR	0x200A0000	0x200AFFFF	64K	SPLB	plb_v46_1	xps_bram_if_cntlr	1.00.b
DPR_6_16_if	C_BASEADDR	0x200B0000	0x200BFFFF	64K	SPLB	plb_v46_1	xps_bram_if_cntlr	1.00.b
xps_gpio_out	C_BASEADDR	0x20400000	0x204001FF	512	SPLB	plb_v46_0	xps_gpio	1.00.a
pcibusif_0	C_BASEADDR	0x33000000	0x330000FF	256	SPLB	plb_v46_0	pcibusif	1.00.a
plbv46_plbv46_bridge_0	C_BRIDGE_BASEADDR	0x40000000	0x4000FFFF	64K	SPLB	plb_v46_0	plbv46_plbv46_bridge	1.02.a
RS232	C_BASEADDR	0x84000000	0x8400FFFF	64K	SPLB	plb_v46_0	xps_uartlite	1.01.a
RS232_1	C_BASEADDR	0x84020000	0x8402FFFF	64K	SPLB	plb_v46_0	xps_uartlite	1.01.a
xps_bram_if_cntlr_1	C_BASEADDR	0xFFFF0000	0xFFFFFFFF	64K	SPLB	plb_v46_0	xps_bram_if_cntlr	1.00.b
ppc440_0	C_SPLB1_RNG_MC_BASEADDR			U	Not Connected		ppc440_virtex5	1.01.a

Figure 4-4: Designed PowerPC address map

0x00000000	Used by Windows® XP
0x0FFFF54	Current ref PA board 1
0x0FFFF58	Current ref PA board 2
0x0FFFF5C	Current ref PA board 3
0x0FFFF60	Current ref PA board 4
0x0FFFF64	Current ref PA board 5
0x0FFFF68	Sensor1 x-error
0x0FFFF6C	Sensor1 y-error
0x0FFFF70	Sensor1 z-error
0x0FFFF74	Sensor2 x-error
0x0FFFF78	Sensor2 y-error
0x0FFFF7C	Sensor2 z-error
0x0FFFF80	PA1 error
0x0FFFF84	PA2 error
0x0FFFF88	PA3 error
0x0FFFF8C	PA4 error
0x0FFFF90	PA5 error
0x0FFFF94	Sensor1 x-position
0x0FFFF98	Sensor1 y-position
0x0FFFF9C	Sensor1 z-position
0x0FFFFA0	Sensor2 x-position
0x0FFFFA4	Sensor2 y-position
0x0FFFFA8	Sensor2 z-position
0x0FFFFAC	Radial PID: parameter P
0x0FFFFB0	Radial PID: parameter I
0x0FFFFB4	Radial PID: parameter D
0x0FFFFB8	Axial PID: parameter P
0x0FFFFBC	Axial PID: parameter I
0x0FFFFC0	Axial PID: parameter D
0x0FFFFC4	State
0x0FFFFC8	State check
0x0FFFFCC	PA 1 actual current
0x0FFFFD0	PA 2 actual current
0x0FFFFD4	PA 3 actual current
0x0FFFFD8	PA 4 actual current
0x0FFFFDC	PA 5 actual current
0x0FFFFE0	PA 6 actual current
0x0FFFFE4	PA 7 actual current
0x0FFFFE8	PA 8 actual current
0x0FFFFEC	PA 9 actual current
0x0FFFFF0	PA 10 actual current
0x0FFFFF4	Interrupt: PID
0x0FFFFF8	Interrupt: State change
0x0FFFFFC	Extra

Figure 4-5: Designed DDR SDRAM memory space

4.3 Embedded state machine

The specification for the different states and state transitions of the ADES system were given in chapter 3. These states and state transitions should occur on the embedded controller as the system changes between states either by the high-level GUI running on the SBC, or by a PLC registered error. As will be discussed in section 4.5, these state changes will register an interrupt by the embedded control. If this interrupt occurs, a value is read from the DDR indicating the following state. This induces a state transition. A specific state has three functions:

- Entry – This function invokes the state specific functions that should occur before the state run is initialized.
- Run – This encapsulates all the functions that execute while the state is running.
- Request – This function compares the current state to the next state and if these two states differ, a state change request is issued.

The embedded state machine is seen in Figure 4-6. It shows the states in which the system can reside, as well as the transitions between states. Please note that another state, "*Maintenance*" is not shown in the figure. The system should be able to make the transition from any state after "*Power_Down*" into the "*Maintenance*" state should the need for maintenance occur. This was left out of the figure, since it made it cluttered and difficult to read.

The functionality of the different states is the same as in the specifications given in chapter 3, with the exception of the two added states: "*NonCritStop*" and "*CritStop*" and some minor adjustments. These two states were added should a condition occur in which a critical or non-critical stop is invoked.

Even though the state machine includes states such as "*Programming"*, "*Sensitivity_Stand*", "*Sensitivity_Norm*" and "*Maintenance*", time constraints and controller limitations prohibited the implementation of these states.

A short discussion of the state machine follows Figure 4-6. Where applicable, each discussion will be divided into the three functions of each state.

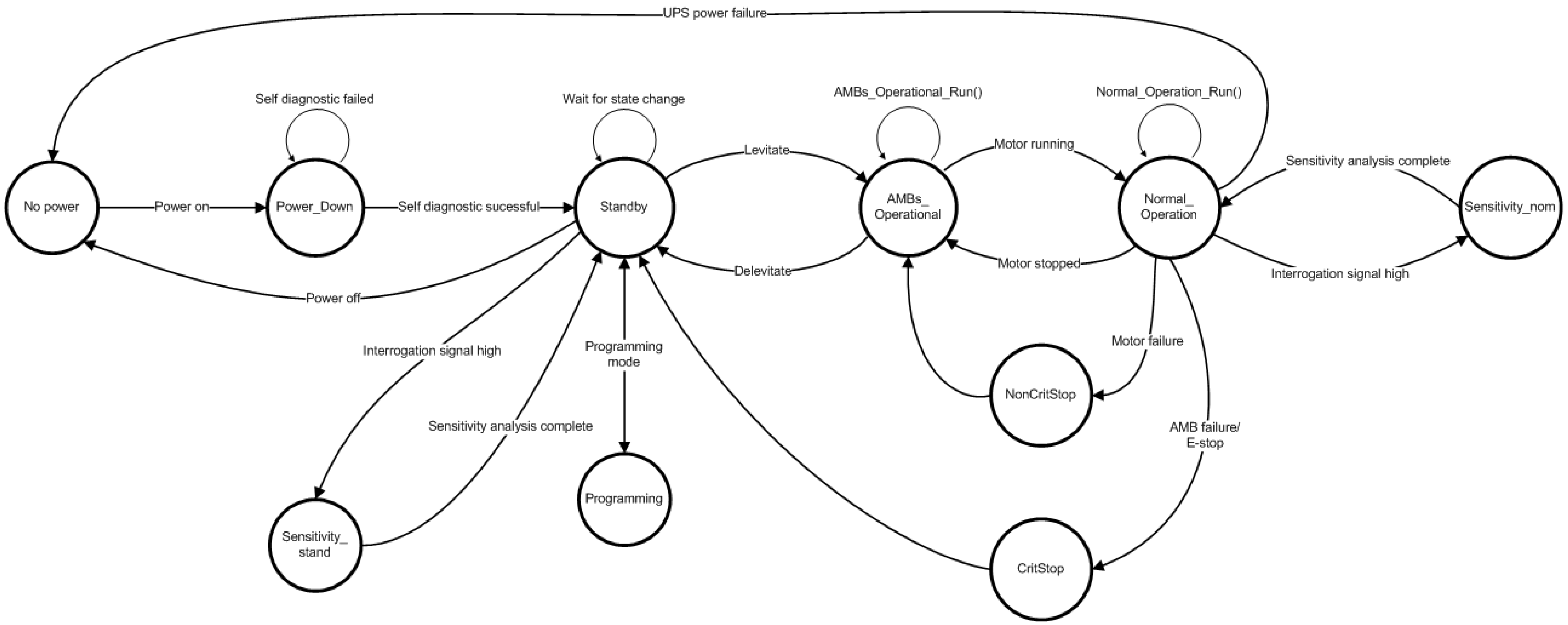


Figure 4-6: Embedded state machine

Table 4-2: Discussion of embedded state diagram (Part 1)

State	Functionality	Required conditions	% Implemented
No_power	Not a physical state. This state was brought in to simplify logistics of the state machine.	No power or UPS fails to deliver power to the system.	Not implemented – not a physical state.
Power_Down	Power-up state. Name was chosen to correspond with specified states. State initializes interrupts, exception handling and general purpose I/O ports. Self-diagnostic test and start-up procedures are also performed.	If self-diagnostic test fails, system resides in this state. If successful, system makes a transition to Standby	50 % - Self-diagnostic test needs to be included in future iterations of this project.
Standby	State initializes all PID parameters and enters the Standby-state. Data logging is enabled and no current reference is send to the power amplifiers.	Interrupt for a state change is triggered from the SBC. Transition to only four states is allowed: Power_Down, Programming, Sensitivity_stand and AMBs_Operational.	100% - All functionality is included in this state.
Programming	The programming state of the system. This state will enable an administrative user to update the firmware of the system.	Only possible to enter the Programming-state from the Standby position. A standby-request is issued form the administrator and the Programming-state is entered.	1% - State included for future expansion of the project. Out of scope for the first iteration of the system.
Sensitivity_stand	The basic idea behind this state was to include an automatic sensitivity analysis of the system while in Standby-mode.	State is entered when an interrogation signal is sent to the main controller during Standby.	1% - Due to time and resource constraints, all sensitivity analysis were moved off the embedded controller. Possible to include in future iterations.

Table 4-3: Discussion of embedded state machine (Part 2)

State	Functionality	Required conditions	% Implemented
AMBs_Operational	In this state the AMBs are operational. A current reference is sent to the PAs which drives the electromagnetic actuators to levitate the rotor. PID control is performed and data is also logged in this state.	This state is entered form *Standby* if the levitate command is issued, or from the *Normal_Operation* if the motor is stopped.	100% - Although this state is fully implemented, a few improvements can be made. For instance the current references can also be logged.
Normal_Operation	The only difference between this state and *AMBs_Operational* is a safety lockout which prevents transition to Standby while the motor is operational. Currently only the run-function from *AMB_Operational* is called.	This state in entered from *AMBs_Operational* when the motor is enabled.	80% - In future iterations of this project, motor control can be included in this state.
Sensitivity_norm	The basic idea behind this state was to include an automatic sensitivity analysis of the system while in normal operation.	State is entered when an interrogation signal is sent to the main controller during *Normal_Operation*.	1% - Refer to *Sensitivity_stand*.
NonCritStop	This state is added to provide for a non-critical stop during normal operation. Should this stop occur, the system transitions to *AMBs_Operational* via this state.	A transition to this state occurs when a non-critical error is detected and the system is in normal operation.	1% - Although this state was included, no action is taken. Future studies can determine the best course of action for a non-critical stop.
CritStop	Included for the same reason as *NonCritStop*, with the exception that the system transitions to Standby via this state should a critical stop occur.	A transition to the state occurs when a critical error is detected during normal operation.	1% - See *NonCritStop*.

Table 4-4: Discussion of embedded state machine (Part 3)

State	Functionality	Required conditions	% Implemented
Maintenance	This is the state in which all maintenance is done. This also includes physical maintenance on the system. At this stage, this state was out of scope for this project and included for modularity.	A transition from any state can be made to this state if maintenance is needed.	1% - Although out of scope for the first iteration of this project, maintenance is a crucial part of an industrial system. This state needs to be included in future iterations of this project.

4.4 Control algorithm

The control algorithm for the main controller is PID control. This first part of the section is dedicated to the design of the PID control algorithm with an additional pole for stability. Since system timing is of utmost importance, the second part is dedicated towards designing the system timing.

4.4.1 PID control

Section 2.2.1 gave an example of a simple discrete PID controller. However, external tests on the mechanical system using dSPACE® deemed it necessary to move the pole of the differentiator from a fixed position at $-\frac{2}{T}$ to an adjustable value: $-\omega_p$ [56]. This limits the gain of the differentiator at high frequencies. The PID block diagram with shifted pole is given in Figure 4-7.

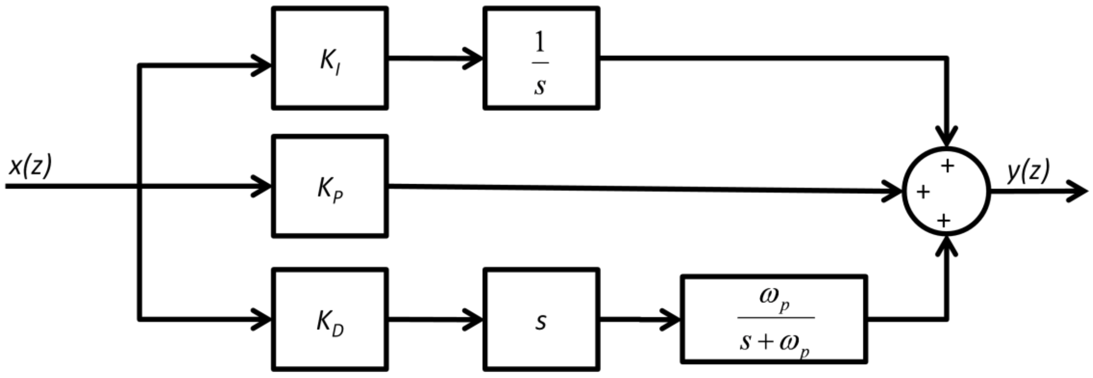


Figure 4-7: PID block diagram with differential pole moved to $-\omega_p$

If this PID controller is given by:

$$D(s) = K_p + \frac{K_I}{s} + K_D s \tag{4.1}$$

then a PID controller with a pole at -ω_p is given by:

$$D(s) = K_p + \frac{K_I}{s} + \frac{K_D s}{\left(1 + \frac{s}{\omega_p}\right)} \tag{4.2}$$

By using the bilinear transform and replacing $s = \frac{2}{T}\frac{z-1}{z+1}$:

$$D(z) = \frac{z^2\left[K_P \;\; 2T\omega_p + 4 \;\; + K_I T \;\; T\omega_p + 2 \;\; + 4K_D\omega_p\right]}{z^2 \;\; 2T\omega_p + 4 \;\; + z(-8) + 4 \;\; 4 - 2T\omega_p} + \frac{z\left[-8K_P + K_I T \;\; 2T\omega_p \;\; - 8K_D\omega_p\right]}{z^2 \;\; 2T\omega_p + 4 \;\; + z(-8) + 4 \;\; 4 - 2T\omega_p}$$

$$+ \frac{\left[2K_P \;\; 2 - T\omega_p \;\; + K_I T \;\; T\omega_p - 2 \;\; + 4K_D\omega_p\right]}{z^2 \;\; 2T\omega_p + 4 \;\; + z(-8) + 4 \;\; 4 - 2T\omega_p} \tag{4.3}$$

If *D(z)* is written in the form of $D(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}}$:

$$\begin{aligned}
a_0 &= K_p + K_I \frac{T}{2} + K_D \frac{4\omega_p}{2T\omega_p + 4} \\
a_1 &= K_P \left(\frac{-8}{2T\omega_p + 4}\right) + K_I T \left(\frac{2T\omega_p}{2T\omega_p + 4}\right) + K_D \left(\frac{-8\omega_p}{2T\omega_p + 4}\right) \\
a_2 &= 2K_P \left(\frac{-T\omega_p + 2}{2T\omega_p + 4}\right) + K_I T \left(\frac{T\omega_p - 2}{2T\omega_p + 4}\right) + K_D \left(\frac{4\omega_p}{2T\omega_p + 4}\right) \\
b_1 &= \frac{-8}{2T\omega_p + 4} \\
b_2 &= \left(\frac{-2T\omega_p + 4}{2T\omega_p + 4}\right)
\end{aligned} \tag{4.4}$$

which can be implemented[13] by the 1D[14] second order differential equations [56]:

$$\begin{aligned} 1D: m\ \ k\ \ &= x\ \ k\ \ - b_1 m\ \ k-1\ \ - b_2 m\ \ k-2 \\ y\ \ k\ \ &= a_0 m\ \ k\ \ + a_1 m\ \ k-1\ \ + a_2 m\ \ k-2 \end{aligned} \tag{4.5}$$

The structure for the 1D second-order module is seen in Figure 4-8.

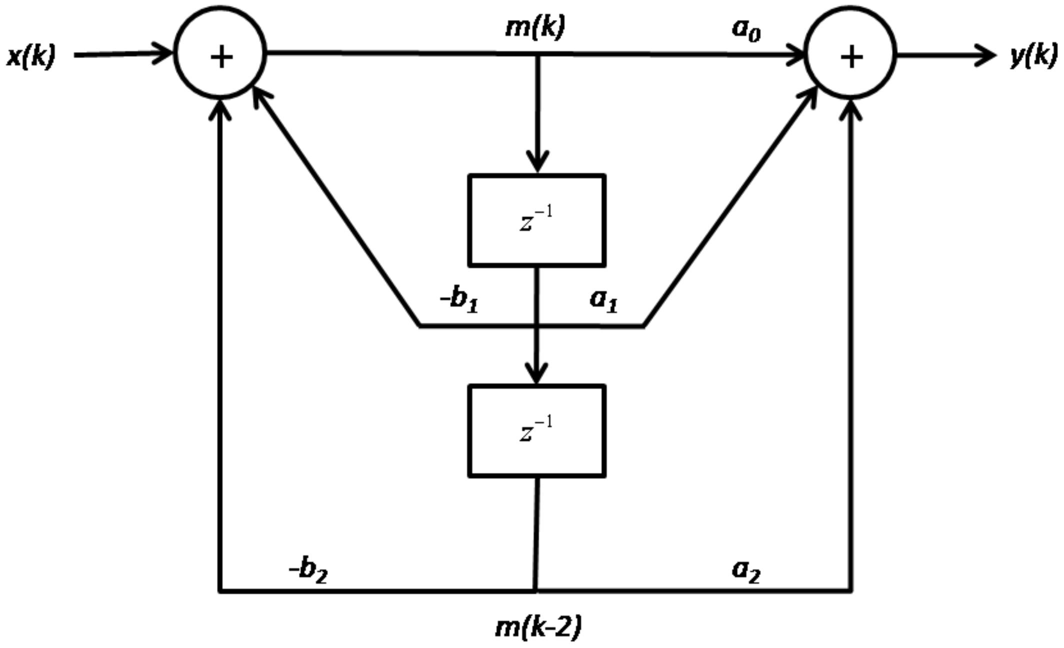


Figure 4-8: 1D second-order structure

4.4.2 System timing design

As mentioned previously, the control cycle is specified to be 20 kHz. The control, filtering and communication should execute in this 50 µs control cycle. The primary functions that should execute, are given in Figure 4-9.

As can be seen from the figure, the control is always a clock cycle behind the current position data sent from the sensor. From the sensor's[15] communication timeline, it can be seen that the sensor transmits the current position value at the beginning of the 20 kHz sync signal. Should the sync signal fail, the UART controller on the main controller unit prompts the sensor board to transmit data.

[13] The implementation of the PID on the system is discussed in the next chapter.

[14] First direct structure

[15] iSensor in the figure, which is the in-house developed inductive sensor.

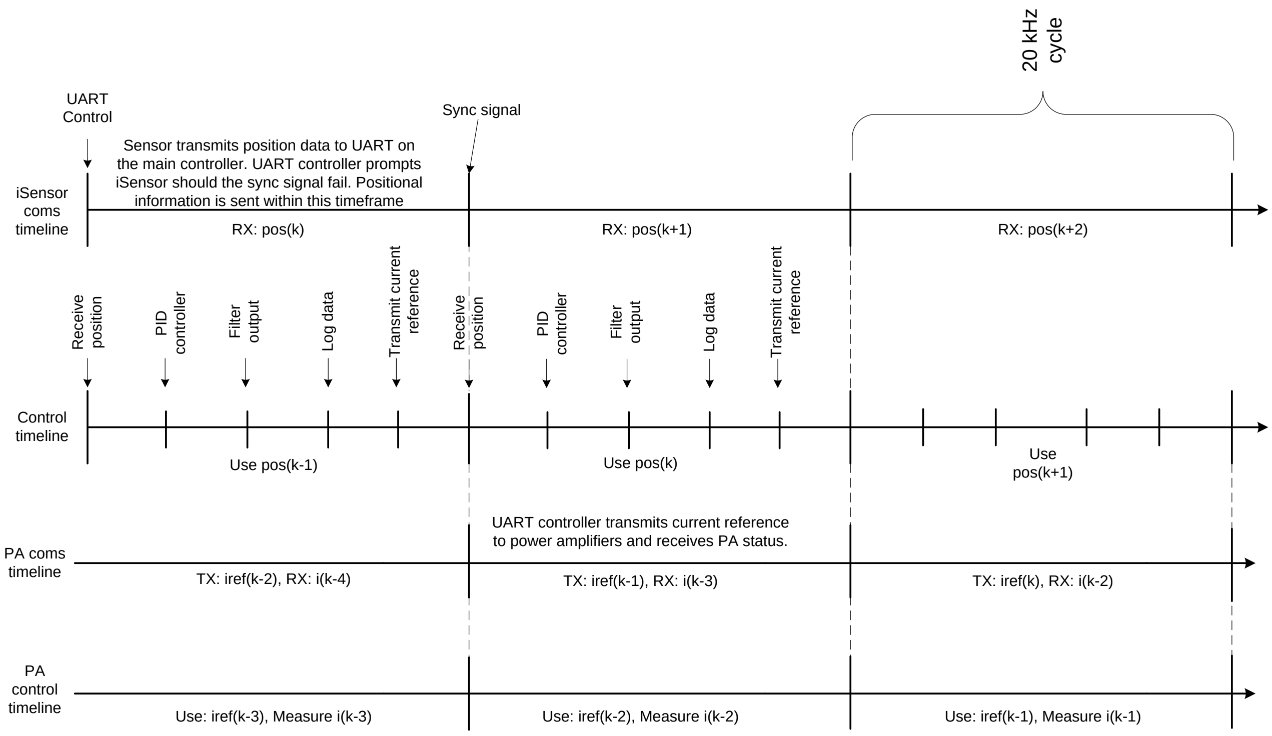

20 kHz cycle
UART Control
Sync signal
Sensor transmits position data to UART on the main controller. UART controller prompts iSensor should the sync signal fail. Positional information is sent within this timeframe
iSensor coms timeline
RX: pos(k)
RX: pos(k+1)
RX: pos(k+2)
Receive position
PID controller
Filter output
Log data
Transmit current reference
Control timeline
Use pos(k-1)
Use pos(k)
Use pos(k+1)
UART controller transmits current reference to power amplifiers and receives PA status.
PA coms timeline
TX: iref(k-2), RX: i(k-4)
TX: iref(k-1), RX: i(k-3)
TX: iref(k), RX: i(k-2)
PA control timeline
Use: iref(k-3), Measure i(k-3)
Use: iref(k-2), Measure i(k-2)
Use: iref(k-1), Measure i(k-1)

Figure 4-9 : System timing diagram

At the start of the 20 kHz control cycle, the main controller retrieves the previous position data from the memory. This data is then sent to a PID control algorithm to calculate the current reference. The current reference is then filtered before it is sent to the power amplifiers. After the value is sent, the controller logs data into memory to be displayed on the SBC.

By the time the power amplifiers receive the current reference, the current reference is about two control cycles old and by the time the current reference is used by the control algorithm, it is about three control cycles old. When the control algorithm elapses, it transmits its current status back to the main controller unit. This allows the power amplifier's actual current to be viewed on the user interface.

4.5 Interrupt design

Six possible interrupts for the main controller unit were identified. These interrupts were identified regardless whether or not they will be used. They were selected as such to enable future upgrades and expansions of the interrupt management. The identified interrupts and their corresponding variable names are listed in Table 4-5.

.Seeing as these interrupts originate externally from the embedded PowerPC, these interrupts should be connected to the PowerPC in some way. However, the PowerPC only has one external interrupt pin, EICC440EXTIRQ [44]. Only one interrupt could thus be connected and an interrupt controller should be used. The interrupt controller is IP (intellectual property) available for the PowerPC, which enables the use of numerous interrupts from various sources. This also enables future expansion of interrupts since a wide variety of interrupts can be connected to this controller and the severity of each interrupt can be set.

Table 4-5: Identified interrupts

Identified interrupt	Variable name
New data in units U1.6 to U6.6	DPR_Ux_6_int
New data in units U1.10 to U6.10	DPR_Ux_10_int
New data in units U1.16 to U6.16	DPR_Ux_16_int
A state change from the single board computer (SBC)	DDR_int_ST
PID parameter change	DDR_int_PID
20 kHz synchronization signal	Syncsig

Since each interrupt is generated externally, an interface to the VHDL code is necessary. Another IP called a general purpose I/O (GPIO) controller is connected to the PowerPC. This GPIO controller is connected to the processor local bus (PLB) which allows for data to be transferred internally between the PowerPC and the GPIO controller. The controller also allows for the connection of an external bus. The width of the external bus is also adjustable. In this particular case a bus width of eight bits was adequate. This will allow for the connection of the six interrupt signals. The GPIO controller can also be set-up to generate an interrupt. The interrupt can be set-up to trigger should any bit on the bus change (either on a rising or falling edge) or stay the same for a certain amount of time. This triggered interrupt is connected to the interrupt controller. The advantage of this setup is that the GPIO controller's external port can be expanded to take more (or less) bits and any bit can trigger an interrupt. Interrupts from other components (such as the PCI bus or RS232 ports) can be connected to the interrupt controller. The hardware setup of the interrupts is depicted in Figure 4-10. As can be seen, the GPIO controller is connected to IOINT. IOINT is a signal declared in VHDL with the same width as the output of the GPIO controller (in this case, eight bits). The GPIO controller is setup so that a rising edge on any bits in IOINT will trigger an interrupt on IP2_INT_IRPT.

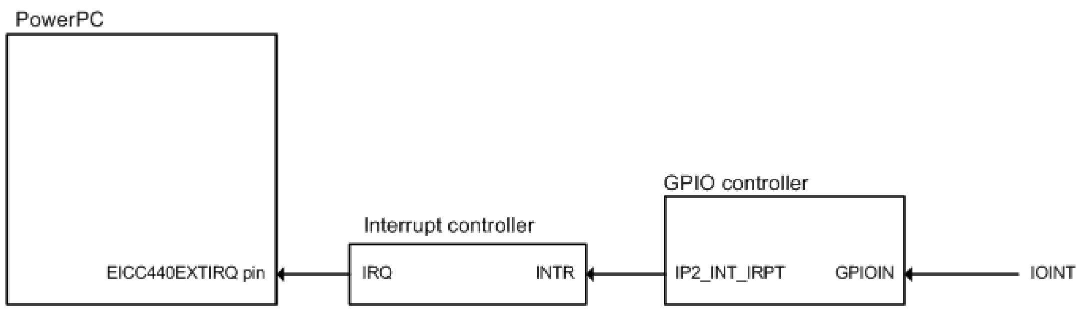


Figure 4-10: PowerPC interrupt hardware

As mentioned, IOINT is a signal of width eight. Each interrupt identified is connected to a bit in the bus. This is shown in Figure 4-11.

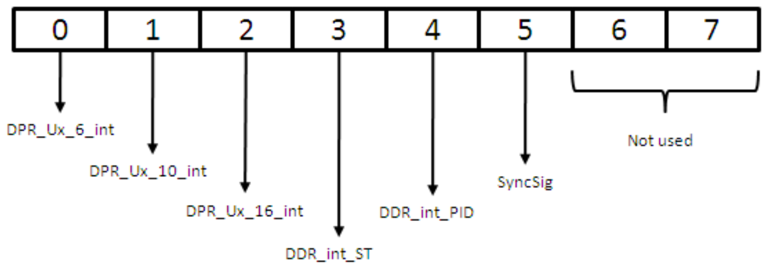


Figure 4-11: VHDL interrupt bus

In order to generate the interrupt if the state is changed or if the PID parameters are updated, the data sent to the DDR RAM via the PCI bus needs to be decoded. It was decided to trigger the interrupt if a certain value is written to a specified position in the DDR RAM. The value decided upon was 0xABCDEF12 at position 0x0FFFFFF4 for the PID parameter change, and 0x0FFFFFF8 for a state change. Should any of these two values be detected at those specific positions, the state value or new PID parameters should be read from the DDR RAM.

4.6 Filter design

As mentioned earlier, the filters in the control loop are essential not to excite critical frequencies in the system. It is also necessary to reject high-frequency noise from the control loop. Three types of digital filters were designed to reject these frequencies: an IIR notch filter, a FIR low pass filter and an IIR band-stop filter. All FIR filters have a linear phase shift in the pass-band, which is proportional to the order of the filter. Where the phase shift is critical to the application, IIR filters are rather used, since the pass-band phase shift is less that those of the FIR filters.

4.6.1 IIR notch filter

A notch filter is a band reject filter with a very narrow band. This filter was selected to filter a very specific frequency from the control, which excites a critical frequency in the mechanical axis. The IIR notch filter was selected due to the minimal phase shift it exerts in the control band. The filter is designed to remove 1.448 kHz with a bandwidth of 200 Hz.

A second order band reject filter is given by [62]:

$$\frac{V_O}{V_{IN}} = \frac{s^2 + \omega_z^2}{s^2 + \frac{\omega_p}{Q_p} s + \omega_p^2} \tag{4.6}$$

where $\omega_z = \omega_p$. Thus,

$$\frac{V_O}{V_{IN}} = \frac{s^2 + \left(2\pi(1448)\right)^2}{s^2 + \frac{2\pi(1448)}{7.24} + \left(2\pi(1448)\right)^2} \tag{4.7}$$

By using the bilinear transformation on the transfer function in MATLAB®, the transfer function of the IIR notch filter can be obtained as:

$$\frac{V_O}{V_{IN}} = \frac{0.9696z^2 + 1.742z + 0.9696}{z^2 - 1.742z + 0.9391} \tag{4.8}$$

The bode diagram of this transfer function is seen in Figure 4-12. In second-order section (SOS) form, the IIR filter can be written as:

$$SOS = \begin{bmatrix} a_0 & a_1 & a_2 & b_0 & b_1 & b_2 \end{bmatrix} \tag{4.9}$$

where:

$$\frac{y(z)}{x(z)} = \frac{a_0 + a_1 z^{-z} + a_2 z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}} \tag{4.10}$$

If the transfer function is quantized with a gain, *g*, the above equation can be written as:

$$y(z)\left[g + b_1 g z^{-1} + b_2 g z^{-2}\right] = x(z)\left[a_0 g + a_1 g z^{-1} + a_2 g z^{-2}\right] \tag{4.11}$$

which can be simplified to:

$$y(z) = \frac{1}{g}\left[x(z)a_0 g + x(z)a_1 gz^{-1} + x(z)a_2 gz^{-2} - y(z)b_1 gz^{-1} - y(z)b_2 gz^{-2} \right] \quad (4.12)$$

which, together with the parameters from the SOS form, are used in the discrete implementation of this filter.

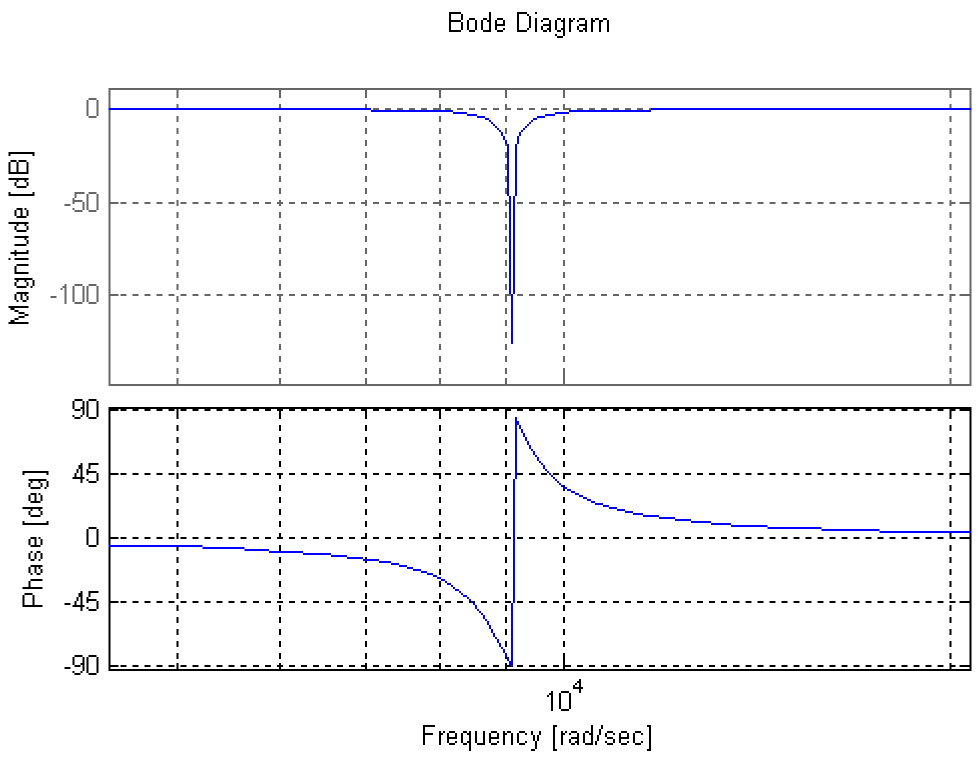


Figure 4-12: Designed IIR notch filter bode diagram

4.6.2 FIR low-pass filter

The FIR low-pass filter is used to filter high-frequency noise from the control band. In contradiction with the notch and band reject filters, the low-pass filter is implemented as a FIR filter. This is because the pass-band phase shift of the low-pass filter is manageable and does not influence the control.

The FIR low pass filter was designed using MATLAB®s "*fdatool*". This tool was used to setup the filter and to generate the coefficients. These coefficients were then implemented in VHDL using the SOS form of the filter. The specification of the filter can be seen in Table 4-6.

Table 4-6: FIR low-pass filter specification

Response type	Low pass filter
Design method	Least squares
Sampling frequency	20 kHz
Pass frequency	250 Hz
Cut-off frequency	2.5 kHz
Order	9

The filter designed using MATLAB® exhibits the magnitude and phase characteristics depicted in Figure 4-13 and Figure 4-14. When the filter is exported to VHDL, the implemented filter should still show these characteristics.

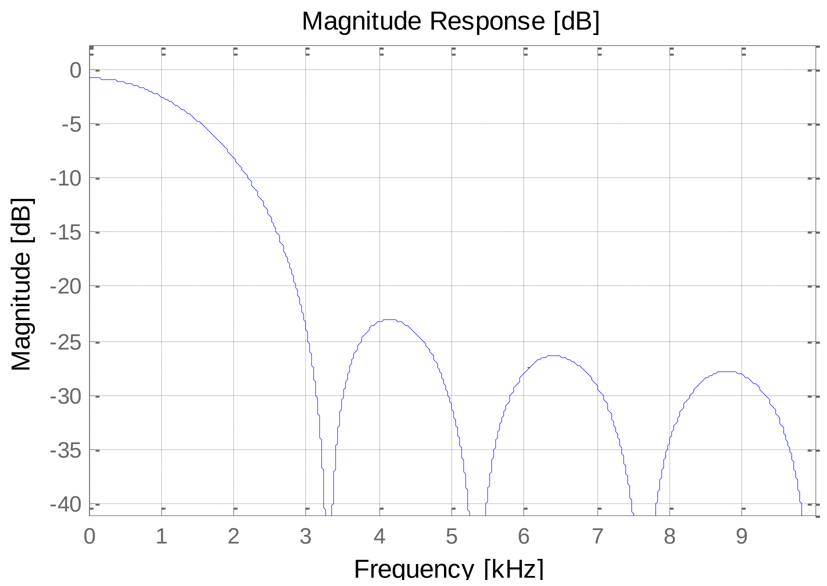


Figure 4-13: Simulated magnitude response of FIR low-pass filter

After the filter coefficients were obtained using MATLAB®, the filter needs to be implemented in VHDL. Before this can be done, the functionality of the filter as implemented in the SOS form needs to be tested. A MATLAB® program was written to test the functionality. The magnitude and phase response of the SOS-implemented filter is seen in Figure 4-15.

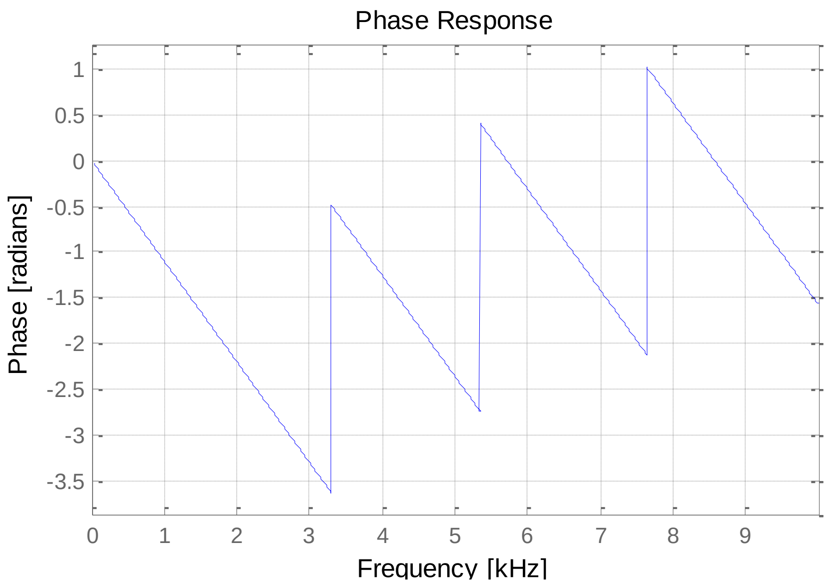


Figure 4-14: Simulated phase response of FIR low-pass filter

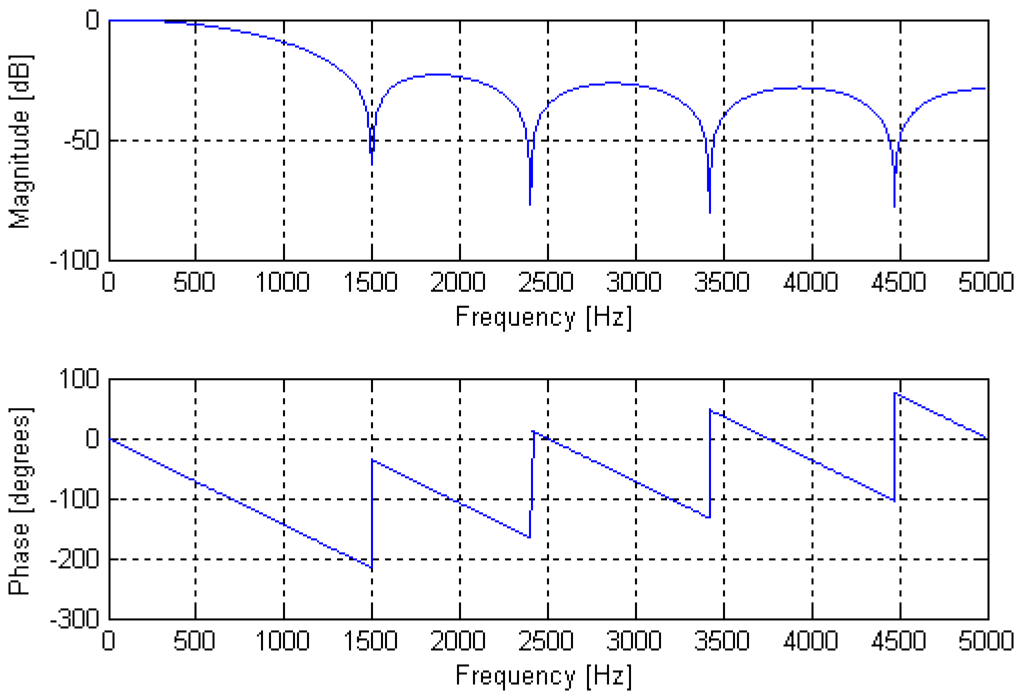


Figure 4-15: True phase and magnitude of implemented FIR low-pass filter

A signal is generated to test the filter. The test signal is a summation of three sinusoidal signals of different frequency, followed by a square wave. The input signal consists of:

- A 50 Hz sinusoidal wave quantized to 14-bit signed values.
- A 1448 Hz sinusoidal wave quantized to 12-bit signed values.
- A 430 Hz sinusoidal wave quantized to 12-bits signed values.
- A square wave quantized to 14-bit signed values.

As can be seen from Figure 4-16 the filter's input and output correlates. The filter has a 7.2° phase shift and almost no loss at 50 Hz, 62° phase shift and 1.5 dB loss at 430 Hz and 201° phase shift and 32 dB loss at 1448 Hz. These results correlate with the magnitude and phase response of the filter.

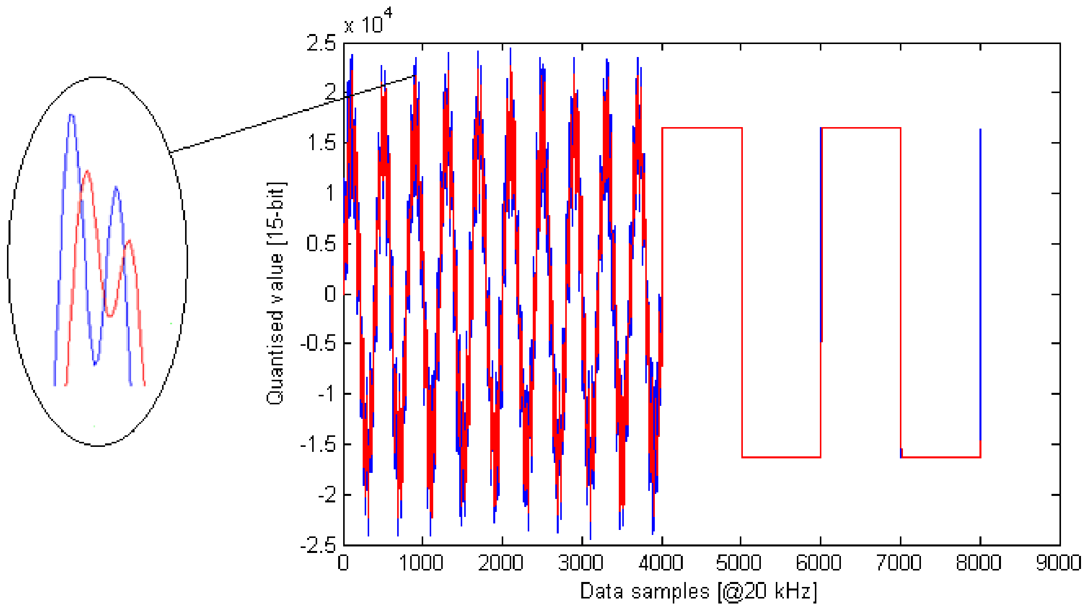


Figure 4-16: FIR low-pass filter response for test input

4.6.3 IIR band-stop filter

The IIR band stop filter is used to attenuate a critical frequency at around 440 Hz in the system. The filter was also designed using the *butter*-function in MATLAB® and implemented using a difference equation. The specification of the band-stop filter can be seen in Table 4-7 and the frequency response of the filter can be seen in Figure 4-17.

The filter was tested by using a 50 Hz sinusoidal wave with a high frequency (430 Hz) noise signal, seen in Figure 4-18. The 430 Hz signal lies within the stop-band of the filter and as can be seen, this frequency is filtered from the input. From the Bode diagram, it is found that at 430 Hz an attenuation of 300 dB is experienced by the signal with a phase shift of around -190°.

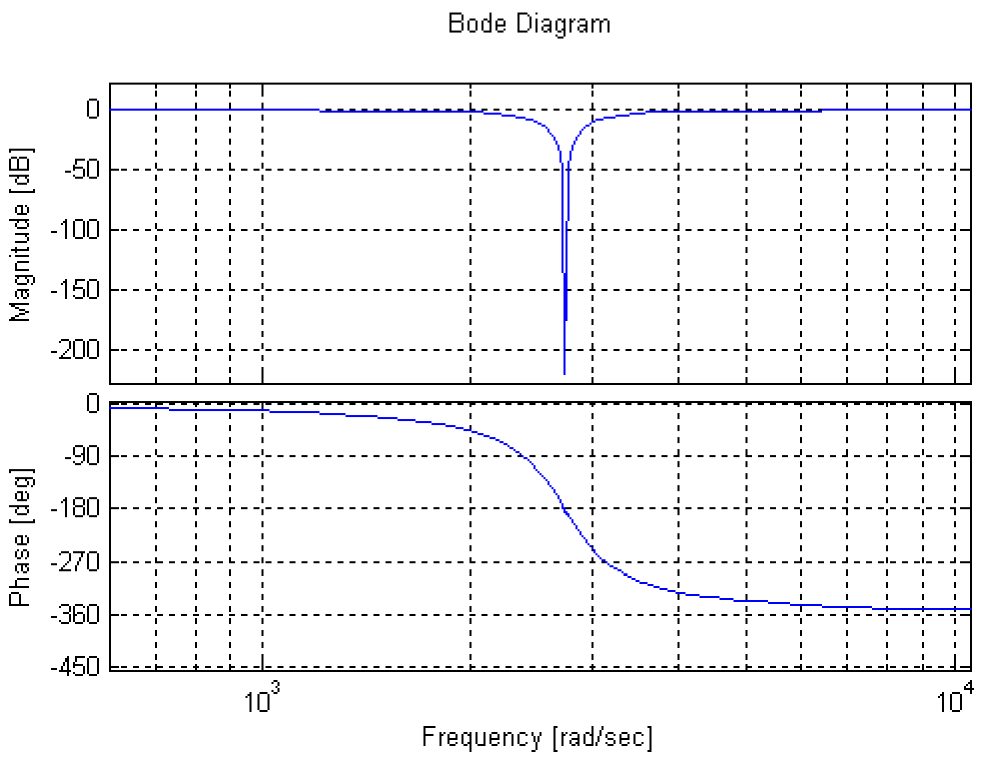


Figure 4-17: Phase and magnitude of implemented IIR band-stop filter

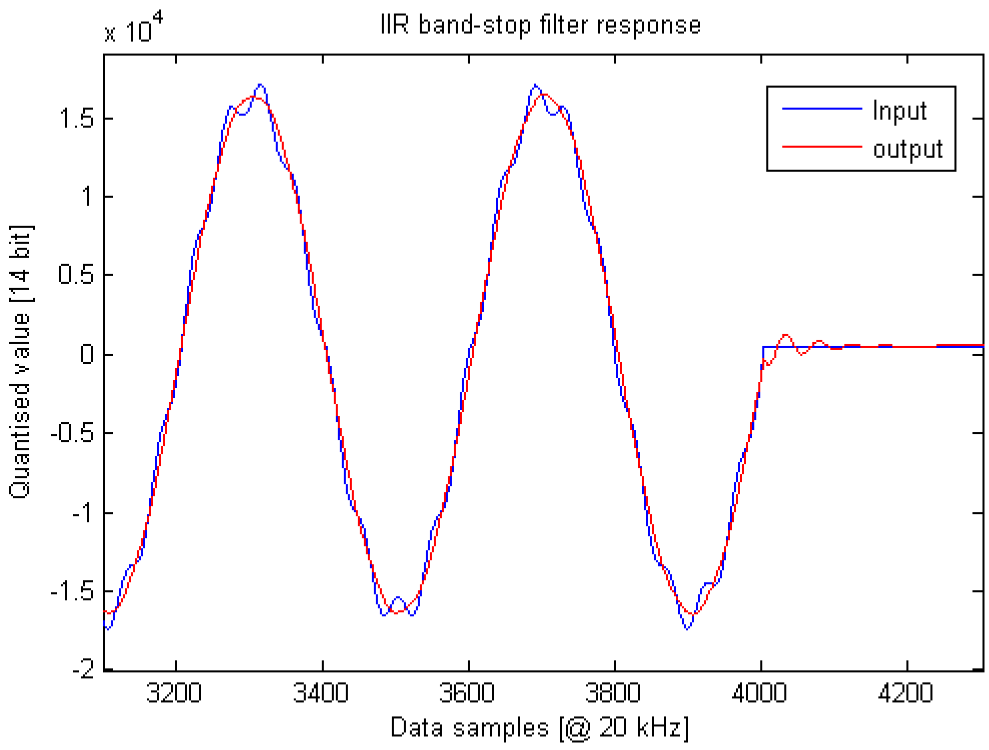


Figure 4-18: IIR band-stop filter response for test input

Table 4-7: IIR band-stop filter specification

Response type	Band stop
Design method	Butterworth
Sampling frequency	20 kHz
Stop frequency	380-500 Hz
Order	2

4.7 Conclusion

After selecting a suitable controller and controller architecture, the necessary components should be implemented. The component design process was based on the software control flow shown in Figure 4-1. The Acromag® code accommodating the VFX70T PMC module includes the general setup of the FPGA and PowerPC. However, additional components are required:

- Dual port RAMs to interface the PowerPC with the FPGA.
- Filters to filter noise from the control band and to attenuate critical frequencies in the system.
- Filter controllers which controls the throughput of the data to and from the filters.
- An interface to the SBC used for data logging and interfacing to the GUI.

It was also seen that additional PowerPC hardware components are also needed, which also includes an FPU and memory interface controllers. Each component added to the PowerPC requires an address in memory. The PowerPC memory space allocation was also designed in this chapter.

The next step was designing the software for the PowerPC and the firmware for the FPGA. The designed embedded state machine is based on the system state machine given in the system requirements specification. Each state defines a certain operation for the controller. The next components designed were the PID control, system timing, interrupts and filters. The next chapter discusses the implementation of the designed components.

CHAPTER 5

Controller Implementation

The hardware designed in chapter 4 now needs to be implemented. This chapter discusses how the different components were implemented on the hardware. Since it is important to verify the functionality of each component, each component is verified as it is implemented. Chapter 6 discusses the validation of each component.

5.1 System simulation

Normally when programming in a certain programming language (such as embedded C on the PowerPC), a certain concept or idea can easily be tested just by compiling the code into assembly code and running it on the designated platform. When programming in VHDL, however, it is not that simple. Since VHDL is compiled and synthesized into hardware, this process takes a significant amount of time. It is thus essential to simulate the code to determine the reaction before it is implemented on the system. This section focuses on establishing a simulation model of the system as a whole, before the implementation of the different components is discussed. Even though the VHDL components can be simulated separately before it is integrated into the system, the need exists to simulate the components integrated into the system, as well as the different components' interfacing with the PowerPC.

ModelSim® is a powerful software package used to perform behavioural simulation of VHDL code before it is implemented. The simulation process can be divided into a few sub-processes:

- Writing the VHDL code to be simulated.
- Writing the test bench used to provide stimuli to the written VHDL code.
- Instantiating the VHDL in the test bench.
- Simulating the test bench with instantiated VHDL code.
- Analysing the results and behaviour of the VHDL code.

From the above list, it can be seen that in order to simulate VHDL code, two basic components are necessary: the code to be simulated and the test bench. In this case, the code to be simulated is the modified Acromag® example design and only a test bench is needed. Designing, coding and using a test bench for such a large design proved to be very exhaustive. The base design needs to be analysed in order to determine the exact functionality and then to accurately reproduce the necessary stimuli. It is also possible to generate a test bench using

ModelSim®. However, ModelSim® only generates stimuli for the PowerPC and if the system is simulated, an interface to the FPGA components is not available.

After some searching, a demo design from Xilinx® was found where the ML507 development kit is used and a simulation model is set-up for the PowerPC and VHDL components. The ML507 development board is a development kit which contains the same VFX70T FPGA as the PMC module used and is thus a suitable platform to try and get a simulation running. The main problem with this idea was that the PMC module and ML507's architectures were entirely different. So many modifications were needed to either design, that it will take a significant amount of time. This defies the point since it will still not be possible to simulate the system as a whole.

If the PowerPC is simulated separately from the VHDL components, the need still exists to simulate data coming from the FPGA. An example is the PID control. Typically, it is not feasible to hardcode a few thousand entries just to simulate the input to the PID control. File handling will simplify this task significantly. The input values can then be generated using a program such as MATLAB® and saved to a file, which can be read later as an input to the simulation. The simulation should thus be capable of file handling. File handling in VHDL is not synthesizable and only possible during simulation, but since the VHDL interface is not available during PowerPC simulation (as stated above), this is futile. File handling is possible in embedded C, but since the PowerPC is embedded on a PMC module, it is difficult to access an external file on the SBC's hard drive. The only way of reading a file using the PowerPC, is to convert the file into a Xilinx® memory file system (xilmfs) file and then to load it into external memory.

This, however, was not as easy as it seemed and a lot of effort went into trying to get a simulation running within the scheduled timeframe. A quick solution was proposed: by using a *pcore*, the problem proposed by the PowerPC ModelSim® simulation not being able to interface with the VHDL components could be bypassed due to the *pcore* being VHDL written code. The *pcore* was set-up with an external bus. This external bus was loaded using the file handling in the ModelSim® simulation. The input data was then sent to the PowerPC core. This model can be seen Figure 5-1. By using this model, the PowerPC could be simulated even without an interface to the VHDL components.

The drawback of this simulation model is that the external VHDL components should be simulated separated from the PowerPC model. When using this model, the external VHDL components should be simulated meticulously to ensure correct operation.

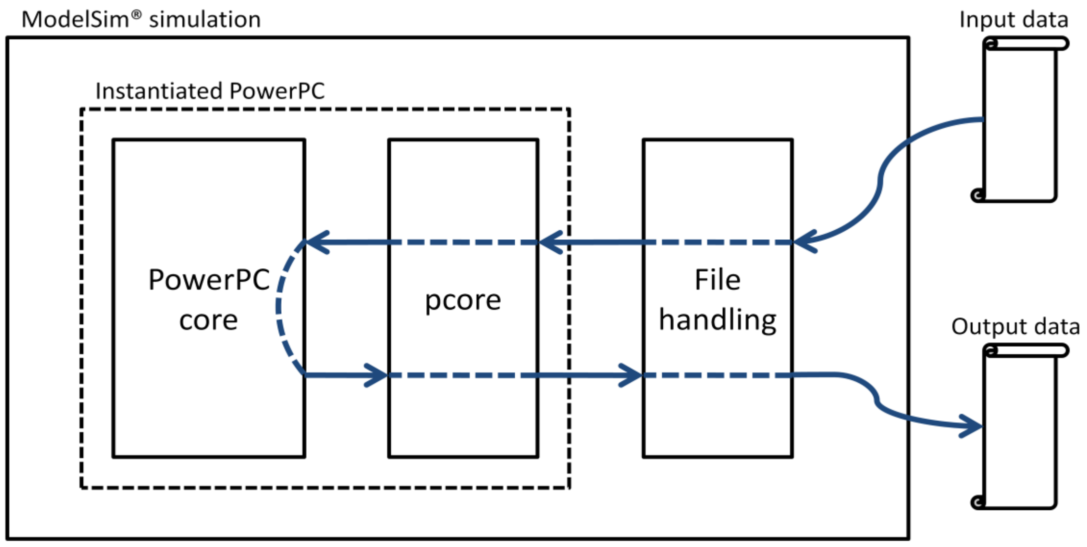


Figure 5-1: ModelSim® *pcore* simulation model

Another problem experienced while trying to simulate the PID control, was that, although the basic functionality behind this model works, the timing between the components was extremely difficult to accomplish, due to the clock domain differences between the PowerPC and FPGA which complicates synchronization. The external data bus is synchronously clocked into the *pcore* using the system clock, which runs at 133 MHz, whereas the PowerPC is clocked at 200 MHz. In the meanwhile, the code running in the test bench is also running at 133 MHz. A state machine was implemented on the *pcore* and test bench in an attempt to solve this problem. The basic idea was to read the file in the test bench. The test bench should then wait until the *pcore* issues a read command. The *pcore* then loads the data received from the test bench onto the SPLB. The state machine on the *pcore* should then wait until the PowerPC is finished with the data, before it lets the state machine in the test bench know it's ready to receive new data. Throughout this process, the PowerPC should be timed or triggered when new data is put on the SPLB.

At this stage, a couple of months have been spent trying to establish a working simulation model for the system and a choice between two options existed: either continue struggling with the timing between the components (which could have taken a significant amount of time) or attempt an alternative route; emulation.

5.2 System emulation

The idea behind system emulation is simple: the student development kit (SDK) used to develop software for the embedded PowerPC has a "*debug on hardware*" option. What this entails it that while the FPGA is configured with its configuration bitstream, the PowerPC code is loaded onto the PowerPC using a JTAG (Joint Test Action Group) interface. The JTAG interface is only an IC debug and probing port. This enables debugging of the software running on the PowerPC, while the code on the FPGA is running.

By using "*printf*" or "*xil_printf*", which is the Xilinx® equivalent "*printf*" (only with less overhead and buffering), it is possible to write data via RS232 to a terminal program using Acromag®'s engineering design kit (EDK). The EDK is a development board which slots into the PMC module instead of the AXM-I/O module and has two RS232 serial ports and a 32-pin header. The basic principle behind the EDK is for debugging while in development. The idea behind the emulation is two-fold; firstly, to debug the PowerPC software and to be able to print data to a terminal using the RS232 serial port. Although emulation encapsulates both firmware and software, and although it is possible to determine an FPGA error using emulation, the firmware should still be simulated using a simulation program (in this case ModelSim®), since an error in VHDL could be difficult to pinpoint and takes a long time to correct due to the synthesization-time. The EDK board can be seen in Figure 5-2.

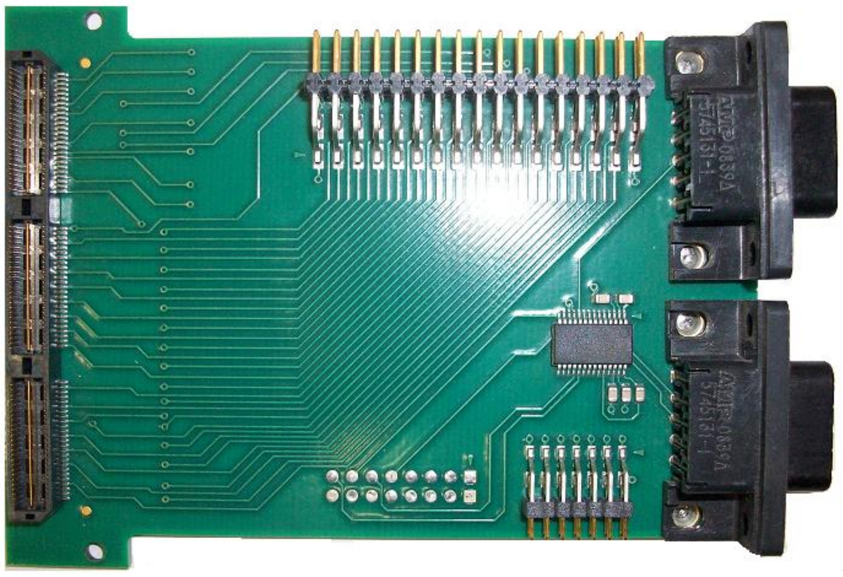


Figure 5-2: Acromag® EDK module

5.3 Control algorithm

The derivation of the PID algorithm in section 4.4.1 is for a digital PID controller with a single pole in the differentiator path at $-\omega_p$. However, once this control was implemented, a mechanical frequency was excited, which induced an audible noise. A second pole was then introduced to reduce the differential gain at high frequencies. The following two sections discuss the two implementations of the PID control.

5.3.1 PID implementation with differential pole at -ω_p

The PID control diagram with the differential pole at -ω_p was shown in Figure 4-7. This pole was moved to a higher frequency to limit the gain of the differentiator at higher frequencies. This control was implemented by using the two 1D second order differential equations (4.5) and derived parameters (4.4) given in section 4.4.1. The equations and derived parameters are shown again for clarity:

$$\begin{aligned} 1D: m\;k\; &= x\;k\; - b_1 m\;k-1\; - b_2 m\;k-2 \\ y\;k\; &= a_0 m\;k\; + a_1 m\;k-1\; + a_2 m\;k-2 \end{aligned}$$

with:

$$\begin{aligned}
a_0 &= K_p + K_I \frac{T}{2} + K_D \frac{4\omega_p}{2T\omega_p + 4} \\
a_1 &= K_P\left(\frac{-8}{2T\omega_p + 4}\right) + K_I T\left(\frac{2T\omega_p}{2T\omega_p + 4}\right) + K_D\left(\frac{-8\omega_p}{2T\omega_p + 4}\right) \\
a_2 &= 2K_P\left(\frac{-T\omega_p + 2}{2T\omega_p + 4}\right) + K_I T\left(\frac{T\omega_p - 2}{2T\omega_p + 4}\right) + K_D\left(\frac{4\omega_p}{2T\omega_p + 4}\right) \\
b_1 &= \frac{-8}{2T\omega_p + 4} \\
b_2 &= \left(\frac{-2T\omega_p + 4}{2T\omega_p + 4}\right)
\end{aligned}$$

The implemented control was tested by replacing U1.6 to 6.6[16] in Figure 4-1 with six single port ROMs, pre-loaded with position data. This position data was then used by the PID controller to perform PID control. The resulting current reference was then sent to a terminal program[17] where it is captured and the response determined by plotting the data. This emulated response was then compared to MATLAB® simulations. Figure 5-3 shows the setup of the hardware used to emulate the PID response.

[16] The DPRs containing the position data.
[17] Such as HyperTerminal™ of Docklite™

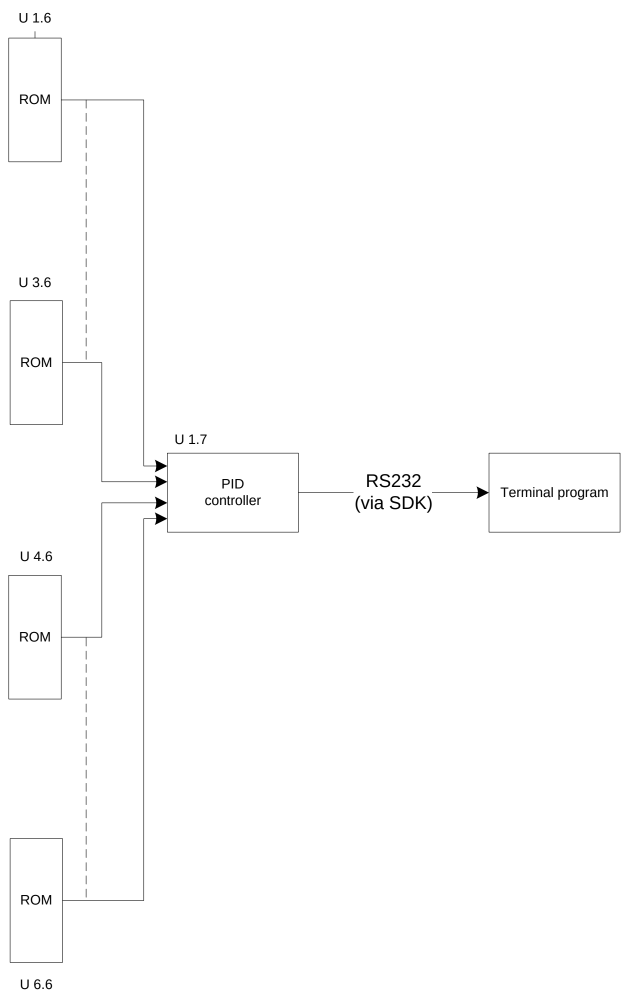


Figure 5-3: Differential pole on -ω_p PID control emulation setup

The first verification of the PID controller was done by comparing the simulated and emulated response of the open-loop system. The input used in both the simulation and emulation is the start-up response of the system where the rotor is held in its reference position (thus at position 0) and dropped. Figure 5-4 shows the comparison between the simulated and emulated response. The pole is placed at 1 kHz on the radial axis and at 2 kHz for the axial axis. The PID parameters[18] were:

- K_P = 20000
- K_I = 3
- K_D = 38

The x-axes shows the number of data points evaluated and the y-axes shows the scaled amplitude of the data points for each graph. Each data point represents an unscaled current reference sent to the power amplifiers.

[18] The emulation was done on a radial PID controller. These were the applicable parameters.

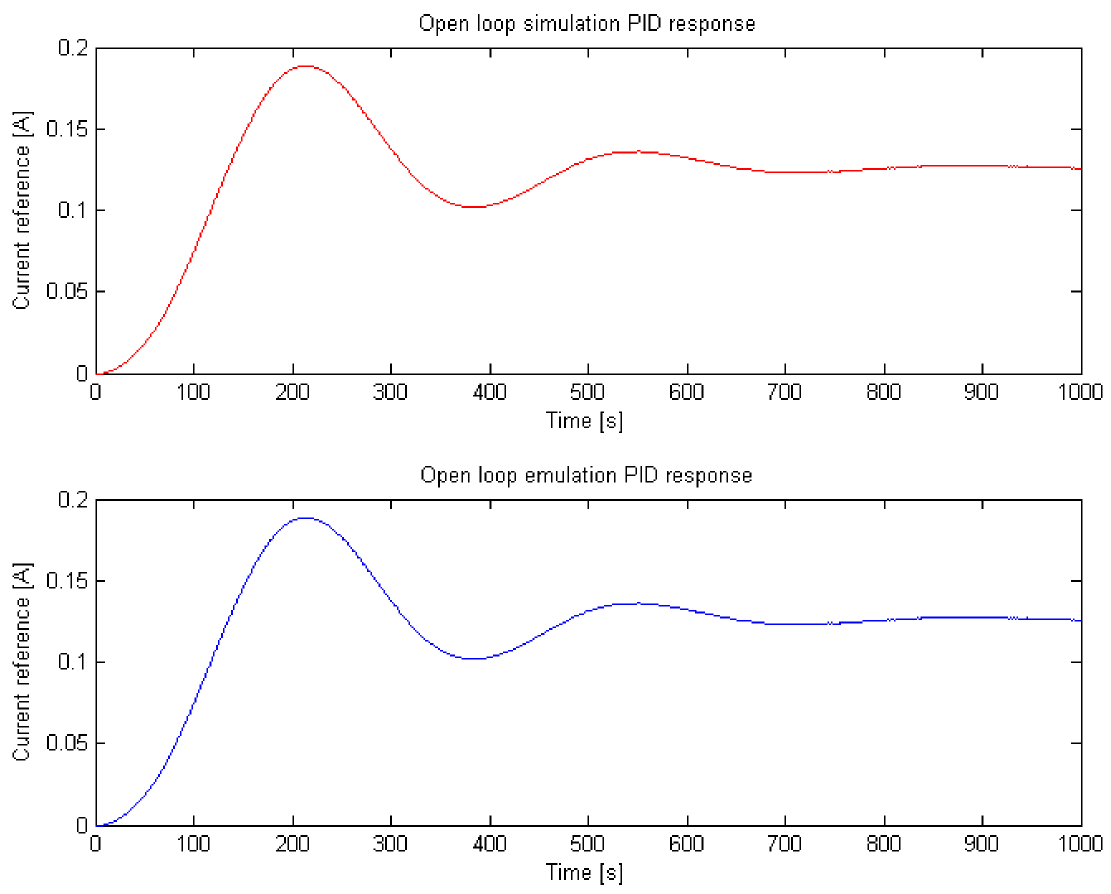


Figure 5-4: Simulated vs. emulated open-loop PID response

The next step was to verify the closed-loop response of the system. This is done by comparing the PID controller used in the emulation of the system, to the PID controller used in the simulation in a closed-loop simulation. This comparison is shown in Figure 5-5 against the normal PID control. The same input and PID parameters were used for the closed-loop response as for the open-loop response.

As can be seen from both types of responses, the designed PID reacts as expected. It is also seen that the digital implementation of the PID controller closely follows its analogue equivalent. Figure 5-6 shows a comparison between the open and closed-loop digital PID response. The simulated response shown was calculated for a closed-loop system, whilst the response of the PowerPC was calculated for an open-loop system. It is clear that the closed-loop response is slower to react than the open-loop equivalent. The negative feedback of the simulated system results in a more stable system, but the response of the PID is slower.

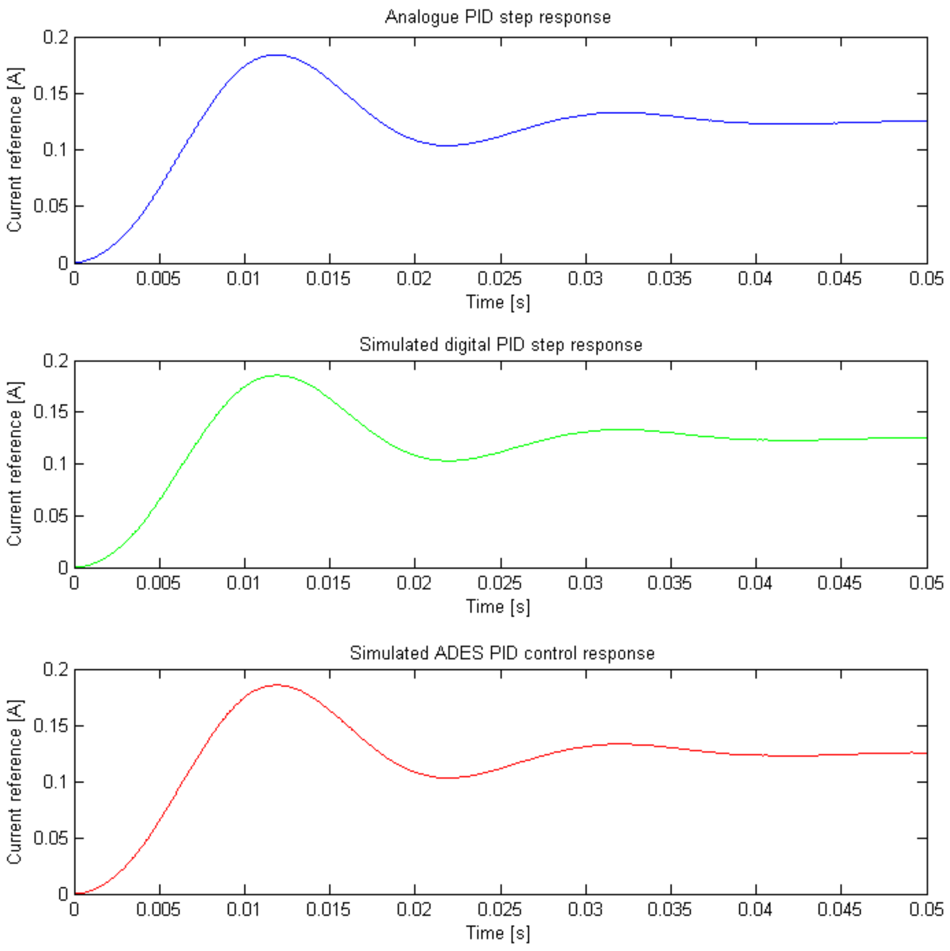


Figure 5-5: Digital closed-loop PID response comparison

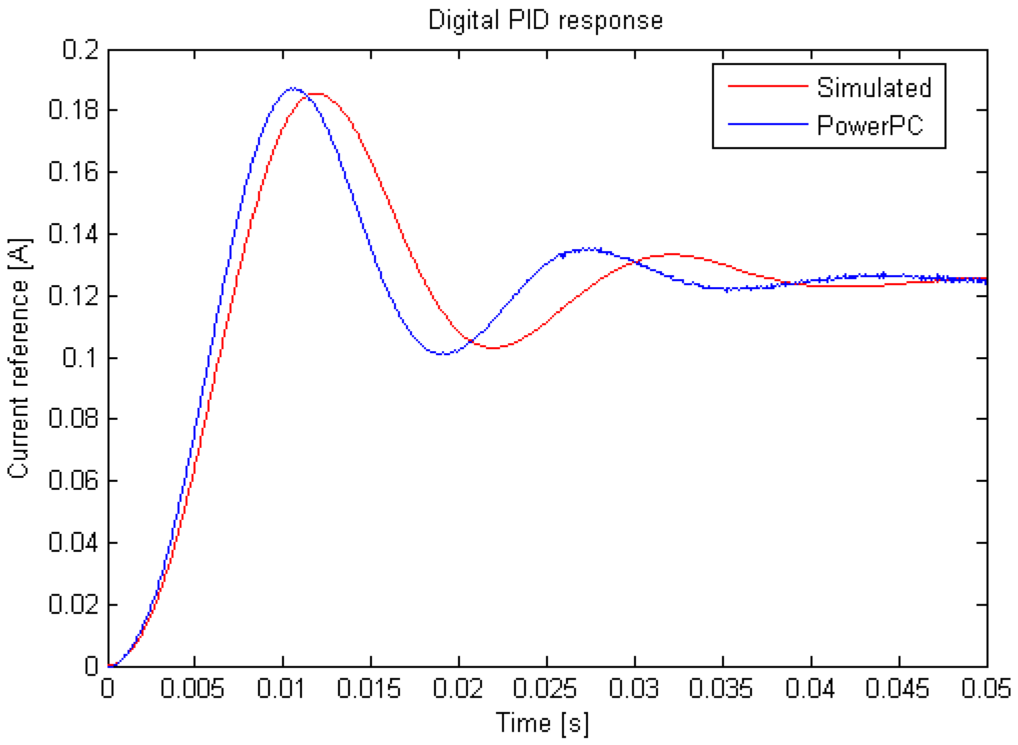


Figure 5-6: Comparison between open and closed-loop digital PID response

5.3.2 Double differential pole PID implementation

Even though the PID control was designed with the differential pole at ω_p, it was necessary to add a second pole in the differentiator path to reduce the differential gain at higher frequencies. The block diagram of the PID controller with the two poles at ω_{p1} and ω_{p2} in the differential-path is seen in Figure 5-7.

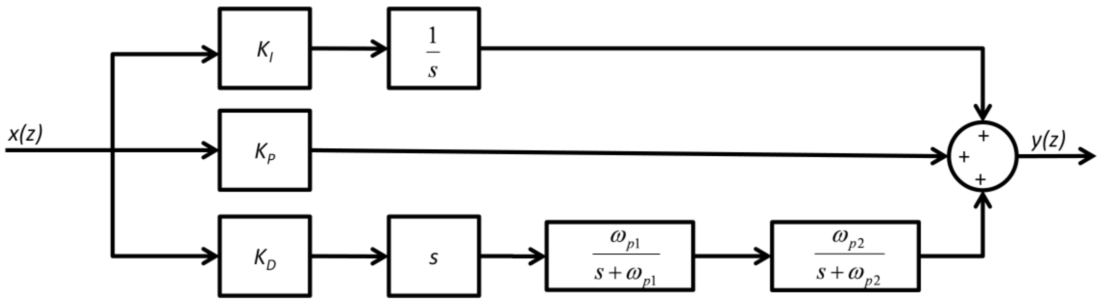

K_I
$\frac{1}{s}$
$x(z)$
K_P
$+$
$+$
$+$
$y(z)$
K_D
s
$\frac{\omega_{p1}}{s+\omega_{p1}}$
$\frac{\omega_{p2}}{s+\omega_{p2}}$

Figure 5-7: Double differential pole PID control block diagram

If the PID control with the shifted pole is given by (from section 4.4.1):

$$D(s) = K_p + \frac{K_I}{s} + \frac{K_D s}{\left(1 + \frac{s}{\omega_p}\right)} \tag{5.1}$$

by adding another pole, this equation is transformed to:

$$D(s) = K_p + \frac{K_I}{s} + \frac{K_D s}{\left(1 + \frac{s}{\omega_{wp1}}\right) + \left(1 + \frac{s}{\omega_{wp2}}\right)} \tag{5.2}$$

Using MATLAB® equation (5.2) is first transformed into the z-plane using the Bilinear-transformation and then, by using the symbolic toolbox, transformed into the form:

$$D(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}}{1 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}} \tag{5.3}$$

which is implemented using :

$$y(k) \;=\; \sum_{n=0}^{3} b_{3-n} e\;\; k-n \;\; - \sum_{m=1}^{3} a_{3-m} y\;\; k-m \tag{5.4}$$

This implementation was verified by using a MATLAB® simulation. The results are seen in Figure 5-8. The position input is a square wave (seen in red in the figure) and the control response is seen in blue. As can be seen, the control response follows the input, except for the overshoot on each transition. While the single additional poles were placed at 1 kHz and 2 kHz for the radial and axial control respectively, the double pole implementation features these poles on both radial and axial PID control. The PID parameters for the simulation were based on the parameters[19] obtained by suspending the rotor using dSPACE®:

- K_P = 20000
- K_I = 10000
- K_D = 55

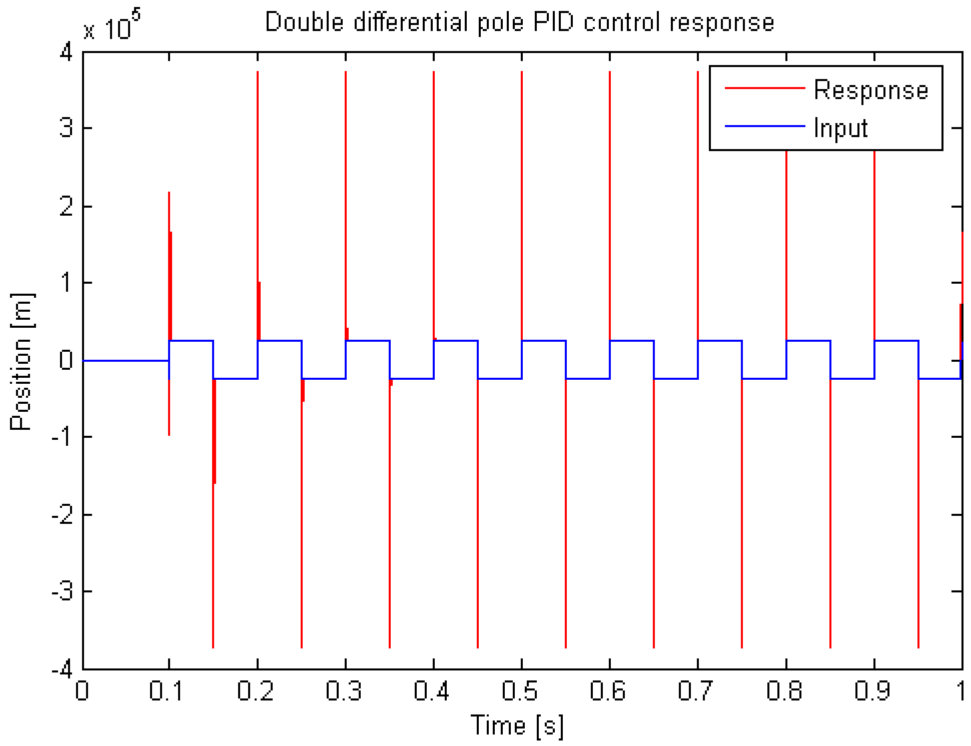


Figure 5-8: Double differential pole PID control response verification

The double pole simulation was not compared to an emulation equivalent since the double differential pole implementation was implemented after the single differential pole implementation was validated. The simulation model was thus implemented and then validated. Refer to chapter 6 for details.

[19] Also done on a radial PID controller

5.3.3 System timing

The system timing is regulated by means of a sync-generator instantiated in VHDL. This sync generator can be seen in Figure 5-9 and Table 5-1 describes the interfaces to the sync generator.

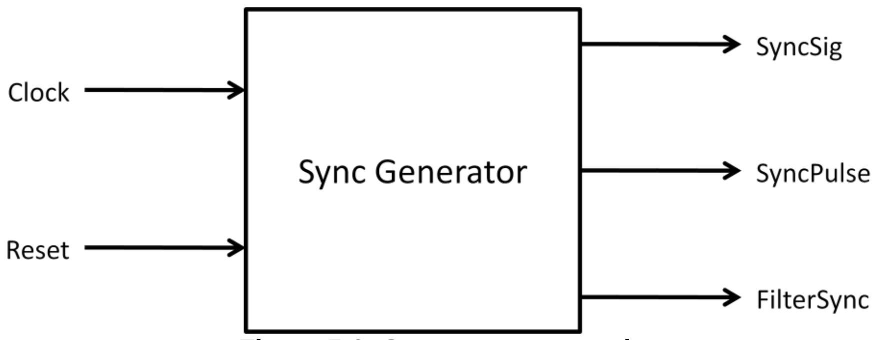


Figure 5-9: Sync generator entity

Table 5-1: Sync generator interface description

Port name	**Type**	**Description**
Clock	Input	This is the reference clock. The synchronous processes, such as incrementing counters, are driven using this clock.
Reset	Input	This resets the sync generator. It is an active high signal. All counters are cleared and the output is initialized to zero when the reset is active.
SyncSig	Output	This sync signal typically has a 50% duty cycle to ensure effective propagation to the different components in the ADES system. This signal is routed to a distribution circuit board from where it is routed to the different components in the system. It is used to sync the power amplifiers and sensor board.
SyncPulse	Output	This is a pulse generated on each rising edge of SyncSig, and is high for one clock cycle. It is used to trigger the communication controller (U1.2, U2.2 and U1.14 to U6.14) and components internal to the FPGA, such as the interrupt to the PID controller (U1.7) which needs only a single-pulse trigger.
FilterSync	Output	This is also a pulse, but generated one microsecond prior to SyncPulse. It is used to trigger the filters controllers (U1.11 to U1.12).

A generic parameter is a parameter set during component instantiation and used in the internal operation of the component. The sync generator has three generic parameters:

- *sync_freq* – This generic parameter is used to specify the frequency of the sync signal in kHz..
- *sync_high* – Specifies the time in micro seconds the sync signal should be high.
- *freq* – The frequency of the system in MHz.

These generic parameters are used to calculate the values the counter must match to toggle the output. For instance, *freq* and *sync_freq* are used to calculate *sync_match* in Figure 5-10 by using:

$$sync\_match = floor\left[\frac{freq \times 1000}{sync\_freq}\right] \tag{5.5}$$

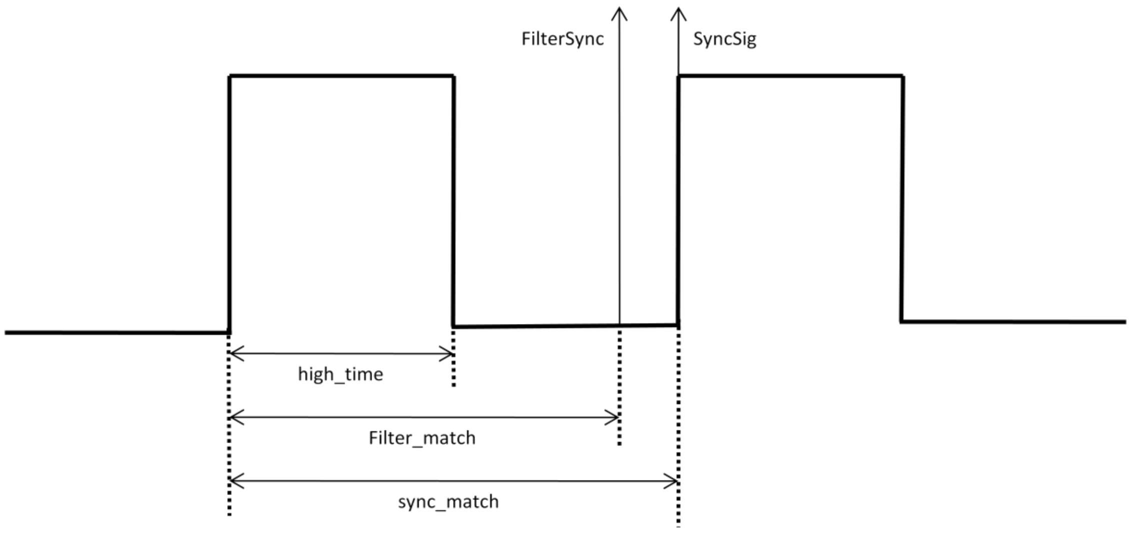


Figure 5-10: Sync generator timing

The sync generator generates the output by incrementing a counter on the rising edge of input clock. If this counter matches a predefined value, the output is toggled to generate the various sync signals. For instance, the 20 kHz interrupt is generated by the counter counting to *sync_match*. For an FPGA running at 133 MHz, *sync_match* is calculated to be: $\frac{133MHz}{20kHz} = 6650$. *SyncSig*, which is initialized to zero when the system is in reset, is inverted for one clock cycle.

5.4 Filter implementation

Units 1.12 to 6.12 in Figure 4-1 depict the filters implemented in the control loop. Each filter is accompanied by a filter controller (U1.11 to 1.12). The filter controller is responsible for supplying the filters with the data to be filtered. It reads the current reference from DPR Ux.10 and provides this data to the filters. The filter controller waits while the filtering process commences. Once the filtering process terminates, a pin is toggled and the filtered data is written into DPR Ux.13. A segment from Figure 4-1 showing the filter controller, DPRs and filters are shown in Figure 5-11. Figure 5-12 shows the filter entity (Ux.12) in Figure 5-11 complete with interfaces. The interface description can be seen in Table 5-2.

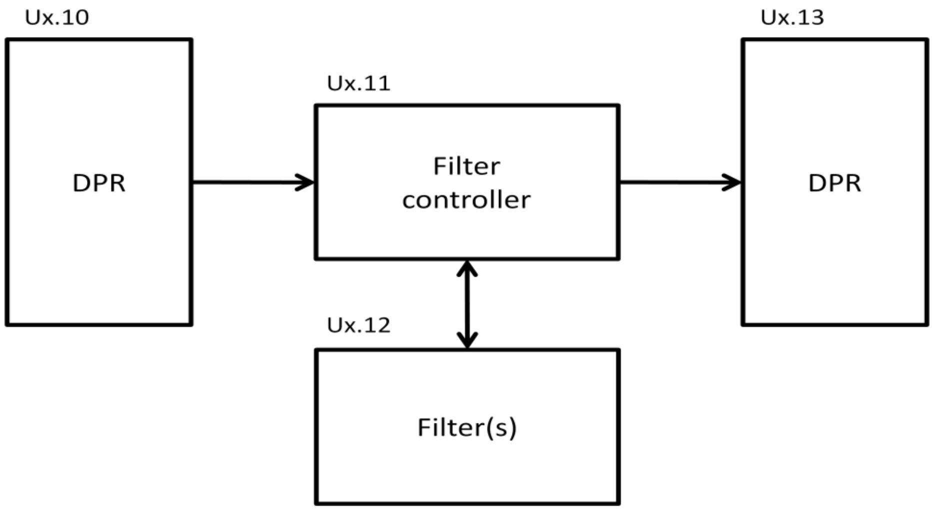


Figure 5-11: Filter and filter controller connectivity

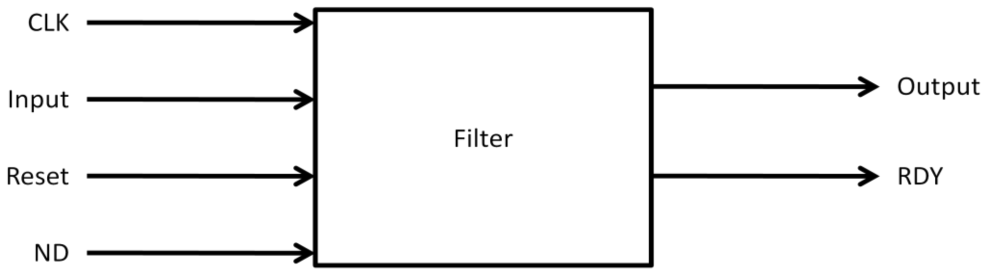


Figure 5-12: Instantiated filter entity

Each filter entity constitutes two or three separate filters which differ for the radial and axial implementation of the filters:

Radial

- Notch filter
- Band-stop filter

Axial

- Low-pass filter
- Notch filter
- Band-stop filter

All these filters were implemented using the SOS as discussed in section 4.6. Since it is possible to simulate the separate entities using ModelSim®, these implementations were verified by using this method. Verification follows Table 5-2.

Table 5-2: Filter entity interface description

Port name	**Type**	**Description**
CLK	Input	The reference clock. The synchronous processes in the filter component are clocked using this clock reference.
Input	Input	The 16-bit data to be filtered is supplied by the filter controller via this port.
Reset	Input	This active high signal resets the filter(s)[20]. It initializes the local variables to zero.
ND	Input	The filter controller initializes the filter(s) when new data is available by toggling this active high pin.
Output	Output	The 16-bit output of the filter.
RDY	Output	The filter controller waits until the filter-process is completed. By toggling this pin, the next state in the filter controller is initialized.

5.4.1 Axial AMB filters verification

To verify the filters in ModelSim®, a setup similar to that in Figure 5-11 is needed, but since ModelSim® is capable of file handling, the two DPRs can be replaced with files; DPRx.10 is replaced by a file containing the input data and DPRx.13 is replaced with a file containing the results of the filter.

[20] The single filter entity contains two or three filters in cascade.

The input file is loaded with a sinusoidal signal containing high-frequency noise. As can be seen from Figure 5-13, the high frequency noise is filtered with a negligible phase shift and a maximum reduction of 10% in the amplitude.

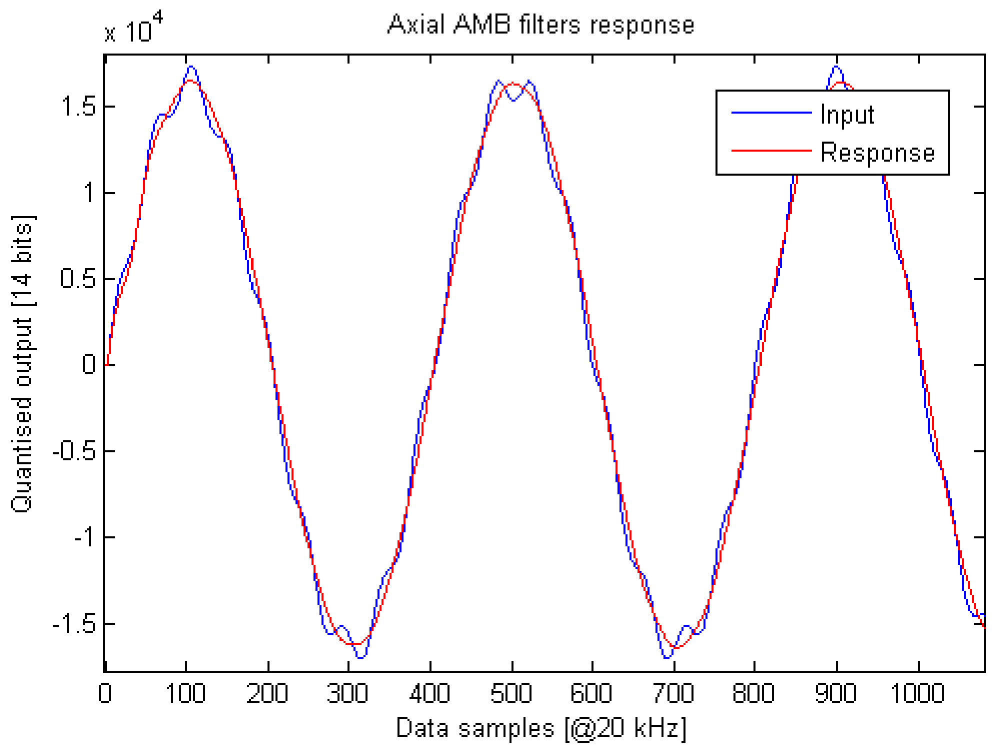


Figure 5-13: Axial AMB filters response verification

5.4.2 Radial AMB filters verification

The verification of the radial filters is based on the same ModelSim® setup as the axial filters. The input to these filters is the same noisy sinusoidal wave as for the axial filters. However, this time the sinusoidal wave is followed by a step input. The reason for adding the step input is to verify correct operation of the filters (specifically the notch filter) even for a step input. If the radial filters' reaction to the step input is correct, the axial filters' responses will coincide.

The filters' responses can be seen in Figure 5-14. As can be seen, the radial filter's response to the sinusoidal input is relatable to that of the axial filters. However, since the radial implementation of the filters consists of two filters instead of three, the maximum loss of the radial filters is less than that of the axial filter. The maximum loss of the radial filters is 5%.

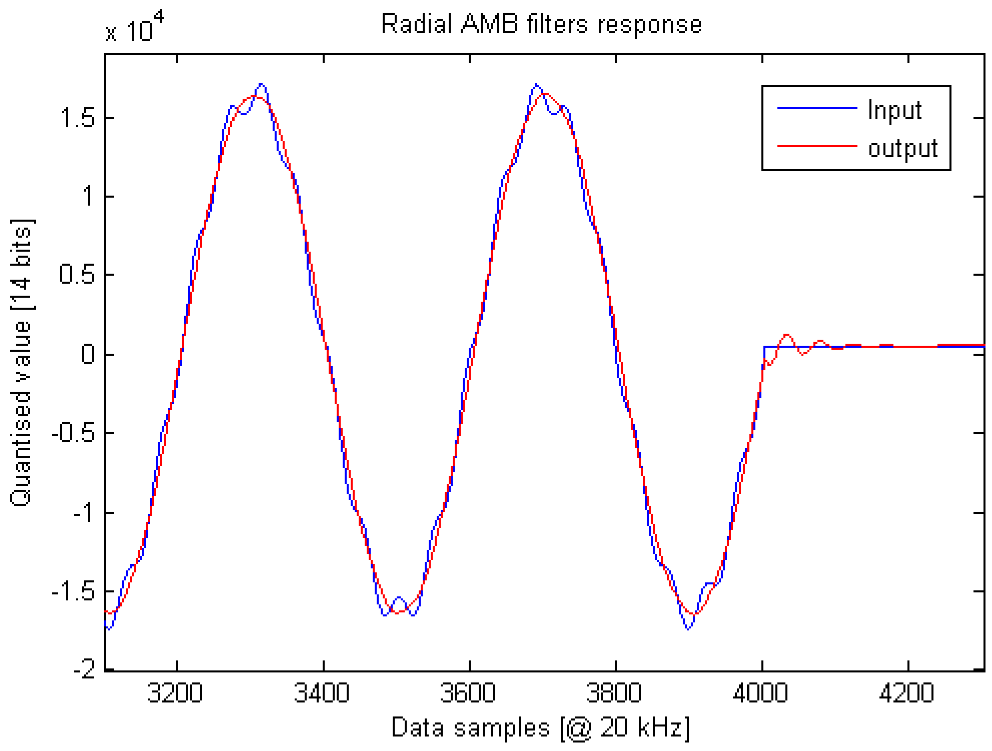


Figure 5-14: Radial AMB filter response verification

5.5 Additional component implementation and verification

All that remains now is a quick discussion on the implementation and verification of the memory and memory interfaces, as well as the interrupt.

The implementation of the DPR was done by multiple instantiations of the DPR entity, shown in Figure 4-2. As can be seen from Figure 4-1, the only DPR set not connected to the PowerPC is U1.13 to 6.13. By connecting these entities to the PowerPC, the interface between the PowerPC and DPRs can easily be verified by using the SDK. The SDK features a memory watch window which can be used to view the contents of the memory. This method is also used to verify the connectivity to the DDR SDRAM. The FPGA connectivity to the DPRs is verified by writing a constant value to all the DPRs and reading the value using the PowerPC.

The design of the interrupt was discussed in section 4.5 and the interrupt was implemented using the sync generator[21]. The verification of the interrupt was done by using system emulation. A *xil_printf* was used to display a message on the terminal program each time the interrupt is triggered. The verification of the sync generator was done by connecting the output ports of the sync generator to an oscilloscope and verifying that each port switches on the correct time interval.

[21] Discussed in 5.3.3

5.6 System execution timing verification

This section is devoted to verifying the system timing. A pre-requisite for the control loop is that it should fit within a 20 kHz cycle. This implies that the functions performed during control should not exceed 50 µs. The PowerPC is clocked at 125 MHz. This implies that one 20 kHz cycle contains 6250 clock cycles. This section measures the number of clock cycles and determines whether the implemented control exceeds the allowed 6250 clock cycles.

Before determining the number of clock cycles each function takes to execute, it is first relevant to discuss how the PowerPC can be optimized to improve performance.

5.6.1 PowerPC optimization

Different ways exist in order to improve the performance speed of the PowerPC. The most common methods are listed below:

1. Increasing the clock frequency.
2. Using a FPU instead of floating-point emulation.
3. Enabling optimization in the SDK.
4. Using cache.

Since increasing the clock frequency induced timing issues on the PowerPC busses, this method was not considered to improve the PowerPC performance. The other methods are briefly discussed before measuring the number of clock cycles the PowerPC takes to execute each function.

5.6.2 FPU vs. emulation

As mentioned in section 4.2.1, in the absence of a FPU, floating-point capability is emulated by the PowerPC. This emulation takes a significant amount of time. For example, a single floating-point instruction such as:

$$T = \frac{1.0}{20000.0}$$

takes 51 clock cycles[22] to perform with the FPU enabled, whereas it takes 1621 clock cycles to perform using floating-point emulation. This results in a performance increase of ±97% on a single floating-point divide.

[22] An explanation on how the number of clock cycles is determined follows in section 5.6.5

5.6.3 SDK code optimization

It is also possible to optimize the code written for the PowerPC using the SDK. Three different modes of optimization are available: low, medium and high. Care should be taken when optimization is enabled and the code is debugged, since it can show unusual behaviour such as unexpected function calls and jumps due to the optimization, but the code will still react in the intended way.

5.6.4 Instruction and data cache

The PowerPC has two types of cache: 32 KB instruction cache and 32 KB data cache [63]. The primary usage of the cache is to improve overall performance of memory systems that are accessed multiple times. Cache devices are normally fast memory components, such as SRAM, which reduces access latency [19]. During each control cycle data is read from different memory spaces. Each time the data is read, it adds to the latency of the system due to each memory space having a setup, read and write time as well as the latency of the bus connecting the memory to the processor. Cache systems store regularly accessed data from slower memory into cache. Not only is the cache fast memory components, it is normally connected to the processor by short busses with low latency.

Figure 5-15 shows a simplified block diagram for the PowerPC 440 [64] [65]. From the block diagram the two types of cache of the PowerPC can be seen:

- The 32 KB instruction cache (I-Cache) controlled by the I-Cache controller.
- The 32 KB instruction cache (D-Cache) controlled by the D-Cache controller.

As can be seen, the instruction unit connects directly to the I-Cache control and the load/store pipeline connects directly to the D-Cache control. If cache is used, the data loaded into the cache, can easily and quickly be used by the instruction unit and load/store pipeline respectively.

The PowerPC cache is enabled by using two functions [66]:

- **void XCache_EnableICache(unsigned int regions);**
  This function enables the data cache for a specific memory region. Each bit in the "*regions"*-parameter represents 128 MB of memory. For example, a value of 0x00000001 caches the first 128 MB of memory (thus 0xF8000000 - 0xFFFFFFFF).

- **void XCache_EnableDCache(unsigned int regions);**
  This function enables the data cache for a specific memory region. The same applies to the "*regions*"-parameter as for **XCache_EnableICache.**

Before the instruction cache can be used, it should first be invalidated using **XCache_InvalidateICache**(void); this invalidates and refills the whole instruction cache.

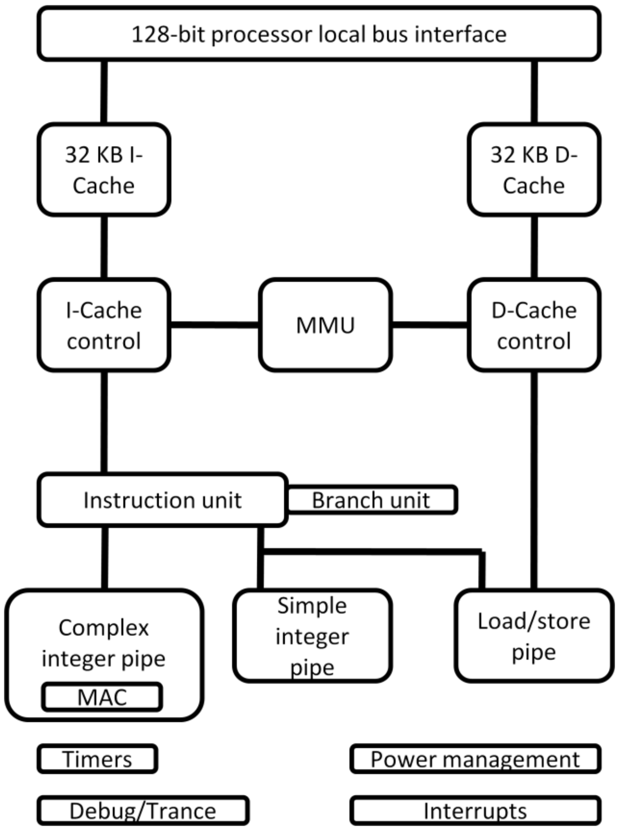


Figure 5-15: PowerPC 440 block diagram

The PowerPC is set-up as such that it uses its internal block RAM (BRAM) to store most of its data and instructions. The VHDL instantiated DPR is used to send data throughout the system and the DDR is used as the interface to the SBC. The software running on the PowerPC, however, uses the internal BRAM. Caching the instructions and data in the BRAM will result in a significant improvement in PowerPC speed.

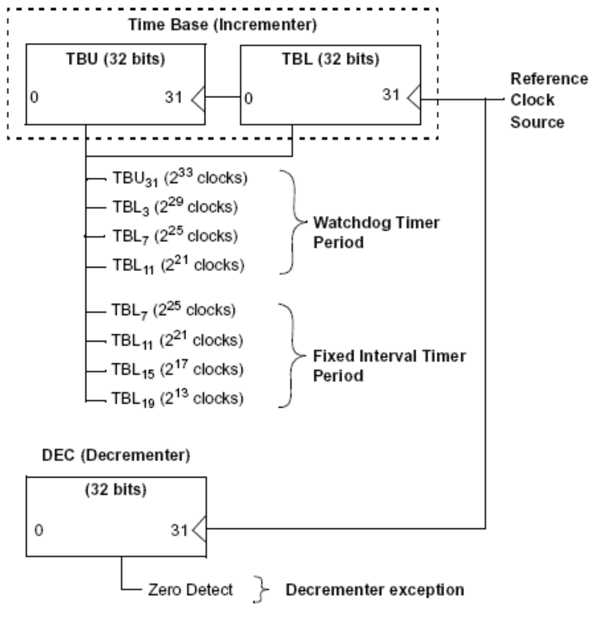
5.6.5 Execution timing

Now that the four ways of improving PowerPC speed have discussed, the timing of the different PowerPC functions can be discussed. Each function will be measured in the following way:

- The system will be evaluated with no FPU, no code optimization and no cache. This will result in the worst case timing. Not all functions will be evaluated without FPU. Only the most significant floating-point instructions will be evaluated without an FPU.
- The first improvement will be to add an FPU, if it was removed for the first test.
- The next improvement is to enable code optimization in the SDK.
- Then the cache is added in two parts: first the instruction cache and then the data cache.

In order to measure the number of clock cycles a function takes to execute, the time base register is used. The time base register is a 64-bit register which increments once during each period of the source clock. The Time Base Lower (TBL) contains the low-order 32-bits and Time Base Upper (TBU) contains the high-order 32-bits [65]. At the start of each function to be measured, the TBL and TBU registers are read with in-line assembler function *mfspr* (move from special purpose register). At the end of the function, these registers are read again and the delta determined. This delta is an indication of the number of clock cycles each function takes to execute. A 20 kHz control cycle contains 6250 clock cycles at 125 MHz.

Figure 5-16 shows the time base register and the relationship to other timer facilities [65]. As can be seen, the time base register is clocked using the reference clock source. The time base register can also be used to set-up the Watchdog and Fixed Interval timer. The Decrementer register shown at the bottom is a register which decrements at the same rate the Time Base register increments. The Decrementer register can be loaded with an initial value and if zero is reached, an exception is triggered



.

Figure 5-16: Time Base register and timer facilities relationship

Table 5-3 and Table 5-4 show the number of clock cycles measured for the most significant functions performed during each 20 kHz control cycle. The table starts off by giving the default amount of clock cycles the function takes. The FPU is then enabled, if it was excluded in the default reading. An even further improvement is then obtained by enabling code optimization in the SDK. Lastly, the instruction and data cache were enabled. Note that, unless otherwise stated, the interrupt was disabled to get an accurate reading.

Even though the method used to measure the number of clock cycles is accurate and adds very little to the overhead[23], some discrepancies can occur. When measuring the number of clocks for a certain function, it can happen that two consecutive measurements vary. This is due to a few reasons:

- Interrupts; the number of interrupts and time at which they occurred.
- The PowerPC 440 has advanced features such as Branch History Table and branch Target Address Cache which are used for instruction pre-fetching and are in a random state at the start of the program. Their initial states influence the program execution time.
- The refresh rate and cycle rate of the external memory influences program execution time.

Due to these factors, the clock measurements were done numerous times and in the cases where the number of clocks varied, the maxima were taken.

It can be seen that by optimizing the PowerPC using the discussed methods have a tremendous influence on the number of clock cycles the functions take to execute. Without optimization, it is impossible to fit the required functions into a 20 kHz control cycle. As can be seen from the table, the run-function takes 265,537 clock cycles to execute without optimization. By enabling optimization, the amount of clock cycles is reduced to 4,880 giving a 98.1% improvement. This results in $6250 - 4880 = 1370$ clock cycles left for minor expansions. Table 5-5 shows the average improvements obtained by enabling each of the optimization methods. As can be seen, enabling the FPU results in a 46.84% improvement. On top of that, enabling code optimization results in a 56% overall improvement, whereas enabling both caches result in an 82.29% overall improvement.

[23] *mfspr* (move from special purpose register) is an inline assembler function.

Table 5-3: PowerPC clock cycle measurement (Part 1)

Function	Description	Default[24]	FPU enabled	Code optimization (High)	I-Cache enabled	D-Cache enabled	I-Cache and D-Cache enabled	Total improvement (%)
Read_Position() ;	Reads the data sent form the position sensor and scales this value into a position (in micron)	15,284	15,475	15,151	10,346	11,815	2,595	83.0
PID_Control ;	Uses the scaled position values from the Read_Position() function to perform PID control for 5-axes.	35,734	6,582	1,094	1,006	554	413	98.8
DPR_Read_Mem() ;	Uses a pointer to the memory interface to read data from the VHDL instantiated DPR into a variable.	N/A[25]	172	86	58	68	27	84.3
DPR_Write_Mem() ;	Uses a pointer to the memory interface to write data to the VHDL instantiated DPR.	N/A	112	18	5	18	5	95.5
DDR_Read_Mem() ;	Uses a pointer to the memory interface to read the state from the DDR if a state change is needed.	N/A	159	83	54	68	19	88.0

[24] No floating-point unit, code optimization and all cache disabled.
[25] Not tested without FPU since the function contains no floating-point instruction.

Table 5-4: PowerPC clock cycle measurement (Part 2)

Function	Description	Default	FPU enabled	Code optimization (High)	I-Cache enabled	D-Cache enabled	I-Cache and D-Cache enabled	Total improvement (%)
DDR_Write_Mem() ;	Uses a pointer to the memory interface to write data to the VHDL instantiated DPR.	N/A	113	18	5	18	5	95.5
AMBs_Normal_ Operation_Run()	The run function of the Normal Operation Mode. This function encapsulates the reading of the position data, performing PID control and logging data.	57,516	23,653	17,094	12,199	13,191	3,595	93.8
Mem_int_handler (void *callback)	This is the function which is executed when the interrupt triggers. The external interrupt port is read and the port is masked to determine the type of interrupt. A specific flag is set for each interrupt.	N/A	1,269	1,234	1,319	1,318	1,226	3.4
AMBs_Normal_ Operation_Run()[26] ;	The run function of the Normal Operation Mode. This function encapsulates the reading of the position data, performing PID control and logging data.	265,537	137,381	87,344	36,804	27,925	4,880	98.1

[26] Interrupt enabled

Table 5-5: Average improvement of system optimization

Enabling the FPU	46.87%
Enabling code optimization	56.00%
Enabling instruction cache	68.21%
Enabling data cache	63.28%
Enabling all cache	82.29%

5.7 System integration

Now that the implementation of the individual components of the main controller were discussed, a discussion on the system integration is necessary.

The hardware architecture of the selected PMC-VFX70T module is shown in Figure 5-17. As discussed in section 3.5.2, this module slots into one of the SBC's PMC sites. The cPCI connectors can be seen on the right-hand side of the figure. The smaller, non-user configurable FPGA[27] is pre-programmed to supply the user with a PCI bus interface via the local bus. Also seen in the figure is that the cPCI connector also supplies the user with a rear-I/O interface via the RTM module. This interface uses LVDS as its electrical signalling system and although it is not used in this project, this interface could prove valuable in future expansions of this project.

Shown in the picture is the hardware architecture inside the larger, user programmable FPGA. This is the hardware used by the PowerPC. The PowerPC 440 core interfaces with the two SRAM blocks and flash memory, which is used to save the non-volatile program code and data. The PowerPC also connects to the external DDR2 memory via the Crossbar, which is a very powerful hardware interface connecting most of the PowerPC hardware interfaces.

The user definable FPGA also interfaces with the front I/O connector. This front I/O connector is used to connect the AXM-D03 interface module. Its 16 digital and 22 differential channels will be used to connect the real-time controller to the external peripherals, such as power amplifiers and sensor drivers.

[27] XC5VLX30

The filters, DPRs and communication controllers are implemented on the FPGA. The position values are received from the sensors on their respective channels via the AXM front I/O connector. This position value is then written into the DPR, from where it is read by the PowerPC. The embedded state machine, as discussed in section 4.3, is implemented on the embedded PowerPC. The rear cPCI connection is the interface between the real-time controller and non-real time operating system running on the SBC.

A state machine identical to the one running on the PowerPC is running on the SBC. A state change from this state machine will trigger an interrupt on the PCI bus and the embedded state machine will change state accordingly. Depending on the state the PowerPC resides in, the position data is treated differently. If the current state is Standby, the data is only logged. If the state is AMBs_Operational, the data is logged and used to perform PID control. The current references from the PID control are then saved in DPR, from where it is filtered and then send to the power amplifiers via the front I/O module.

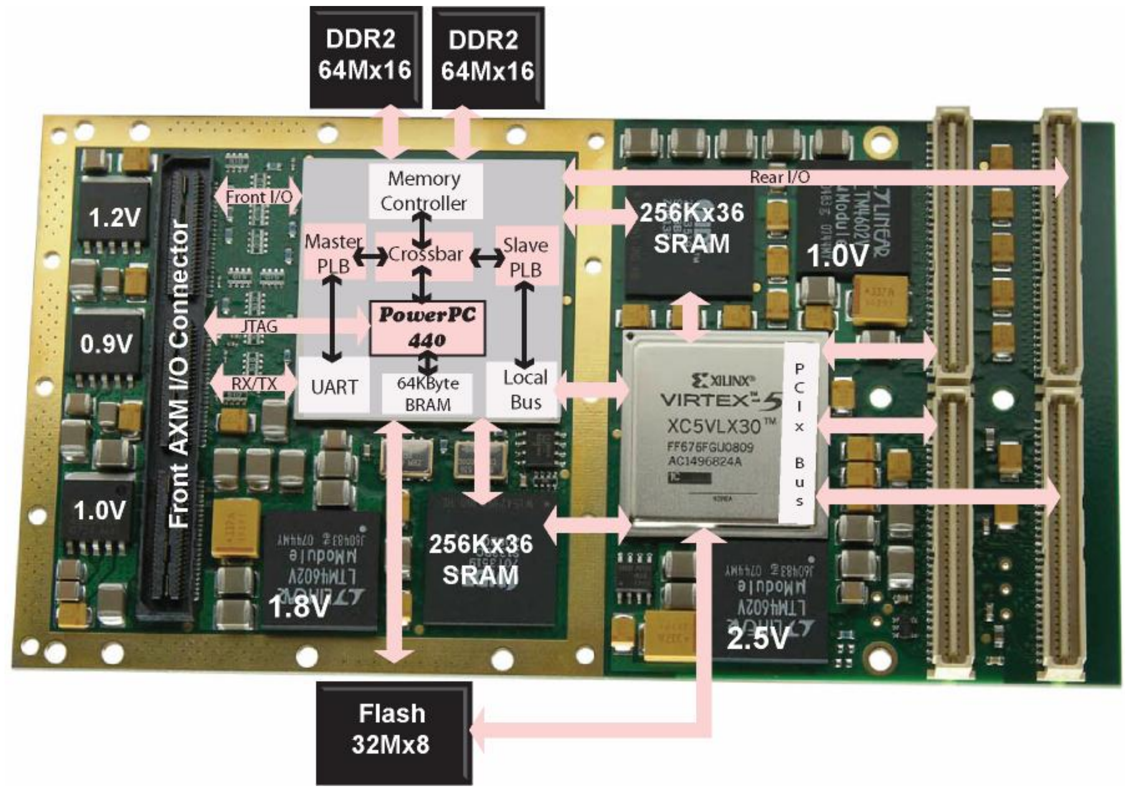


Figure 5-17: PMC-VFX70T hardware architecture

5.8 Conclusion

The components designed in Chapter 4 were verified and implemented into the system as discussed in section 5.7. Despite the issues regarding system simulation, each component could effectively be verified by either a ModelSim® or MATLAB® simulation, or by using emulation.

Even though the PID control was designed to have a differential pole at ω_p, it was necessary to include a second pole in the differentiator path to reduce the gain of the differentiator at higher frequencies. The PID control was implemented on the PowerPC using the first direct (1D) second order equations and the derived a_0, a_1, a_2, b_1 and b_2 parameters. The verification of the control algorithm was done by using a MATLAB® simulation and by emulating the PID control on the system. From the results obtained, it was seen that the response of the PID controller was as expected.

The components in the system is timed by using a VHDL instantiated sync generator. This sync generator, shown in Figure 5-10, is used to realize the timing designed in section 4.4.2. The FPGA is a suitable platform for implementing the sync generator since it is capable of generating signals concurrently. This implies that every component in the system receive the synchronization signal simultaneously.

The filters were implemented either by using the SOS form or by using a difference equation. The control of the filters are managed by a filter controller, which is responsible for supplying the filters with the data to be filtered. The filters were verified using a ModelSim® and/or MATLAB® simulation. It was seen that the required frequencies were filtered out and the phase shifts were marginally small.

The memory interfaces and interrupts were verified using the SDK, the memory watch window and *xil_printf*'s to print values to a terminal program. The timing was verified using the time base registers. It was seen that four methods exist for improving the performance of the PowerPC. A performance increase of up to 82.29% can be obtained by enabling all the methods.

After verifying and implementing the designed components, focus can now be shifted towards evaluating the controller.

CHAPTER 6

Controller evaluation

This chapter provides the validation procedures followed to validate the selected controller. The validation is done by determining the stability and sensitivity of the controller. The controller is also validated by comparing it to the specifications given.

6.1 Evaluation process

The controller was evaluated by using the following techniques:

1. **Discussing controller clock cycles and execution time**
   Section 5.6.5 was dedicated to verifying the different functions in terms of the number of clock cycles necessary for performing each function. The results obtained in this section need to be correlated to the physical system and specifications given.

2. **Validating system time**
   After discussing the controller's performance in terms of the number of clock cycles necessary for performing certain functions, these measured number of clock cycles is then compared to the estimated number of clock cycles given in section 3.2.

3. **Verifying the control response and discussing controller benchmarks**
   The main controller is then compared to the estimated control response given in Table 3-2 to determine whether or not it is possible to implement MIMO control in future iterations.

4. **Determining the stability**
   The stability was determined by adding a step-input to the position data. From the step response, the control performance measures are calculated. The stability of the system can be determined from these parameters.

5. **Determining the sensitivity**
   This is done by adding a frequency sweep to the position reference. The sensitivity analysis indicates the sensitivity of the system to unbalance excitation at each critical speed [67].

6. **Comparing the controller performance to the specification**
   On the completion of these tests, the final system needs to be compared to the specifications given.

6.2 Controller performance

As seen in section 5.6.5, the normal operation[28] of the controller takes 4,880 clock cycles to execute. This implies 39 µs is needed to read and scale the position data, perform PID control and to log the necessary data (the position of the rotor and actual currents of the power amplifiers). The execution timing diagram can be seen in Figure 6-1. The functions shown inside the bar are measured with interrupts disabled. The time shown for normal operation was obtained by measuring all the functions shown in the bar with interrupts enabled. The time was obtained by using the clock cycles shown in Table 5-3 and Table 5-4 and by multiplying the number of clock cycles with the period of the PowerPC clock.

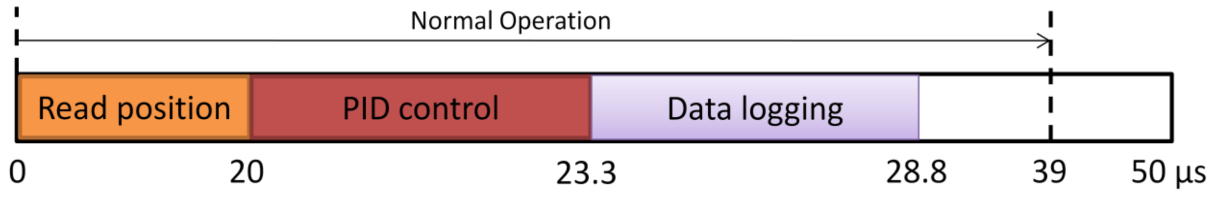


Figure 6-1: Execution timing diagram

By the time the functions performed during normal operation elapses, 11 µs remain for expansion or improvement. This signifies 1,375 remaining clock cycles to implement the functions not implemented. A function such as performing the self-diagnostic test could easily fit within the remaining clock cycles if a simplistic self-diagnostic test is used. Performing motor control, however, will not fit into the remaining clock cycles, since an estimated 47 µs is needed to perform motor control using 3-phase pulse width modulation (PWM) [68].

6.3 System timing validation

Section 3.2 gave an estimate of the number of clock cycles needed to perform the control of the ADES system. From Table 3-2 it was estimated that 1,757 clock cycles were needed to perform the PID control. This is comparable to the 1,094 clock cycles measured for PID control on the PowerPC with code optimization, shown in Table 5-3. However, enabling the cache reduces this number to 414 clock cycles. The initial estimation, even though it was based on the TMS320F/C24x DSP instruction set, gave an accurate estimate to the number of clock cycles needed for PID control.

Table 6-1 shows the comparison between the estimated and actual number of clock cycles. Due to the fact that condition monitoring wasn't explicitly implemented, the exact number of clock cycles is difficult to determine. However, it was seen in section 5.6.5 and 6.2 that the remaining clock cycles are sufficient to implement a basic form of condition monitoring.

[28] Function ***AMBs_normal_Operation_Run()*** discussed in section 5.6.5

Even though the sensitivity analysis wasn't implemented as a real-time function, the sensitivity of the system was determined by adding a sinusoidal wave to the reference position and determining the response of the system to this input. This was possible within the remaining clock cycles, if the built-in sin-function of the PowerPC is not used. The sin-function takes between 11,000 and 13,000 clock cycles to execute, depending on the additional math-functions included in the sin-function (such as a phase shift or amplitude change). However, the remaining clock cycles are insufficient to analyse the frequency spectrum of the data. This makes it impossible to determine the sensitivity in real-time on the main controller. It is clear that the number of clock cycles for sensitivity analysis was greatly under estimated.

The actual number of clock cycles required for communication, shown in the table, is based on the number of clock cycles it takes to write data to the memory, since the remaining communication is handled by the FPGA. Due to the ability of the FPGA to transmit data concurrently, each time the 20 kHz sync-signal triggers, the various data streams are to the different components simultaneously. The communication process on the FPGA is synchronous and each data packet is sent on the rising edge of the FPGA clock. This makes it difficult to determine the exact amount of clock cycles. It is for certain that the number of clock cycles estimated for communication was overestimated.

Table 6-1: Estimated vs. actual clock cycle comparison

	Estimated number of clock cycles[29]	**Actual number of clock cycles**[30]
PID control	1,757	1,094
Monitoring conditions	330	Not explicitly implemented
Sensitivity analysis	60	Not explicitly implemented
Communication	1,925	205

6.4 Control response verification and controller benchmarks

Table 3-2 estimated that 304 accumulations and 484 multiplications per cycle are needed to implement MIMO control. The number of accumulations and multiplications per second (MACS) was estimated as being 1500 MACS. It was also recommended that a 64-bit floating-

[29] Based on the TMS320F/C24x DSP instruction set.
[30] As measured on the PowerPC using the TBL and TBU registers (without caching)

point processor be used for MIMO control. Before it is determined whether or not the selected controller adheres to these specifications, the benchmarking of the selected controller must first be discussed.

FPGAs are not benchmarked according to the number of instructions they can perform per second. This is due to the fact that FPGAs are concurrent devices capable of executing multiple instructions simultaneously. The code implemented on the FPGA is synthesised into hardware, which is not benchmarked in the same way as software. The size and power of the FPGA are indicated by the number of logic cells, slices and flip-flops the FPGA contains. The higher these numbers are, the more components can be instantiated on the FPGA. The different ranges of FPGAs also include different memory sizes and different hard and soft IP cores. Hard IP cores are hardware based IP cores, such as the PowerPC-core whereas soft IP cores are software based IP cores.

Some of these IP cores and auxiliary processors instantiated on the FPGA can be benchmarked according to the number of instructions they can perform. For example: the XC5VFX70T Virtex®-5 FPGA situated on the PMC module is equipped with an embedded hardware PowerPC 440 and DSP-slices. The PowerPC is a 32-bit floating-point processor. The floating-point capability is either emulated or enabled by instantiating an FPU. This FPU is capable of performing 34.8 MFLOPS compared to the 6.2 MFLOPS[31] achieved by using software emulation [69]. The FPU is capable of double precision (thus 64-bit) floating point calculations. The specifications of the PowerPC 440 core are seen in Table 6-2 [70]. As can be seen, the PowerPC is benchmarked using the Dhrystone benchmark version 2.1.

Table 6-2: PowerPC 440 core specifications

Performance (Dhrystone 2.1)	1000 MIPS @ 555 MHz (nominal silicon) 720 MIPS @ 400 MHz (slow silicon)
Frequency	0 – 400 MHz (slow silicon) 555 MHz (nominal silicon)
Power dissipation	2.5 mV/Hz @ 1.8 V
Data bandwidth	Up to 6.4 GB/s via three 128-bit CoreConnect™ bus interfaces.

The XC5VFX70T Virtex®-5 FPGA includes 128 DSP48E DSP-slices. These DSP-slices are capable of 352 GMACS[32] [39]. Various functions are available as IP cores for FPGAs with DSP slices. Functions include: FIR filters, FFTs and multiplier-accumulators. These functions make FPGAs extremely effective when used as co-processors in DSP systems. The only drawback

[31] Both the emulated an FPU results were benchmarked using double precision floating-point calculations
[32] Based on a high speed grade 5VSX95T FPGA

when using FPGAs in a setup such as this is that the floating-point values need to be represented as fixed-point values before it can be used by the FPGA.

From these results, it can be seen that MIMO control is possible using the VFX70T PMC module. The DSP-slices on the XC5VFX70T is capable of extremely high MACS and the estimated 1500 MACS are easily realisable. The FPGA’s ability to perform instructions and calculations concurrently makes this the ideal platform for the implementation of MIMO control. An extremely high cycle rate is also achievable and numerical accuracy can be as high as required. The only drawback is the difficulty representing the floating-point numbers as fixed-point. Although the embedded PowerPC can be used in conjunction with the FPGA to simplify the floating-point calculations, it is not powerful enough to individually perform MIMO control within the required 50 µs. This is due to the FPU only being capable of 34.8 MFLOPS. This is significantly lower that the estimated number.

The signal processing requirements estimated in section 3.3, is compared in Table 6-3 to the performance of the selected controller. As discussed in the preceding paragraph, the specified number of MACS is easily obtainable when implemented on an FPGA with DSP48E DSP-slices. The numerical accuracy of the selected controller corresponds to the acceptable numerical accuracy of 32-bits, but the number of FLOPS is significantly lower than the specified number. The instantiated FPU is not capable of such high floating-point calculations. This could be problematic if more intensive floating-point calculations are required. It can be seen that the selected controller is capable of 20 kHz control cycles as well as floating-point number representation.

Table 6-3: Estimated vs. capable performance comparison

	Estimated	**Capable**
Cycle rate	20 kHz	20 kHz
Number of MACS	1,000 MACS	352 GMACS[33]
Numerical accuracy	32/64-bit	32-bit
Number type	Floating-point	Floating-point
FLOPS	1 GFLOP	34.8 MFLOPS

6.5 System stability

The stability of the system was determined by adding a step with an amplitude of 10 µm as an input position reference. The response to this step-reference is then measured for each degree of freedom.

[33] Theoretical maximum obtainable using the high speed grade 5VSX95T FPGA

The control configuration for this setup is seen in Figure 6-2. The unscaled position data is sent from the sensor. These values are then multiplied by a gain and an offset added. This is needed to convert the position data to a value representing a position in microns. This value is then subtracted from a reference value, which represents the required position where the rotor needs to be suspended. In this case, the reference is zero in order to suspend the rotor in the middle of the sensors. The added step input acts as a changed reference position. It is the responsibility of the PID control to ensure that the rotor is levitated at the oscillating reference position.

To log the response of the system and to determine the stability, both the input and position of the rotor need to be logged. This is done by writing the data to the DDR memory space on each 20 kHz interrupt. The data is then extracted by the SBC and written into a file. MATLAB® is then used to analyse the result and to obtain the settling time (T_s), rise time (T_r), percentage overshoot (*P.O.*), the damping ratio (ζ), natural frequency (ω_n), equivalent stiffness (k_{eq}) and equivalent damping (b_{eq}). These parameters are calculated using the following equations:

$$P.O. = 100e^{\frac{-\pi\zeta}{\sqrt{1-\zeta^2}}} \tag{5.6}$$

$$\omega_n = \frac{4}{T_s\zeta} \tag{5.7}$$

$$k_{eq} = \omega_n^{\ 2} m \tag{5.8}$$

$$b_{eq} = \zeta \cdot 2 \cdot \sqrt{k_{eq} \cdot m} \tag{5.9}$$

where *m* is the equivalent mass of the rotor. The rotor mass is 47 kg and due to the symmetry of the rotor, the assumption is made the mass of the rotor is evenly distributed among the two AMBs. Any inconsistencies in the weight distribution are negligible when compared to the total weight of the rotor. It is thus assumed that the equivalent masses experienced by the two AMBs are 23.5 kg each. The stability test parameters are shown in Table 6-4.

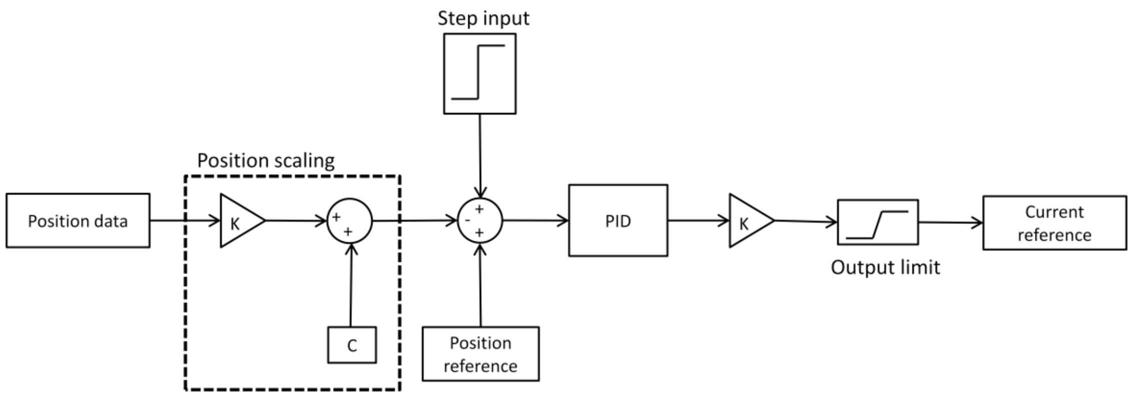


Figure 6-2: System stability control configuration

Table 6-4: Stability test parameters

Step amplitude	10 µm
PID parameters:	
K_P	20,000
K_I	10,000
K_D	30
Differential poles	
$\boldsymbol{\omega}_{p1}$	1 kHz
$\boldsymbol{\omega}_{p2}$	2 kHz
Position reference	0

Figure 6-3 shows the magnetic orientation of the AMBs. Radial AMB 2 is located on the main impeller end of the rotor and radial AMB 1 is located at the cooling impeller end[34]. The AMBs were rotated 45°, since the diagonal orientation of the magnets allows for a more even distribution of the magnetic forces. The positive direction of the z-axis is to the right in the figure. This figure will act as a reference to both the stability and sensitivity analyses. A step response in the positive x-direction of radial AMB 1 will be upwards and 45° right.

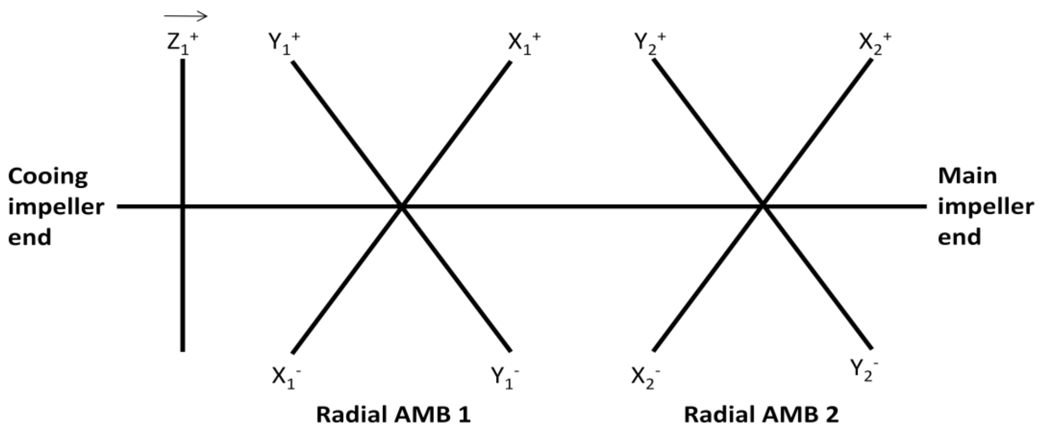


Figure 6-3: AMB magnetic orientation

6.5.1 Radial AMB 1

Figure 6-4 shows the comparison between the dSPACE® and ADES step response of radial AMB 1 along the x-orientation. The two responses are shown separately in Figure 6-5. As can be seen, the noise on the dSPACE® position signal is substantially more than the noise on the ADES position signal. The noise-ripple in the steady state area on the dSPACE® system is 15.5 µm compared to the 2.750 µm noise-ripple on the ADES. This is an improvement of about 82.3%. Table 6-5 shows a comparison between the dSPACE® and ADES control performance measures.

[34] A CAD rendered sectional view of the rotor de-levitation system can be seen in Appendix C.

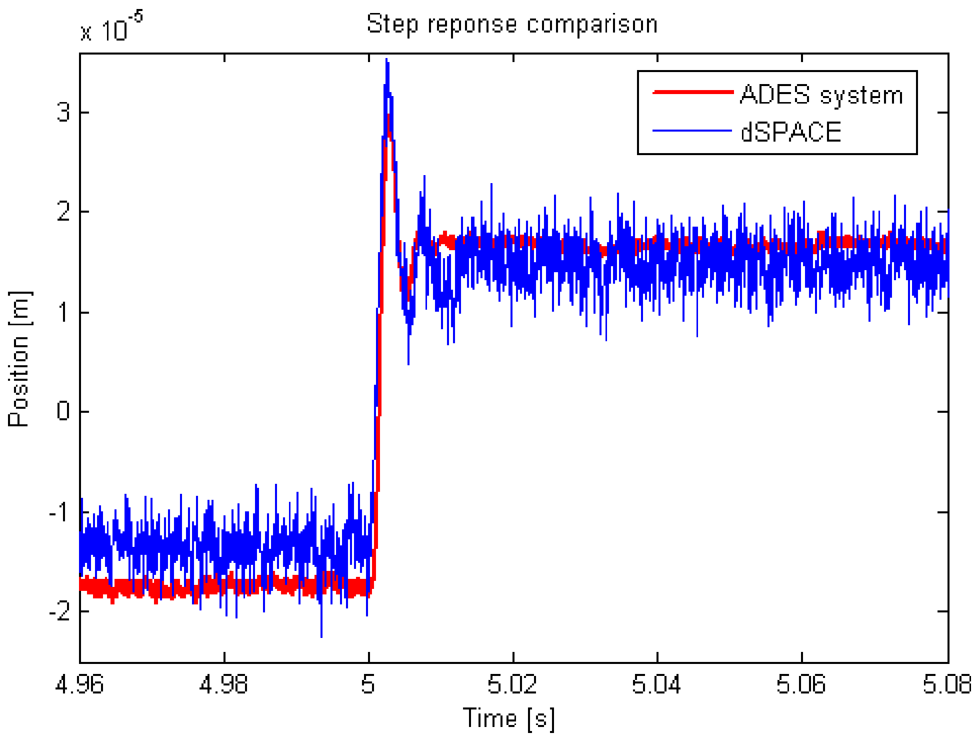


Figure 6-4: Radial AMB 1 x-axis step response comparison

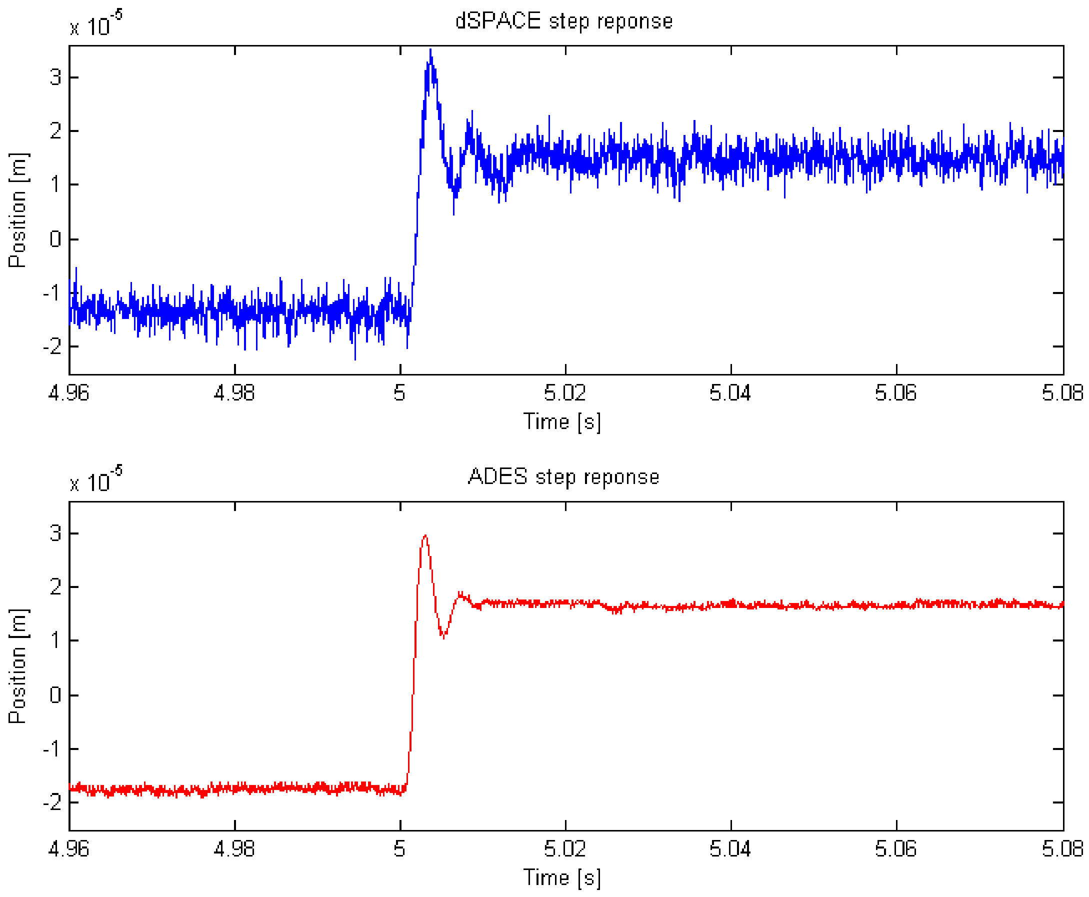


Figure 6-5: Radial AMB 1 x-axis step response

Table 6-5: Radial AMB 1 x-axis control comparison

	dSPACE®	ADES
***P.O.* (%)**	71.8	34.037
T_s (s)	0.024	0.025
T_r (s)	0.007	0.0011
ζ	0.1048	0.3245
ω_n (rad/s)	1,589.268	499.075
k_{eq} (N/m)	59,355,697	5,853,273
b_{eq} (N.s/m)	7,833.33	7,611.336

The step response of radial AMB 1 along the y-orientation is shown in Figure 6-6. As can be seen, the same noise levels exist on both signals as with the step response in the x-orientation. However, a damped oscillation exists in the step response of the ADES system. The damped oscillation can be seen in Figure 6-7 which shows the two responses separately.

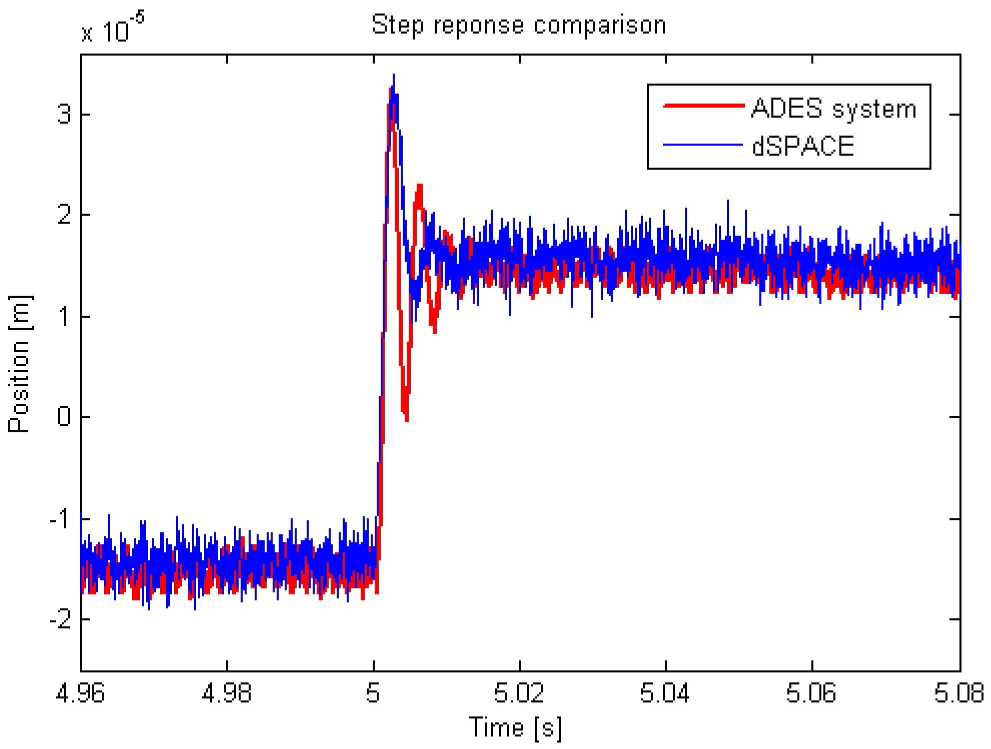


Figure 6-6: Radial AMB 1 y-axis step response comparison

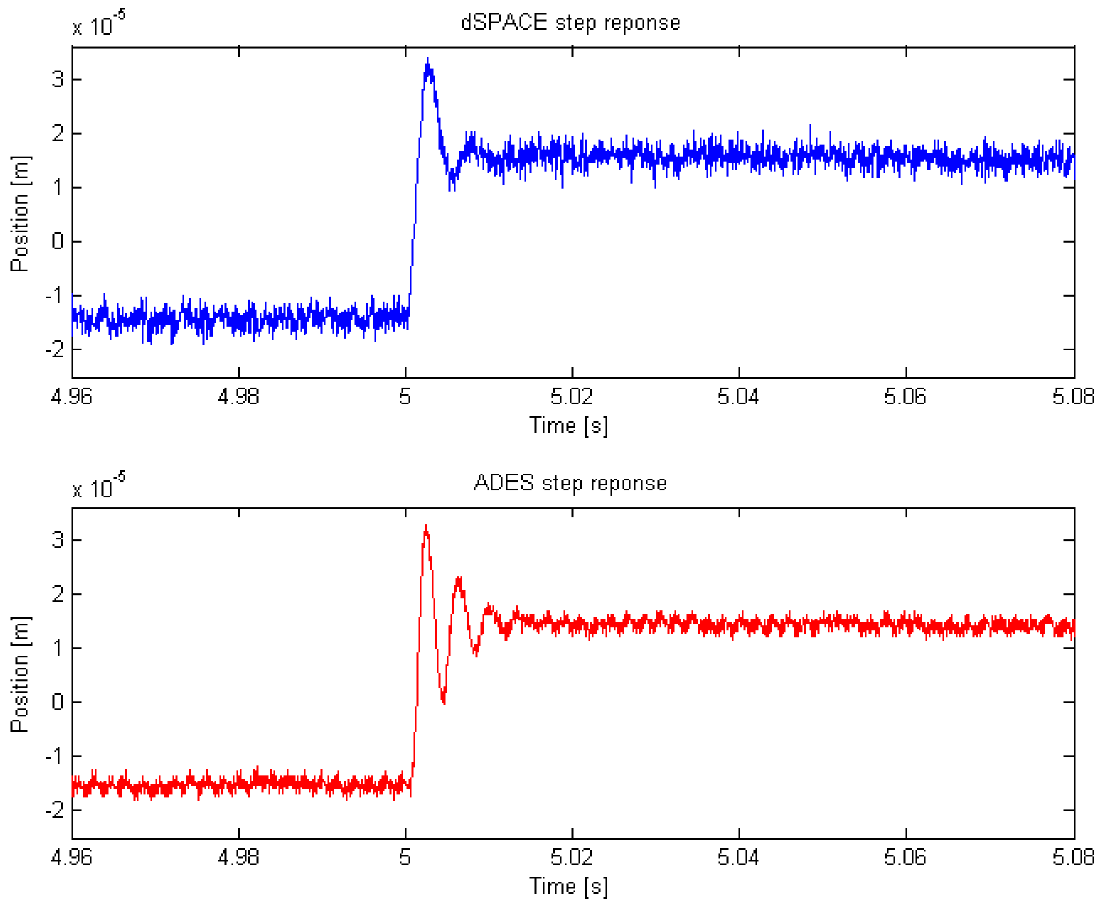


Figure 6-7: Radial AMB 1 y-axis step response

The difference between the two responses can be attributed to a difference in the point of operation between the two systems. The ADES system senses the position of the rotor using the in-house developed iSensor[35], whereas dSPACE® uses eddy current sensors. Calibrating the system to suspend in the middle of the iSensor does not necessarily imply that the system is suspending in the middle of the eddy current censor, and vice versa. The eddy current probes are situated on the backup bearing housing which is fitted with elastomeric material to add damping to the rotor should a rotor drop occur. This implies that the backup bearing housing, and consequently the eddy current probes, are capable of moving a few micrometers if a force is applied. The iSensor, on the other hand, is fixed to stator housing.

When these tests were performed, the ADES sensor data were calibrated until dSPACE® showed the rotor was suspending in the middle of the backup bearing housing. This is done to ensure that the rotor doesn't touch the backup bearings when it is levitated. As mentioned above, the centre of the backup bearing housing does not necessary coincide with the centre of the iSensor. Figure 6-8 shows an example of this configuration. The iSensor is fixed and measures along the orange dashed line. The eddy current probes' measurements are relative

[35] Inductive sensor developed to replace the eddy current sensors.

to the position of the backup bearing housing. This configuration shifts the linear point of operation, illustrated in Figure 6-9, which influences the response of the PID controller when the step is added. As will be seen, this effect is evident in the response of both AMBs along both axes, with the exception of radial AMB 1 x-axis.

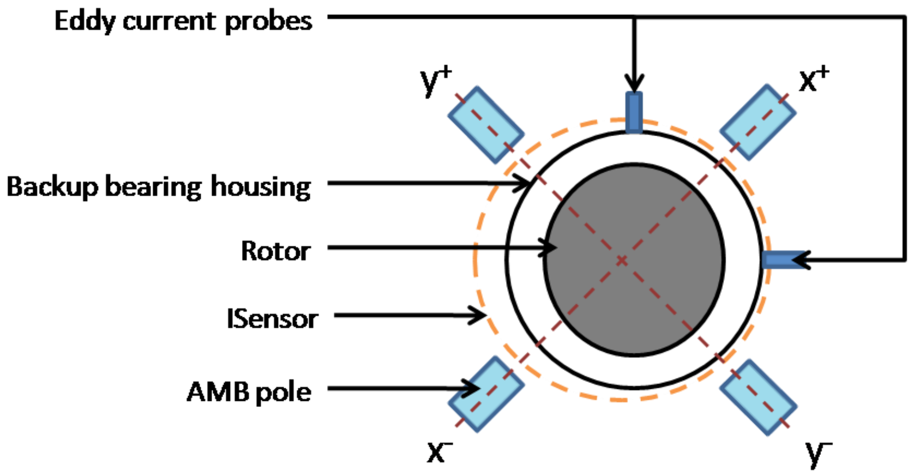


Figure 6-8: Difference in senor positions

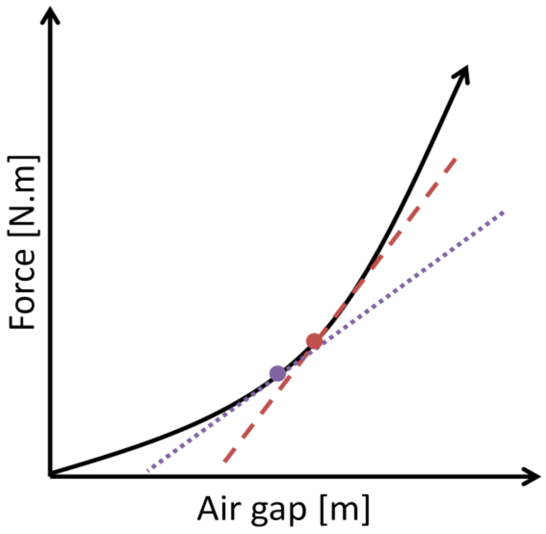


Figure 6-9: Linear point of operation for the two sensors

A comparison between the step responses are shown in Table 6-6. It can be seen that despite the damped oscillation and higher percentage overshoot of the ADES system, the settling time is still the same of the dSPACE® system.

Table 6-6: Radial AMB 1 y-axis control comparison

	dSPACE®	ADES
P.O. **(%)**	50.86	62.21
T_s **(s)**	0.025	0.025
T_r **(s)**	0.0093	0.0061
ζ	0.2104	0.149
ω_n **(rad/s)**	760.49	1,071.0
k_{eq} **(N/m)**	13,591,151	26,955,961
b_{eq} **(N.s/m)**	7520.0	7520.0

6.5.2 Radial AMB 2

This section discusses the step response of radial AMB 2 along the x and y-orientation. The step response of the x-axis is shown in Figure 6-10. Both step responses exhibit a damped oscillation before settling, as is seen from Figure 6-11. As can be seen, the settling time for the ADES is remarkably shorter than that of the dSPACE® system. Table 6-7 shows a comparison between the two step responses.

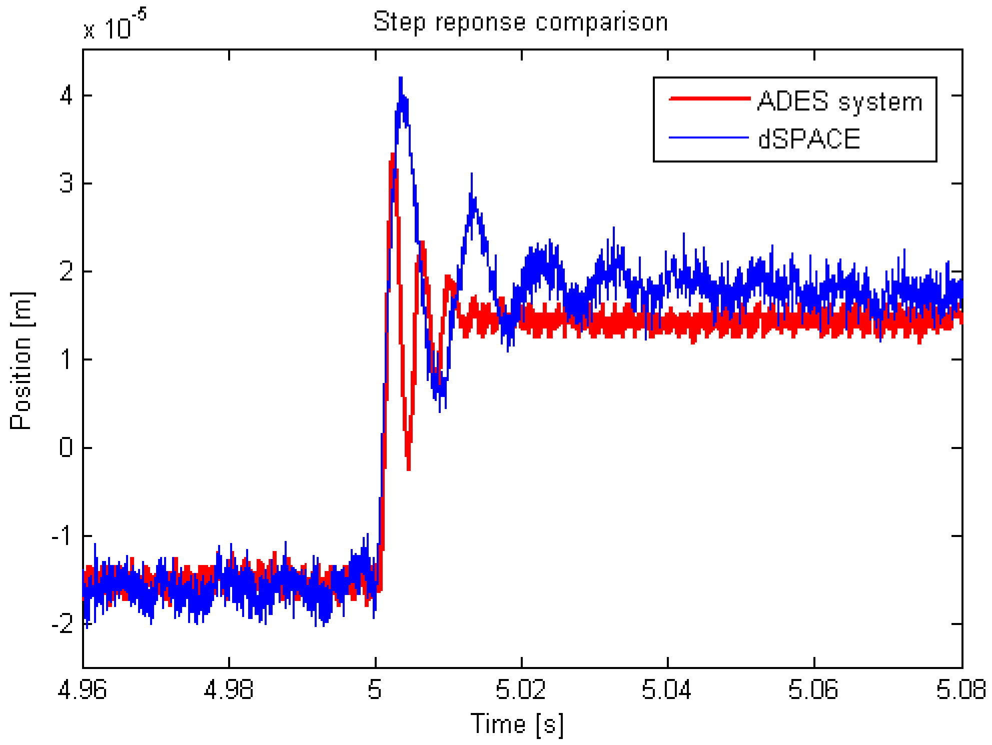


Figure 6-10: Radial AMB 2 x-axis step response comparison

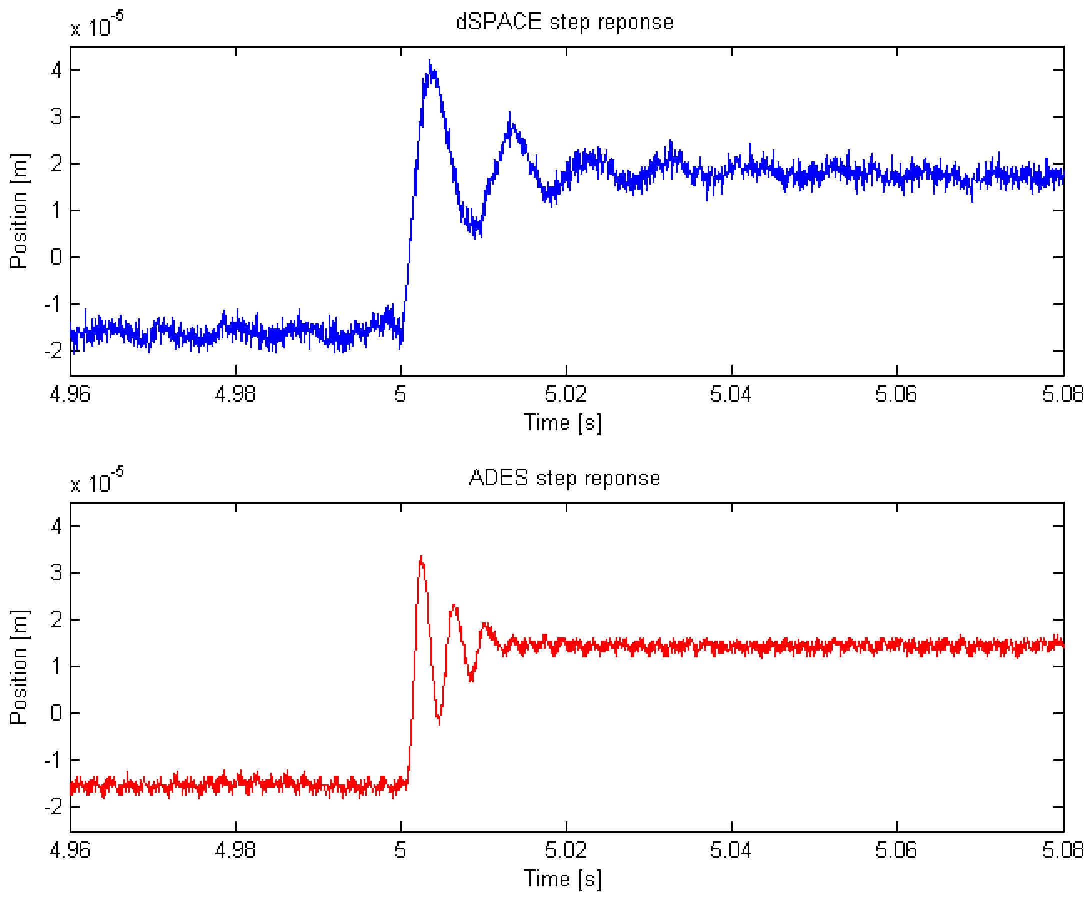


Figure 6-11: Radial AMB 2 x-axis step response

Table 6-7: Radial AMB 2 x-axis control comparison

	dSPACE®	**ADES**
***P.O.* (%)**	74.21	62.99
T_s (s)	0.0998	0.025
T_r (s)	0.0049	0.0055
ζ	0.9451	0.1456
ω_n (rad/s)	424.049	1099.25
k_{eq} (N/m)	4,225,721	28,396,082
b_{eq} (N.s/m)	1883.77	7520.0

The last step response evaluated is the step response of the second radial AMB in the y-orientation. The comparison between the ADES and dSPACE® step responses is shown in Figure 6-12. Figure 6-13 shows the separated step responses. As can be seen, the damped oscillation before settling is still evident in both responses. The percentage overshoot of the ADES step response is about half of the dSPACE®'s step response. The settling time of the ADES is again remarkably quicker than the dSPACE® system. The control comparison is shown in Table 6-8.

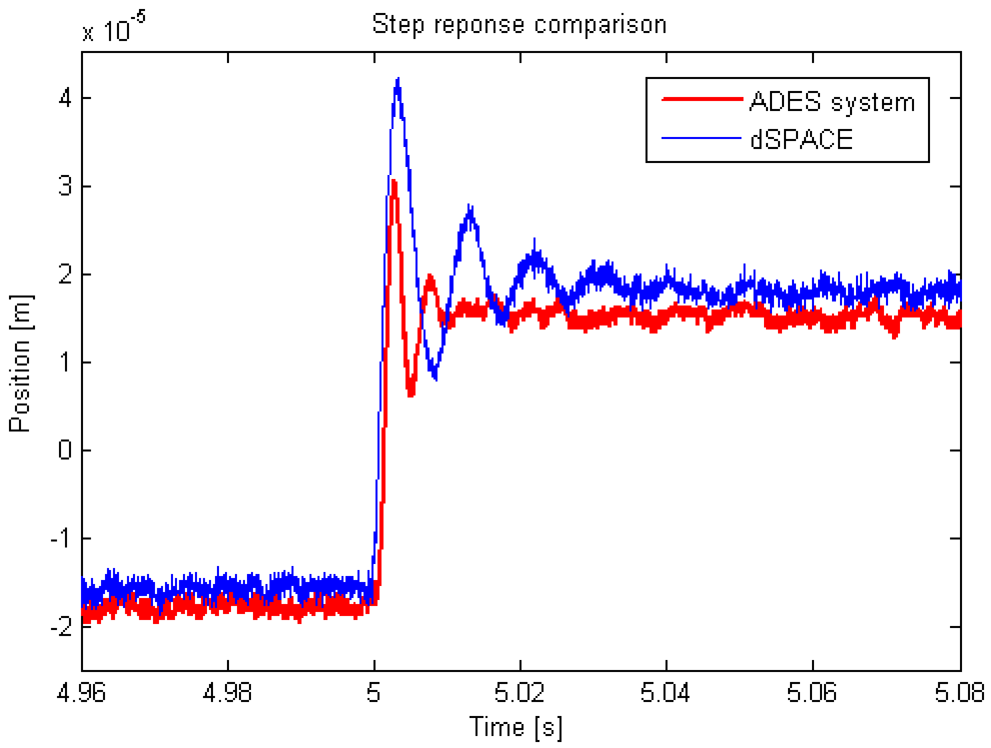


Figure 6-12: Radial AMB 2 y-axis step response comparison

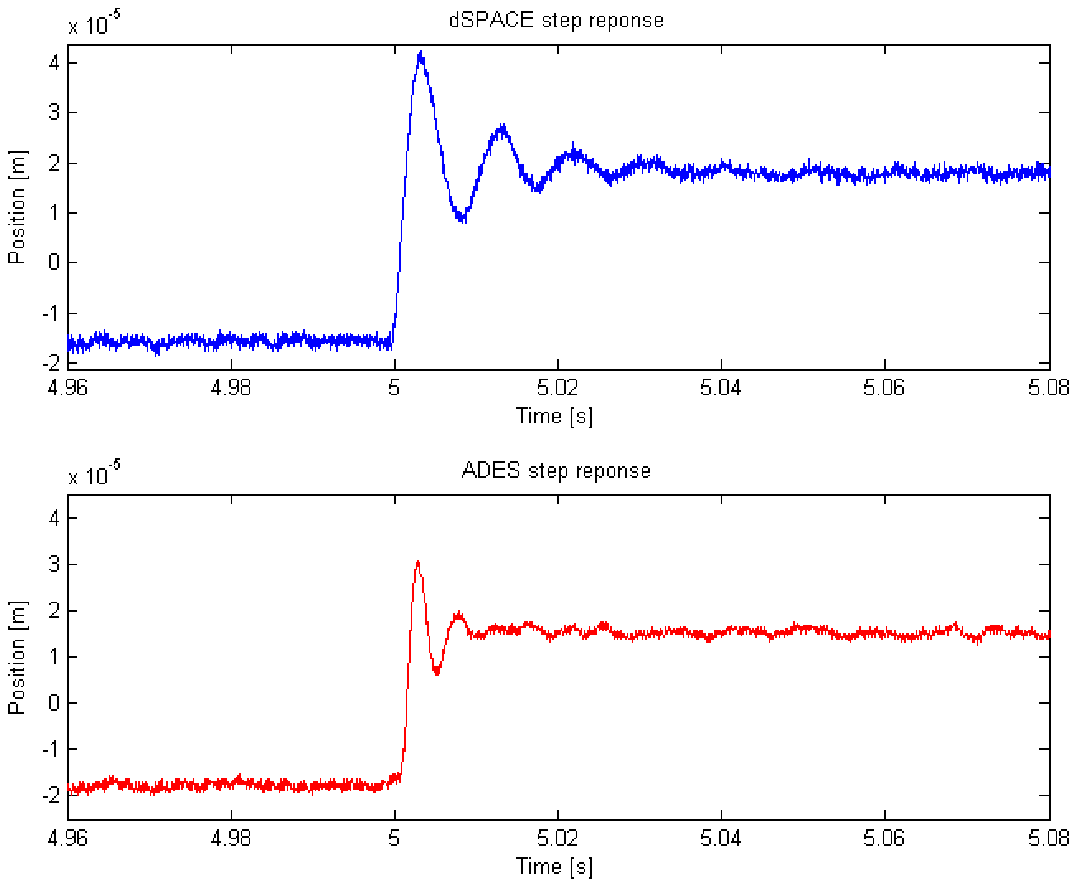


Figure 6-13: Radial AMB 2 y-axis step response

Table 6-8: Radial AMB 2 y-axis control comparison

	dSPACE®	ADES
***P.O.* (%)**	76.04	42.45
T_s (s)	0.049	0.025
T_r (s)	0.0013	0.001
ζ	0.869	0.263
ω_n (rad/s)	939.83	608.063
k_{eq} (N/m)	20,757,090	8,688,923
b_{eq} (N.s/m)	3,836.73	7520.0

6.5.3 Axial AMB

Throughout the above-mentioned tests, the axial AMB was controlled using the dSPACE® system. This is due to the fact that, at this stage, it is impossible to accurately sense the axial position of the rotor using the iSensor. It is thus not possible to control this axis using the ADES system. The hypothesis for this phenomenon is that the machining of the sensing surface causes an error on the iSensor, which fluctuates with the rotation of the rotor. Further study is needed to determine whether this is indeed the case. The axial AMB was also controlled using dSPACE® while performing the sensitivity analysis.

6.6 System sensitivity

The feedback control in AMB systems provides positive stiffness and damping so that the rotor can be stably suspended in the centred position. Without feedback control, the magnetic bearings will have negative stiffness and damping, which would result in an unstable system. The ISO CD 14839-3 standard defines an evaluation method to determine the stability margin of AMB systems [67]. The sensitivity function is defined as the ration of V_R over V_D where V_R is the measured error signal and V_D the sinusoidal disturbance. The positions of these signals are shown in Figure 6-14. The system sensitivity is then calculated, in dB, using equation (5.10):

$$G_s(s) = 20 \cdot \log\left(\frac{v_R(s)}{v_D(s)}\right) \tag{5.10}$$

The ISO CD 14839-3 standard specifies that the sensitivity should be characterized to a frequency larger than three times the rated speed, or a maximum frequency of 2 kHz. A frequency sweep from 2 Hz to 2 kHz in increments of 2 Hz and an amplitude of 20 µm is thus used. Also stated in the standard is that all the axes have to be evaluated independently and that the overall system sensitivity will be determined by the worst rating of any measured axis.

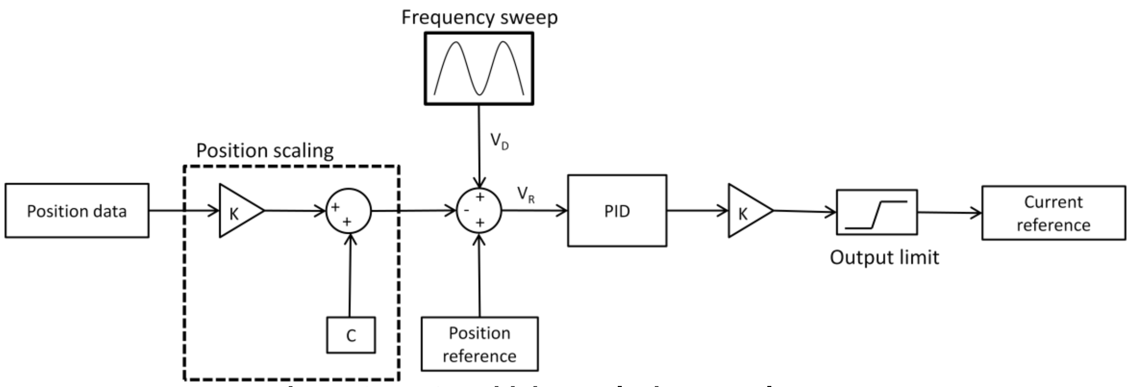


Figure 6-14: Sensitivity analysis control setup

Systems can be grouped into one of four zones [67]:

Zone A: The sensitivity functions of newly commissioned machines would normally fall within this zone.

Zone B: Machines with the sensitivity functions within this zone are normally considered acceptable for unrestricted long-term operation.

Zone C: Machines with the sensitivity functions within this zone are normally considered unsatisfactory for long-term continuous operation. Generally, the machine may be operated for a limited period in this condition until a suitable opportunity arises for remedial action.

Zone D: The sensitivity functions within this zone are normally considered to be sufficiently severe to cause damage to the machine.

The zone of the specific system is determined by calculating the worst sensitivity rating of the individual axes and by comparing it to Table 6-9.

Table 6-9: Peak sensitivity at zone levels

Zone	Peak sensitivity level
A/B	8 dB
B/C	12 dB
C/D	14 dB

The parameters used in the sensitivity tests are shown in Table 6-10. The same PID parameters were used in the sensitivity analysis as in the stability testing.

Table 6-10: Sensitivity analysis parameters

Sinusoidal amplitude	20 µm
PID parameters:	
K_P	20,000
K_I	10,000
K_D	30
Differential poles	
ω_{p1}	1 kHz
ω_{p2}	2 kHz
Position reference	0

Figure 6-16 shows a comparison between the dSPACE® and ADES sensitivity response. As can be seen, the two responses differ greatly. This can be contributed to the non-collocation of the two sensors [71], illustrated in Figure 6-15.

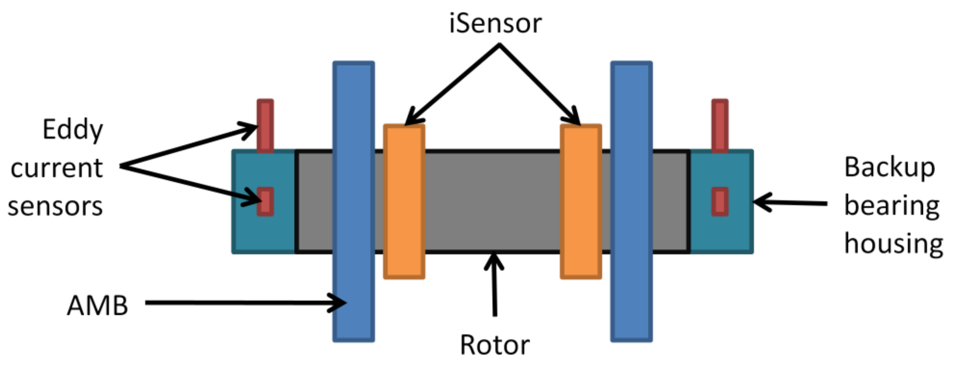


Figure 6-15: Non-collocation of the sensors in the AMB system

The different positioning of the sensors are clearly shown in Figure 6-15: the eddy current sensors measure at the end-points of the rotor, whereas the iSensor measure more towards the middle of the rotor. As the frequency sweep excites the critical frequencies of the system, the response of the rotor varies between sensors. The effect of the bending modes are also measured at different positions on the rotor, which gives different sensitivity analysis results. Figure 6-17 shows the first two bending modes of the rotor. As can be seen, the position of the sensors have a direct impact on the displacement measured. For example, the eddy current sensors measure a small displacement for the first bending mode, whereas the iSensor measures a large displacement.

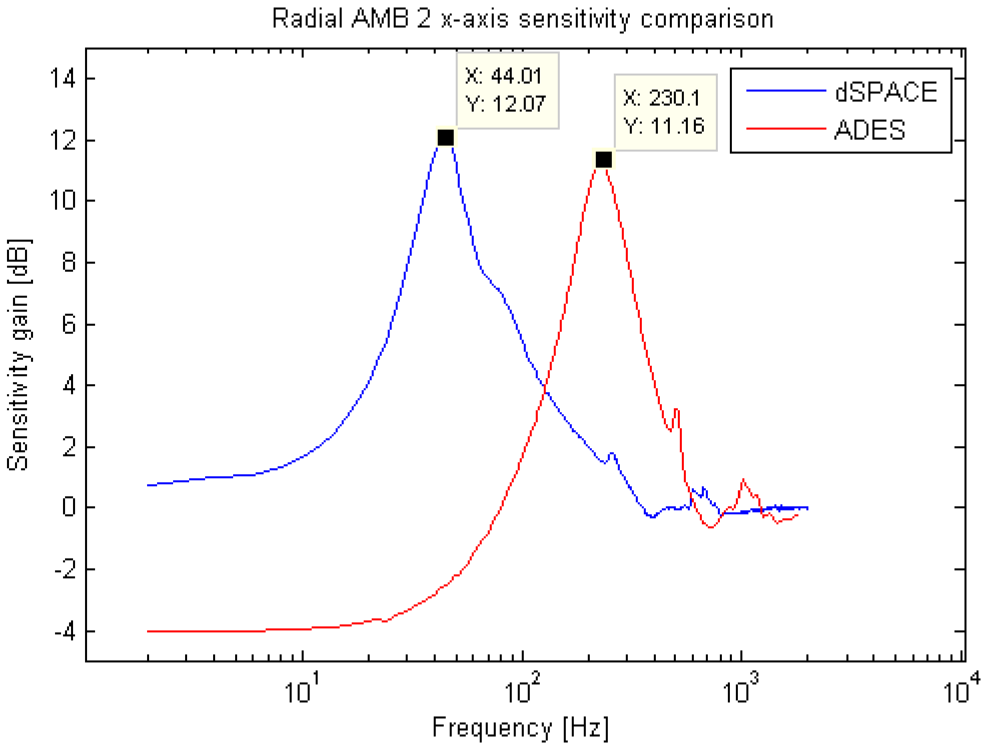


Figure 6-16: dSPACE® and ADES sensitivity response comparison

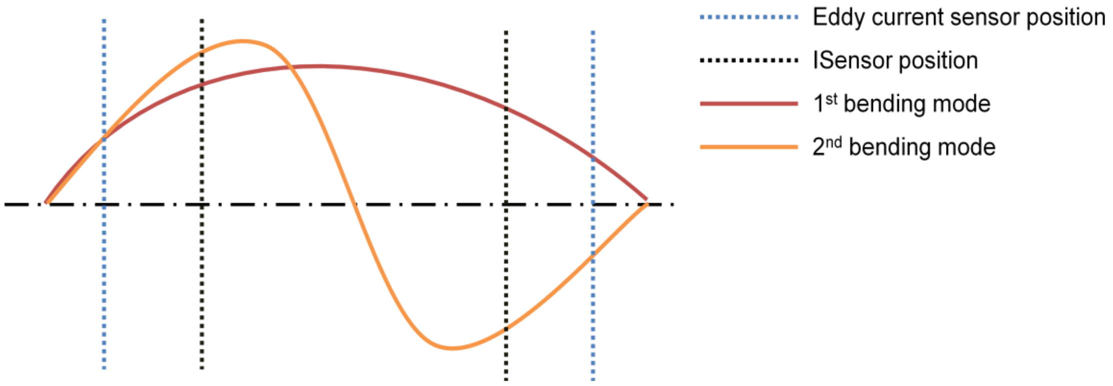


Figure 6-17: Bending modes and non-collocation of the sensors

Due to this effect, the sensitivity of the ADES will not be compared to the sensitivity of the dSPACE® system. From Figure 6-16 it can be seen that the profiles of the sensitivity plots between the dSPACE® and ADES correlate. This verifies the method used to analyse the sensitivity. The most prominent critical frequencies excited by the frequency sweep was determined and are listed in Table 6-11 [72]:

Table 6-11: Prominent critical frequencies of the ADES

Frequency [Hz]	Description
45	1st rigid body mode
60	2nd rigid body mode
220	Unknown[36]
400	Impeller critical frequency
780	1st bending mode
1500	2nd bending mode

6.6.1 Radial AMB 1

Figure 6-18 shows the sensitivity of radial AMB 1. As can be seen, the sensitivity gain for the x-axis is between 8 and 12 dB (11.03 dB to exact) but the sensitivity gain for the y-axis is 13.25 dB. The reason why the sensitivity for the x and y-axis differs is due to the rotor not being suspended in the middle of the iSensor[37].

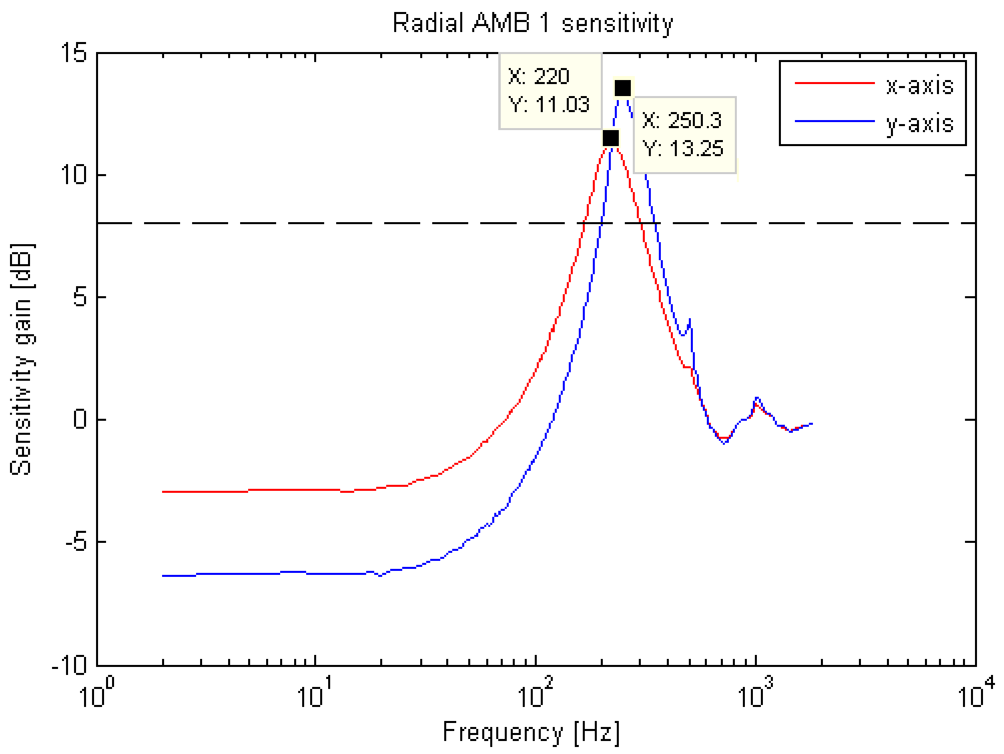


Figure 6-18: Radial AMB 1 sensitivity

[36] Further study is needed to determine the cause of these critical frequencies.
[37] Refer to section 6.5.1 for a complete discussion.

6.6.2 Radial AMB 2

The sensitivity of radial AMB 2 is shown in Figure 6-19. As can be seen, the sensitivity for both the x and y-axis correspond with each other. This implies that the rotor was suspended in the middle of the iSensor for this test. The sensitivity gain for the second radial AMB is also lower than 12 dB.

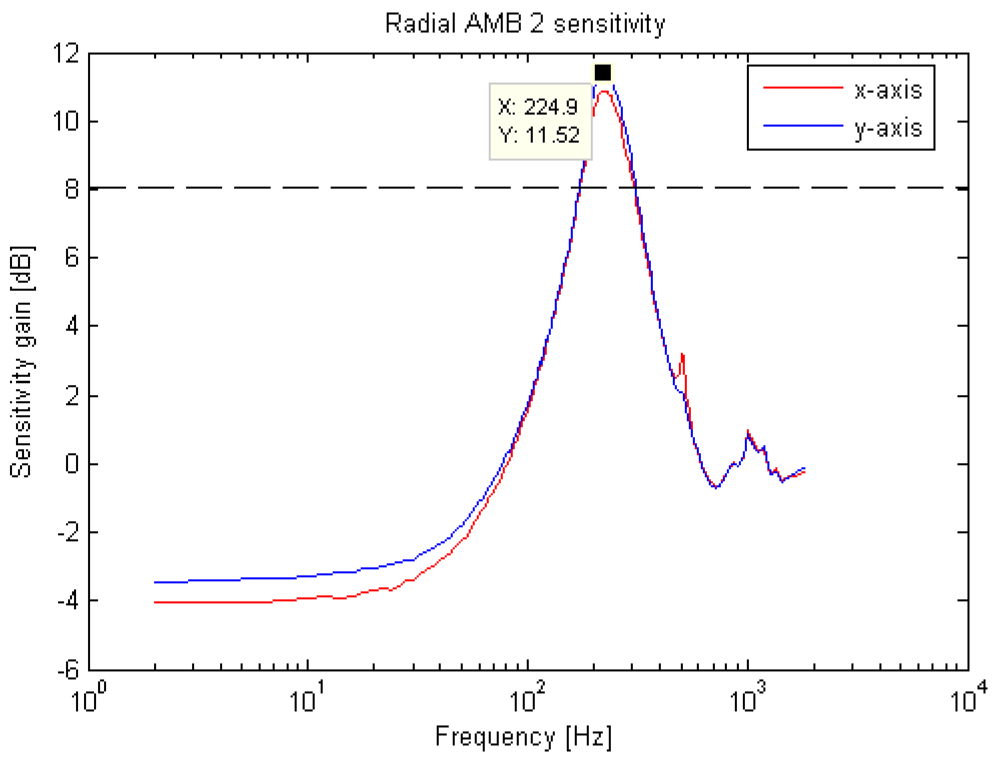


Figure 6-19: Radial AMB 2 sensitivity

6.6.3 System sensitivity

Form the sensitivity plots, it can be seen that the dominant frequency is the unknown 220 Hz. Further study is needed to determine the exact cause of this frequency. It can also be seen that the system is insensitive to the first two rigid body modes. The IIR band-stop filter filters the impeller critical frequency and the second bending mode frequency is filtered using the IIR notch filter. Frequencies higher that 2.5 kHz is filtered using the low-pass filter.

The ISO CD 14839-3 standard specifies that system sensitivity will be determined by the worst rating of any measured axis. From Figure 6-18 it can be determined that the system is a zone C system, since the sensitivity gain of the y-axis of radial AMB 1 is higher than 13 dB. This is the worst rating of any measured axis, and implies that the system is normally considered unsatisfactory for long-term continuous operation.

It was discussed in section 6.5.1, that the linear point of operation differs for the two systems. For these tests, the rotor was again suspended in the centre of the backup bearing modules. This centre is not necessarily the same as the centre of the iSensor. When performing the sensitivity analysis, the rotor is suspended closer to one bearing than the other, and a difference in sensitivity is measured.

By moving the linear point of operation for radial AMB 1 closer to the centre of the iSensor, can improve the sensitivity analysis. It can be seen that the sensitivity gain for radial AMB 1 y-axis is slightly higher than the 13 dB line. A zone B system is easily achievable. However, it was specified that the ADES should be a zone A system, since it is newly commissioned. To achieve such a drastic improvement, the PID parameters should be adjusted. This is discussed in the subsequent section.

6.6.4 Improving system sensitivity

The ADES is a newly commissioned system and the sensitivity gain of the system was specified to be lower than 8 dB, thus placing the ADES in zone A. From the analysis above, it was seen that the ADES is a zone C system. As mentioned, the sensitivity of the system can be improved by adjusting the PID parameters. For example, adjusting the PID parameters to those shown in Table 6-12, and measuring the sensitivity on radial AMB 2 using dSPACE® delivers the result shown in Figure 6-20. As can be seen, the sensitivity is below 8 dB. This implies a zone A system, which adheres to the system requirement specification. By increasing the differential gain and lowering the proportional gain, the damping of the system is increased and the sensitivity to the sinusoidal input is lowered due to the inertia of the rotor.

Table 6-12: PID parameters for zone A sensitivity

Sinusoidal amplitude	20 µm
PID parameters:	
K_P	17,000
K_I	5,000
K_D	50
Differential poles	
$\boldsymbol{\omega_{p1}}$	1 kHz
$\boldsymbol{\omega_{p2}}$	2 kHz
Position reference	0

When these PID parameters are used to evaluate the sensitivity by means of the ADES, the system destabilizes when K_D is increased above 38. This result indicates that the phase shift in the ADES system is larger than that of the dSPACE® system. Further work is needed to establish the source of this phenomenon.

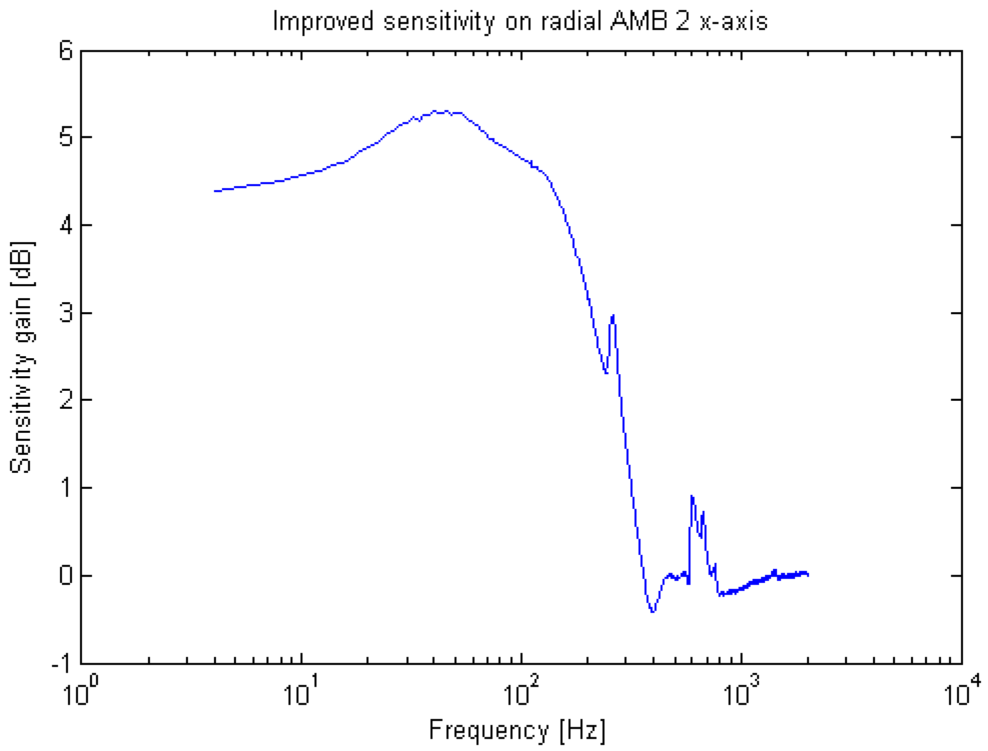


Figure 6-20: Improved sensitivity on AMB 2 x-axis

6.7 Conclusion

In section 3.1.2 it was stated that the selected controller should be capable of the following:

- Controlling the AMBs (F1.1)
  - Calculate control response (F1.1.2)
- Monitoring system (F1.3)
  - Run self-diagnostic test (F1.3.1)
  - Monitor conditions (F1.3.2)
  - Sensitivity analysis (F1.3.3)
- Communicate (F1.4)
  - Transmit (F1.4.1)
  - Receive (F1.4.2)

In this chapter, it was seen that the controller performance is sufficient for implementing the control of the AMBs (F1.1). The performance discussed in section 6.2 is based on the performance of the PowerPC and it is possible to move some features to the FPGA to leave more room on the PowerPC for additional floating-point calculations. Take the motor control for example. It was estimated that 47 µs is needed to perform motor control using 3-phase pulse width modulation. However, not enough room is left within one 20 kHz cycle to perform motor control on the PowerPC. The solution is to implement the motor control on the FPGA, or to move some of the fixed-point calculations to the FPGA to make enough room on the PowerPC for motor control.

Even though the self-diagnostic test (F1.3.1) and condition monitoring (F1.3.2) was not included in this iteration of the project, the remaining clock cycles are sufficient for implementing a self-diagnostic test and/or a simple form of condition monitoring. If a more complex form of condition monitoring, such as neural network based or statistical condition monitoring is required, the same recommendation as above is made; move some of the features off the PowerPC and onto the FPGA.

It was also seen that a real-time sensitivity analysis (F1.3.3) on the PowerPC is not possible. If a real-time sensitivity analysis is required, it is suggested that it should rather be implemented on the FPGA. Determining the stability and sensitivity of the system was done by adding the required signal, either step or sinusoidal, to the reference position of the system. The response of the system was logged and extracted non-real time using the SBC. From section 6.5 it was determined that the response of the system was stable despite the step added to the reference. It was also seen that the noise on the ADES is significantly less than on the dSPACE® system. From the sensitivity analysis, it was determined that the ADES falls within zone B and is thus considered acceptable for unrestricted long-term operation, even though it does not adhere to the specifications given in the system requirements. Even though it is possible to improve the sensitivity response of the system by increasing the differential gain, it is not possible with the ADES control.

Another requirement was that the selected controller's calculation of the control response should be capable of MIMO control (F1.3.3), even though it was not implemented on the first iteration of the project. It was seen in section 6.4 that, although difficult, MIMO control will be possible on the selected controller.

The last function the controller should be able to perform is communicate (F1.4). Even though the design and implementation of the communication does not fall within the scope of this dissertation, the selected controller should be capable of receiving (F1.4.2) and transmitting (F1.4.2) data between components. The communication was implemented on the FGPA, since it is possible to communicate the data concurrently to the required components. The selected PMC module is equipped with both front and real I/O interfaces, which can be used to interface with external components. The front I/O module is the AXM-D03 I/O module. This module has 16 digital and 22 differential channels, which are directly accessible from the FPGA. Data is sent to and from the sensor and power amplifiers via this I/O module. The rear I/O has low-voltage differential signalling lines, which is not used at this stage.

CHAPTER 7

Conclusions and recommendations

This chapter concludes the dissertation by discussing the controller selection, design and implementation process. Recommendations are then given to assist in similar projects. Lastly, the future work is discussed and a closure statement given.

7.1 Conclusions

The aim of this project as a whole was to develop an industrial system capable of controlling AMBs in high-speed applications, such as a helium blower system. The main controller, as part of the ADES, should be capable of performing the control of the AMBs, monitoring the system and communicating values through the system.

7.1.1 Controller selection process

Selecting a suitable controller was based on performing trade-off studies on the system and main controller architectures. The system and main controller architectures were evaluated simultaneously using decision matrices. The decision matrices evaluate the architectures on robustness, efficiency, cost, risk, flexibility and expandability.

From section 2.1.6 it was seen that the final controller selection must pass four critical tests [49]:

1. Is it available in a suitable implementation?
2. Is it capable of sufficient performance?
3. Is it supported by a suitable operating system?
4. Is it supported by appropriate and adequate tools?

The selected controller will now be discussed using these guidelines;

Is it available in a suitable implementation?

Two implementations were considered during the selection process of the main controller unit for the ADES; either an in-house developed controller, or a commercial off-the-shelf product. Since the ADES should adhere to industrial specifications, the latter implementation was selected. This will ensure an industrial product, while minimizing the risks involving development and manufacturing of the controller board.

The off-the-shelf solution selected is based on an industrial, more robust version of PCI called compact PCI or cPCI. Various different configurations of the cPCI architecture were considered. From the system architectural comparison, it was evident that the best architecture for the system is system architecture 1.2. Two suitable options for this architecture were considered, shown in Figure 3-23 and Figure 3-24. The first represented main controller concept architecture 4, shown in Figure 3-10, and the other represented main controller concept 6, shown in Figure 3-12.

The selected configuration consists of an SBC which slots into a cPCI backplane. The SBC has two PMC sites; one sites the Virtex®-5 PMC module and the other sites the Profibus card, which will be used to interface the system to a PLC. The SBC is used for scheduling the non-real time tasks, data logging and as a user interface. The Virtex®-5 PMC module is used for scheduling the real-time tasks such as controlling the AMBs.

The selected architecture is industrial, robust, powerful and expandable. It is thus suitable for controlling AMBs in an industrial environment.

Is it capable of sufficient performance?

The selected controller should be powerful enough to control AMBs, monitoring the system and to communicate the requested values to the required components. Monitoring the system includes performing a self-diagnostic test, monitoring conditions and performing a sensitivity analysis. It was also specified that the selected controller be capable of MIMO control, for future iterations of this project. The control cycle for all these requirements are specified at 20 kHz.

The number of clock cycles it takes the PowerPC to perform the implemented functions was measured and discussed in section 5.6.5. It was seen that it takes 4,880 clock cycles for the PowerPC to read and scale the position values, perform PID control for five axes and to log the data. This figure excludes condition monitoring, performing a self-diagnostic test and performing the sensitivity analysis. Luckily, not all these functions need to be included in the remaining 1,375 clock cycles. Performing the self-diagnostic test is not required to be a real-time function. It should be performed during standby, where it is not critical if the process takes longer than the specified 20 kHz control cycle. The remaining clock cycles are insufficient for implementing complex condition monitoring and/or real-time sensitivity analysis on the PowerPC. To implement these functions, some of the features have to be moved from the PowerPC and implemented on the FPGA.

As discussed in section 6.4, the selected architecture is also powerful enough to perform MIMO control, if the FPGA is used to alleviate the computation-intensive functions from the PowerPC.

Is it supported by a suitable operating system?

One of the advantages of the selected architecture is that it is very flexible, even in terms of operating systems. Not only is it possible to install a real-time OS, such as QNX, on the SBC, it is also possible to use the embedded PowerPC without an OS, or to install a real-time OS. This enables various combinations of this architecture. Where the SBC is currently used for scheduling the non-real time tasks, with a real-time OS it will be possible to incorporate real and non-real time scheduling on the SBC. Installing a real-time OS on the embedded PowerPC adds additional drivers and features which could improve or enhance the embedded system.

Is it supported by appropriate and adequate tools?

The problem with most tools supporting embedded devices, such as FPGAs and DSPs, are that the companies manufacturing these tools primarily focus on hardware and not software. Software is not their primary trademark. The tools are therefore prone to bugs. Even though third party tools may be available for most of the embedded devices, these tools do not necessarily include the drivers, IP cores and support the manufacturing company offers. Due to these added benefits of using the manufacturer’s software, it was decided that it would be used. However, some of the issues experienced with the software include:

- Calculating the PID parameters, which include numerous floating-point divisions, was impossible using Xilinx SDK 10.1. The PowerPC crashes during a floating-point division. This issue was resolved by upgrading to SDK 11.1.

- The SDK used for developing software for the PowerPC can only debug the code by simulating the code on hardware. What this entails is loading the developed software onto the PowerPC using the EDK-board via the JTAG connector. The debug window is then opened from where the code can be debugged. A “debug on software”-mode is also available, but this mode has so little functionality, that it was impossible to debug the code. The problem with debugging the code on the hardware is two-fold. The first is that the EDK-board is necessary for debugging. Since the EDK slots into the same slot as the AXM module used for external communication from the PMC module, external communication is not possible when the EDK is used. The other problem is that if code is already running on the embedded PowerPC, debugging additional code is complicated since this code should be loaded onto the PowerPC to be debugged. Occasionally inconsistent data is read during this debug method.

These are just some of the issues experienced and as can be seen, the biggest issue experienced was with debugging with the SDK. Despite the issues regarding tools, most of the tests were passed and the conclusion can be made that the selected controller is sufficient for the application.

7.1.2 Controller design

The hardware of the system was designed using the software control flow given in Figure 4-1. Twenty-four DPR instantiations were needed to facilitate this design. The DPRs were instantiated in VHDL and they were chosen large enough to contain nine 32-bit values. The data bus width were chosen as 32-bits to simplify integration with the PowerPC memory controller, which is also 32-bit. This, however, complicated the integration with the communication controller, which communicates 16-bit data packages. This meant concatenating the 16-bit data packages with zeros until it is 32-bits long. The problem with this approach is that the most significant bit (MSB) in the VHDL bit-vectors can be declared to be on the left or right side of the bit-vector. The convention used throughout the VHDL code design was to define the MSB as the leftmost bit in the vector; however, the PowerPC memory controller does not follow this convention. By the time this was realized, most of the components in VHDL were already written. The 16-bit data packages should thus not only be converted to 32-bit values, the order and position of the bits should be correct, or else the wrong data is read. The same applies to a write. Since the rest of the system uses only 16-bits, the written data should be positioned in the right half of the data word, or else the wrong data is truncated.

After designing the DPRs and memory interfaces, the rest of the required components can be designed. The first was the embedded state machine. This state machine represents the states the embedded system can reside in. The states contained in the embedded state machine represent the states specified in the system requirements specification. A state transition occurs when the GUI issues a state-change interrupt.

Six possible interrupts were identified and even though the final implementation of the system does not make use of all the identified interrupts, the interface exists and the additional interrupts can be easily utilized. The interrupts are connected to an external bus. This bus is connected to the input port of a general-purpose I/O controller, which is set-up to trigger an interrupt should a rising edge occur on the input port. The triggered interrupt is sent to an interrupt controller, which handles the external interrupts.

7.2 Recommendations

This section discusses the unresolved issues and makes some general recommendations regarding certain aspects.

7.2.1 Controller implementation

As discussed in the previous section, the hardware of the system was designed using the software control flow given in Figure 4-1. As can be seen from the figure, six distinct, yet identical, paths exist; one for each degree of freedom. Identical components also exist in each path. Each path contains (besides the PID controller and PCI memory space):

- 4 DPRs
- A filter controller
- A filter entity containing 2 or 3 filters

A UART controller and packet analyser also exist for each power amplifier and two exist for receiving the sensor data. This brings the total number of components to:

- 24 DPRs
- 6 Filter controllers
- 6 Filter entities
- 7 UART controllers

Instantiating these components individually results in bulky code and a more elegant method exists. Each type of component can be instantiated using a "*for/generate*" VHDL statement. The "*for/generate*"-statement is a concurrent statement which allows for a piece of code to be repeated a number of times, while creating numerous instances with the same assignments [36]. A "*for/generate*"-statement will exist for each of the components mentioned above. Take the DPRs for example. A "*for/generate*"-loop from 0 to 23 will be used to instantiate 24 DPRs. Each individual DPR will then be identified using the index of the loop.

It is also highly recommended to compile a document describing the interfaces to and from each VHDL entity. This document should describe the position of the MSB for each port, the number of bits for each port as well as the connectivity for each port. This document will ensure that the correct number representation is used by each port and will clarify the naming convention, especially when a "*for/generate*"-statement is used.

7.2.2 Debugging and simulation

As discussed in section 7.1.1 during the test to verify whether adequate tools support the selected controller, debugging is a problem when using the SDK. The EDK module played an essential part in debugging the PowerPC code, since no software-debugging mode is available and the only way to debug the PowerPC code is by loading the code onto the PowerPC using the EDK module.

The difficulties regarding the setup of a PowerPC simulation were discussed in section 5.1. A simulation encapsulating both FPGA and PowerPC would accelerate the debugging process tremendously, since it would be possible to simulate the entire system, without the need for the SDK module. Even ChipScope™, the on-chip logic and bus analyser developed by XIlinx®, requires the use of a JTAG connector, and since the only available JTAG is on the EDK, it cannot be used.

Even though emulating the PowerPC code works well to a certain extent, the difficulties in using the EDK-module is time consuming. A complete system simulation is not impossible to set up. The only drawback is that a lot of time is needed to thoroughly analyse the stimulants needed in the simulation and to include the PowerPC. A more effective debugging method will be timesaving.

7.3 Future work

As with all projects, there is room for improving the ADES system. Some features could not be included due to time limitations. This section is dedicated to discussing the future work needed to be done.

7.3.1 Sensitivity analysis inconsistencies

The ADES was specified to be a zone A system in terms of sensitivity. However, the sensitivity analysis indicated that the ADES is a zone C system, which can be improved to a zone B system by adjusting the linear point of operation. The cause of the higher sensitivity gain, can be attributed to the PID control which is incapable of delivering high enough damping. The possible case was given in section 6.6.4 as a phase shift in the ADES system which is larger than that of the dSPACE® system. Further study is needed to establish the source of this phenomenon. A further study is also required to determine the origin of the 220 Hz critical frequency.

7.3.2 Self-diagnostic test

A self-diagnostic test should be performed before entering the *"Standby"* mode. The basic idea is to verify that each component is operational before granting the user control over the system. The general idea behind this test is for the sensor to generate a dummy value. This value is then propagated through the system until the PID controller is reached. The PID control then generates a reference value according to the value received. This reference value is then sent to the power amplifiers, where the control loop is closed. Since all the parameters are known during the self-diagnostic test, the current values sent back from the power amplifiers can be evaluated. If the anticipated value is received, it can be concluded that all the components in the system is functional. This test is concluded without physically suspending the rotor.

7.3.3 Condition monitoring

One of the requirements of the ADES was that it should be capable of performing condition monitoring. Condition monitoring is used to monitor certain conditions while the system is operational. From these conditions, certain characteristics and possible failures can be determined. Various types of condition monitoring exist, from statistical to "intelligent" methods using neural networks and fuzzy logic.

Even though condition monitoring is an excellent feature to include in the ADES, if done properly, the scope for condition monitoring can become quite large. It was thus left out of scope for this iteration of the ADES, but it could be implemented at a later stage.

The number of clock cycles remaining on the PowerPC is not enough to implement condition monitoring within the 20 kHz cycle. It should thus be implemented on the FPGA, which significantly complicates its implementation. sufficient

7.3.4 Real-time sensitivity analysis

Two states were included in the state machine to include a real-time sensitivity analysis at standby and at nominal speed. However, analysing the frequency spectrum of the result before obtaining the sensitivity, takes a significant number of clock cycles. It is thus not possible to implement the sensitivity analysis on the PowerPC in the remaining clock cycles.

Two solutions to this problem exist. The first solution is fundamentally the same implementation as done in section 6.6. The reference and error values are logged using the SBC. The sensitivity is then determined using a program such as MATLAB®.

The second solution is to use the FPGA to analyse the frequency data. The PowerPC can still be used to log the reference and error values, but the frequency spectrum is determined using the FPGA. A 64k point FFT is available when using Xilinx®'s ExtremeDSP™ slices [39], which will be sufficient for determining the frequency spectrum. The result from the error and position FFT should still be divided, whereafter the log is determined and the result multiplied by 20. This can either be done using the PowerPC (if enough clock cycles are available) or on the FPGA. Further study is needed to determine the exact implementation.

7.3.5 Emergency stop procedure

Two states were included in the embedded state machine to provide for two emergency stop procedures. The first is the emergency stop procedure for a critical failure and the second is the emergency stop procedure for a non-critical stop. Currently, the only features included in these two states are a transition to "*Standby*" for a critical stop, and a transition to "*AMBs_Operational*" for a non-critical stop. From a mechanical point of view, further study is needed to determine the correct course of action in a critical or non-critical situation. The determined course of action should then be implemented in the appropriate state.

7.3.6 Motor control

The proposed system was shown in Figure 1-4. As shown in the figure, the ADES system should also be capable of controlling the motor. However, this specification was changed for the first iteration of the ADES. A Siemens® PLC and motor drive is used to control the motor. The next iteration of the ADES system will use the Profibus PMC card to interface with the PLC. From there, the whole system, including the motor control will be done via the PLC. This card must be configured, implemented and connected to the PLC. A project is also underway to develop a Profibus IP core, which will be implemented on the Virtex®-5 PMC module. This IP core will replace the Profibus PMC module.

7.3.7 Remote-access port

The Ethernet/USB port featuring in the main controller architectural discussion in section 3.5.1, will be used as a remote-access port in future iterations of the ADES project and will enable remote control, updates and maintenance of the ADES system. The Ethernet port was selected above the USB port, due to the network features required. The Ethernet port is located on the SBC. An application needs to be developed to interface the PMC module with the Ethernet port.

7.3.8 MIMO control algorithm

MIMO control for AMBs is a new field of study within the McTronX research group. The reason why it was specified that the selected controller should be capable of MIMO control, is to enable future research in MIMO control on the ADES.

7.4 Closure

The purpose of this project was to develop an AMB and drive electronic system (ADES) capable of performing in high-speed industrial applications such as a helium blower. The focus of this dissertation was on the specification, selection and implementation of a controller suitable for the ADES. The selected controller should be capable of controlling the AMBs, monitoring conditions, performing self-diagnostic tests, performing real-time sensitivity analysis and communicating with the external components such as the sensor unit and power amplifiers. Evaluation of the selected controller indicated that the controller is capable of monitoring positions, controlling the AMBs, logging data and communicating; all within 50 µs. It was also determined that the selected controlled adhered to the specifications given, with the exception of the sensitivity analysis. Despite the higher sensitivity, the control was stable enough to suspend the rotor at its designed rating of 19,000 r/min, but due to the sensitivity rating, prolonged operation at this speed is not recommended. It can thus be seen that the FPGA and embedded PowerPC is a versatile, compact and powerful architecture suitable for controlling AMBs in an industrial environment.

Bibliography

[1] R.F Tinder, *Engineering digital design*, 2nd ed.: Academic press.

[2] A. Dhir, *The Digital Consumer Technology Handbook: A Comprehensive Guide to Devices, Standards, Future Directions, and Programmable Logic Solutions*.

[3] R.M Yeomans, "Advanced Gas-Cooled Reactors (AGR)," in *International Atomic Energy Agency*, Austria, 1980, pp. 17-38.

[4] The virtual nuclear tourist. (2009, April) Gas Cooled and Advanced Gas Cooled Reactors. [Online]. http://www.nucleartourist.com/type/gcr.htm

[5] G.P. Greyvenstein and P.G. Rousseau, "Design and successful testing of a physical model of the Pebble Bed Modular Reactor," *International journal of nuclear power*, p. p2.

[6] Universal Instruments, "Generation IV Nuclear Reactors," Australian Uranium Association, UIC briefing paper #77 February 2008.

[7] ESKOM. Pebble Bed Reactor Technology. [Online]. http://www.escom.co.za/nuclear_energy/pebble_bed/pebble_bed.html

[8] R.H Bishop, *Mechatronics: An Introduction*.: CRC Press, 2006.

[9] G. Schweitzer, H. Bleuler, and A. Traxler, *Active magnetic bearings: Basics, Properties and Application of Active Magnetic Bearings*, Authors reprint ed. Zürich, 2003.

[10] M. Aenis, E. Knopf, and R. Nordmann, "Active magnetic bearings for the identification and fault diagnosis in turbomachinery," *Mechatronics* , vol. 12, p. 1011–1021, 2002.

[11] J. Boehm, R. Gerber, and N.R.C Kiley, "Sensors for magnetic Bearings," *IEEE Transactions on magnetics*, vol. 29, pp. 2962-2964, November 1993, No. 6.

[12] H Bleuler, C. Gähler, and R. Hertzog, "Application of digital Signal Processors for industrial magnetic bearings," *IEEE transactions on control system technology*, vol. 2, no. 4, pp. 280-289, 1994.

[13] A. V. Deshmukh, *Microcontrollers: Theory and Applications*.: McGraw-Hill, 2005.

[14] Keep talking. Keep Talking Glossary. [Online]. www.keep-talking.net/glossary.htm

[15] United Electronic Industries. Embedded Controller (definition and description). [Online]. http://www.ueidaq.com/media/static/other/so/embedded-controller.pdf

[16] Intel®. Intel microprocessor hall of fame. [Online]. http://www.intel.com/museum/online/hist_micro/hof/index.htm

[17] Intel®. Intel® microprocessor quick reference guide. [Online]. http://www.intel.com/pressroom/kits/quickrefyr.htm

[18] J. L. Antonakos, *The Intel® Microprocessor family: Hardware and software principles and applications*.: Thompson, 2007.

[19] B. B. Brey, *The Intel® Microprocessors: Architecture, programming and interfacing*, 7th ed.: Pearson.

[20] BDTI. BDTI's DSP Dictionary. [Online]. http://www.bdti.com/articles/dspdictionary.html

[21] IBM®. IBM® WebSphere Voice Server Information Centre: Glossary. [Online]. http://publib.boulder.ibm.com/infocenter/pvcvoice/51x/index.jsp?topic=/com.ibm.websphere.wvs.doc/wvs/glossary.html

[22] E.C. Ifeachor and B.W. Jervis, *Digital Signal Processing: A practical approach*, 2nd ed.: Prentice Hall, 2002.

[23] S.W Smith, *Digital Signal Processing: A practical guide for scientists and engineers*, Imported edition ed.: Newnes, 2003.

[24] S.A. Ward and Jr. R.H. Halstead, *Computation structures*, 5th ed.: The MIT press, 1999.

[25] A.R. Hambley, *Electrical engineering: principles and applications*, 3rd ed.: Prentice Hall, 2004.

[26] Analog Devices Inc., Embedded processors and DSP selection guide, 2008, [Available on the data disc].

[27] BDTI, Speed per Milliwatt Ratios for Floating-Point Packaged Processors (Berkeley design Technoloy, Inc.), 2007, [Available on the data disc].

[28] BDTI, Speed Scores for Floating-Point Packaged Processors (Berkeley design

Technoloy, Inc.), 2007, [Available on the data disc].

[29] J. Milrod, "Continuous real-time signal processing: Comparing the 300 MHz TigerSharC to the 500 MHz MPC7410 PowerPC and the 1 GHz MPC7455 PowerPC," Bittware, Inc., 2002.

[30] RadioLocman. (2002, December) Analog Devices TigerSHARC Processor Delivers Industry's Highest DSP Performance. [Online]. http://www.radiolocman.com/news/new.html?di=24

[31] The Free library. (2004, March) Analog Devices' 600MHz TigerSHARC Earns Top Floating-Point Benchmark Score. [Online]. http://www.thefreelibrary.com/Analog+Devices'+600MHz+TigerSHARC+Earns+Top+Floating-Point+Benchmark.-a0114746520

[32] BNET. (2003, March) Analog Devices' TigerSHARC Processor and iMEMS Gyroscope Named 2002 Innovation of the Year Awards Finalists. [Online]. http://findarticles.com/p/articles/mi_m0EIN/is_2003_March_6/ai_98418802

[33] The Free library. (2004, February) Analog Devices' TigerSHARC Processor Named 2003 Innovation of the Year Award Finalist by EDN Magazine. [Online]. http://www.thefreelibrary.com/Analog+Devices'+TigerSHARC+Processor+Named+2003+Innovation+of+the.-a0112980250

[34] Texas Instruments®. TMS320C67x floating-point DSPs. [Online]. http://focus.ti.com/paramsearch/docs/parametricsearch.tsp?family=dsp&sectionId=2&tabId=135&familyId=327&paramCriteria=no

[35] B. Mealy, The low-carb VHDL tutorial, 2004.

[36] V.A Pedroni, *Circuit design with VHDL*. London, Massachusetts, England: MIT Press, 2004.

[37] I. Kuon, R. Tessier, and J. Rose, "FPGA architecture: Survey and challenges," *Foundations and Trends in Electronic Design Automation*, vol. 2, 2007.

[38] Xilinx®. Getting started with FPGAs. [Online]. http://www.xilinx.com/company/gettingstarted/index.htm

[39] Analog Devices Inc.®, "XtremeDSP solutions selection guide," Selection guide 2008.

[40] V. Pietikäinen, ARM architecture: ARM7, ARM9, TDMI., November 19, 2002, [Available on the data disc].

[41] ARM. ARM architecture for the digital world. [Online]. http://ir.arm.com/phoenix.zhtml?c=197211&p=irol-homeprofile

[42] C.R Moore and R.C Stanphill, "The PowerPC alliance," *Communicaitons of the ACM*, vol. 37, pp. 25-27, 1994.

[43] Freescale Semiconductor®, "Programming Environments Manual for 32-Bit Implementations of the PowerPC™ Architecture," Freescale Semiconductor®, 2005.

[44] Xilinx®, "Embedded Processor Block in Virtex-5 FPGAs," Xilinx®, Reference guide UG200 (v1.6), 2009.

[45] PC/104 embedded consortium. Why PC/104? The Need for an Embedded-PC Standard. [Online]. http://www.pc104.org/specifications.php

[46] PC/104 embedded consortium. PC/104 Specifications. [Online]. http://www.pc104.org/pc104_specs.php

[47] The Free Dictionary. PC/104. [Online]. http://encyclopedia2.thefreedictionary.com/PC/104

[48] IEEE Standards Department, Draft Standard Physical and Environmental layers for PCI Mezzanine cards: PMC, April 4, 1995.

[49] A.S. Berger, *Embedded Systems Design: An introduction to Processes, Tools, and Techiniques*.: CMP Books, 2008.

[50] B. O'Rourke and R. Taylor, "Turbocharging your CPU with an FPGA-programmable co-processor: Co-processor synthesis adds the optimal programmable resources," Xilinx, 2006.

[51] A.R. Weiss, "Dhrystone Benchmark: History, Analysis, "Scores" and Recommendations," ECL, LLC, 2002.

[52] BDTI. BDTI company overview. [Online]. http://www.bdti.com/bdti_overview.html

[53] BDTI, "The BDTImark2000™: A summary measure of signal processing speed," Berkeley design technology, Inc., Benchmarks 2004.

[54] BDTI, Speed per Dollar Ratios for Floating-Point Packaged Processors, 2007, [Available on the data disc].

[55] E.O. Ranft, "The development of a flexible rotor active magnetic bearing system," North-West University, Potchefstroom, Master's dissertation 2005.

[56] C.L Phillips and H.T Nagle, *Digital control system analysis and design*, 3rd ed.: Prentice Hall, 1995.

[57] M. Neser, "Object-Oriented Embedded C," North-West University, Potchefstroom, Technical report.

[58] S.R. Schach, *Object-Orientated and Classical Software Engineering*, 7th ed.: Mcgraw-Hill, 2007.

[59] ANON, TMS320F/C24x DSP Controllers Reference Guide: CPU and Instruction Set, 1999, Doc nr. : SPRU160C.

[60] B. Mackin and N. Woods, "FPGA Acceleration in HPC: A Case Study in Financial Analytics," XtremeData Inc., 2006.

[61] S.Y. Lim and A. Crosland, "Implementing FFT in an FPGA Co-Processor," Altera Corporation, 2005.

[62] G. Daryanani, *Principles of active network synthesis and design*.: Wiley, 1976.

[63] Xilinx®, Virtex-5 FPGA Embedded Processor Block with PowerPC 440 Processor, January 20, 2009.

[64] IBM®. Power Architecture Licensing. [Online]. http://www-03.ibm.com/technology/power/licensing/cores/ppc440.html

[65] IBM®, "PPC440x4 CPU Core User's Manual," IBM®, User manual 2003.

[66] Xilinx®, Standalone Board Support Package, January 8, 2007, [Available on the data disc].

[67] ISO, Mechanical vibration — Vibration of rotating machinery equipped with active magnetic bearings, July 10, 2002, BS ISO 14839.

[68] D. Herbst, "Single board computer based control of an active magnetic bearing," North-

West University, Potchefstroom, Master's dissertation 2008.

[69] G. Steiner, B. Jones, and P. Alfke, "Floating-Point: Have it your way with FPGA embedded processors," *XCell journal: Solutions for a programmable world*, vol. Issue 67, pp. 32-35, 2009.

[70] IBM®, "The PowerPC440 Core: A high-performance, superscalar processor core for embedded applications," IBM® Microelectronics Division, 1999.

[71] M.D. Noh, "Self-sensing magnetic bearings driven by a switching power amplifier," University of Virginia, PhD thesis 1996.

[72] N. Bessinger, "The adaptation of a rotor for active magnetic bearing levitation and the corresponding auxiliary desing," North-west University, Potchefstroom, Master's dissertation Current.

APPENDIX

The additional information used within this dissertation is given here. Appendix A starts off by giving supplementary information regarding the BDTI benchmark suite. BDTI benchmarking examples are also given. Appendix B gives a detailed cost comparison between the two system architectures considered in section 3.6; the T2-6U-cPCI and the PP41x/03x. Appendix C shows a computer-aided design (CAD) generated section of the rotor de-levitation system. Appendix D and E shows photos of the system and graphical user interface (GUI) respectively. Lastly, Appendix F lists the data on the accompanying disc.

Appendix A: BDTI benchmark suite

Table A-1 shows the list of functions included by the BDTImark2000™. Figure A-1 to A-3 shows examples of different BDTImark2000 benchmarks.

Table A-1: BDTI benchmark™ suite

Function	Description	Example application
Real block finite impulse response (FIR)	FIR filter that operates on real (not complex) data	Speech processing or compression
Single-sample FIR	FIR filter that operates on a single sample data	Speech processing or general filtering
Complex block FIR	FIR filter that operates on a block of complex data	Modern channel equalization
Least-mean square (LMS) adaptive filter	LMS adaptive filter that operates on a single sample of real data	Channel equalization, servo control and linear predictive coding
Two-biquad infinite impulse response (IIR)	IIR filter that operates on a single sample data	Audio processing and general filtering
Vector dot product	Sum of the point-wise multiplication of two vectors	Convolution, correlation, matrix multiplication, multi-dimensional signal processing
Vector add	Point-wise addition of two vectors to give a third vector	Graphics, combining audio signals or images or vector search
Vector maximum	Find the location of the maximum value in a vector	Error control coding and algorithms using floating-point
Viterbi decoder	Decodes a convolutional encoded bit stream	Wired and wireless communication
Control	A contrived series of control and bit manipulation instructions	Virtually all signal processing applications
256-point fast Fourier transform (FFT)	An FFT converts an time domain signal into the frequency domain	Radar, sonar, MPEG audio compression or spectral analysis
Bit unpack	Unpacks words or varying length from a continuous bit stream	Audio and speech decompression

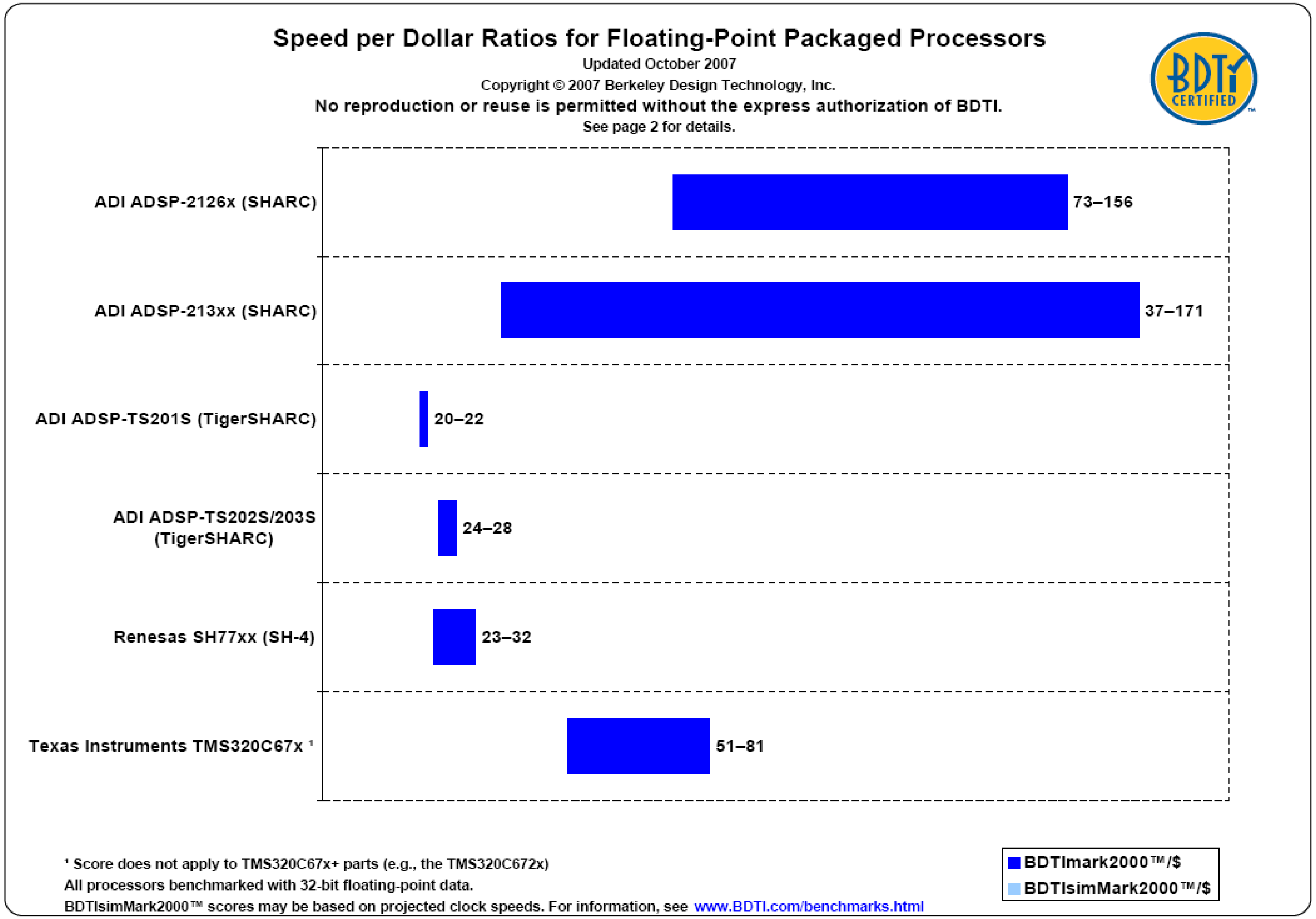


Figure A-1: BDTI speed per dollar ratios for floating-point processors [54]

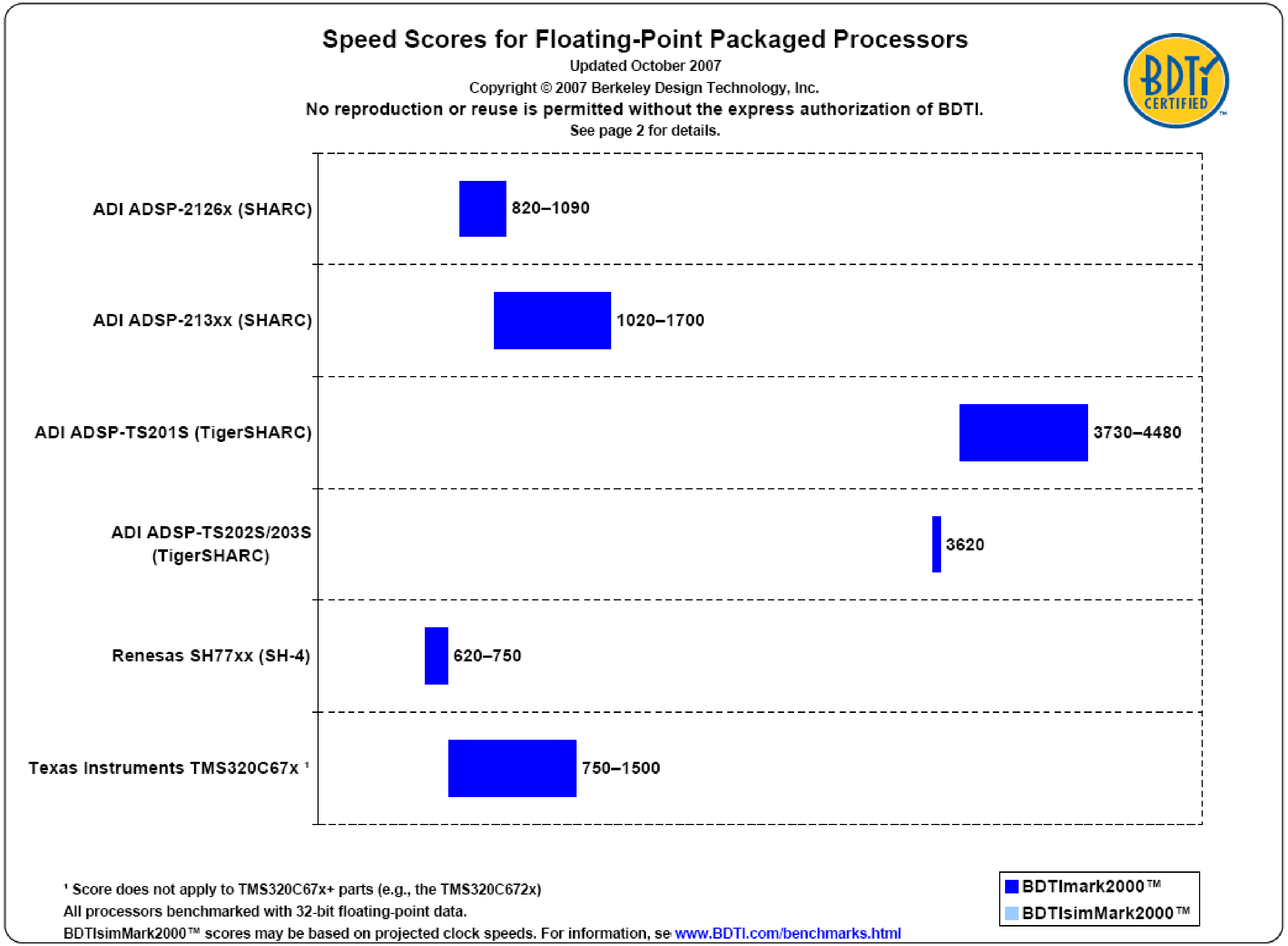


Figure A-2: BDTI speed scores for floating-point packaged processors [28]

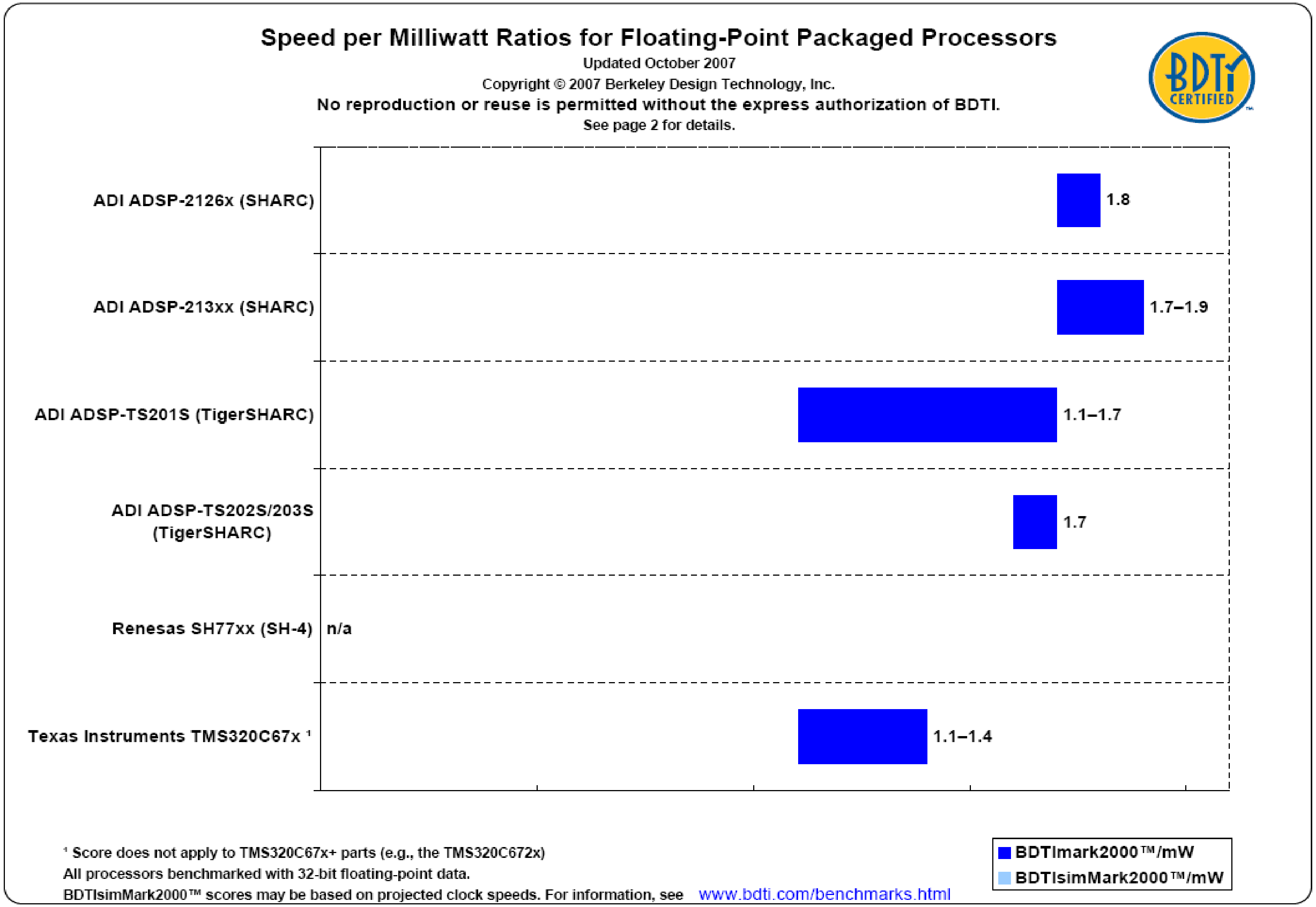


Figure A-3: BDTI speed per millwatt ratios for floating-point packaged processors [27]

Appendix B: Exposition of architecture costs

The two architectures considered in section 3.6 were:

- The T2-6U-cPCI TigerSHARC™ board
- The PP41x/03x Intel® board with Virtex®-5 FPGA PMC

A basic price-comparison was given in Table 3-25 between these two options. The table below give a comprehensive exposition of the architecture costs.

Table D-1: Architectural cost exposition

SBC and assembly components		
SBC	PP412/032-x2	R 31,221.43
Transition module		R 5,000.00
Break out cable		R 500.00
HDD		R 3,000.00
Assembly kit		R 820.00
	Total (excl VAT)	R 40,541.43
Case		
2U case		R 24,000.00
	Total (excl VAT)	R 24,000.00
Profibus module		
Profibus PMC		R 10,800.00
Software		R 7,220.00
	Total (excl VAT)	R 18,020.00
Acromag® FPGA		
Virtex®- 5 PMC	PMC-VFX70	R 55,000.00
Engineering design kit		R 10,000.00
Windows drivers		R 3,000.00
Virtex®- 5 development board		R 25,000.00
	Total (excl VAT)	R 93,000.00
TigerSHARC option		
TigerSHARC™ SBC	T2-6U-cPCI	R229,700.00
Visual DSP	VDSP-TS-PC-FULL	R 39,700.00
Emulator	ADZS-HPUSB-ICE	R 46,200.00
RTM module		R 10,000.00
Carrier card		R 10,000.00
	Total (excl VAT)	R 335,600.00

Appendix C: Rotor de-levitation sectional view

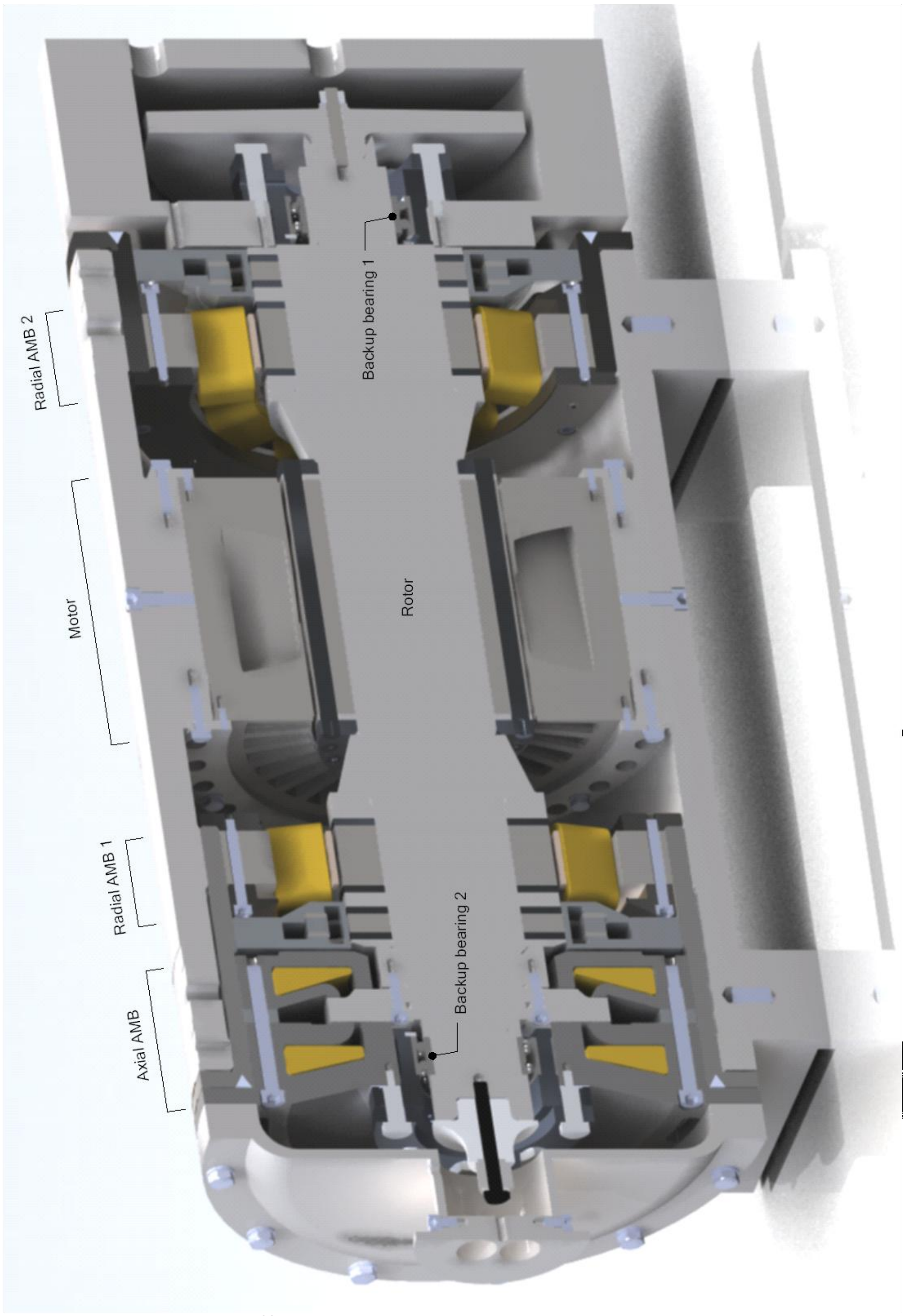


Figure B- 1: CAD[38]-rendered sectional view of the rotor de-levitation system

[38] Computer-aided design

Appendix D: Photos of completed system

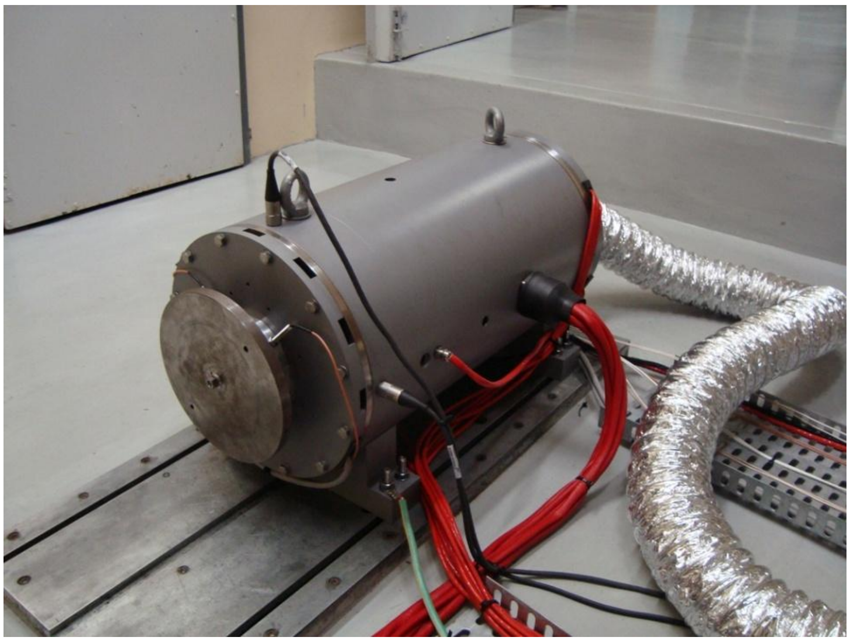


Figure C-1: Rotor de-levitation assembly

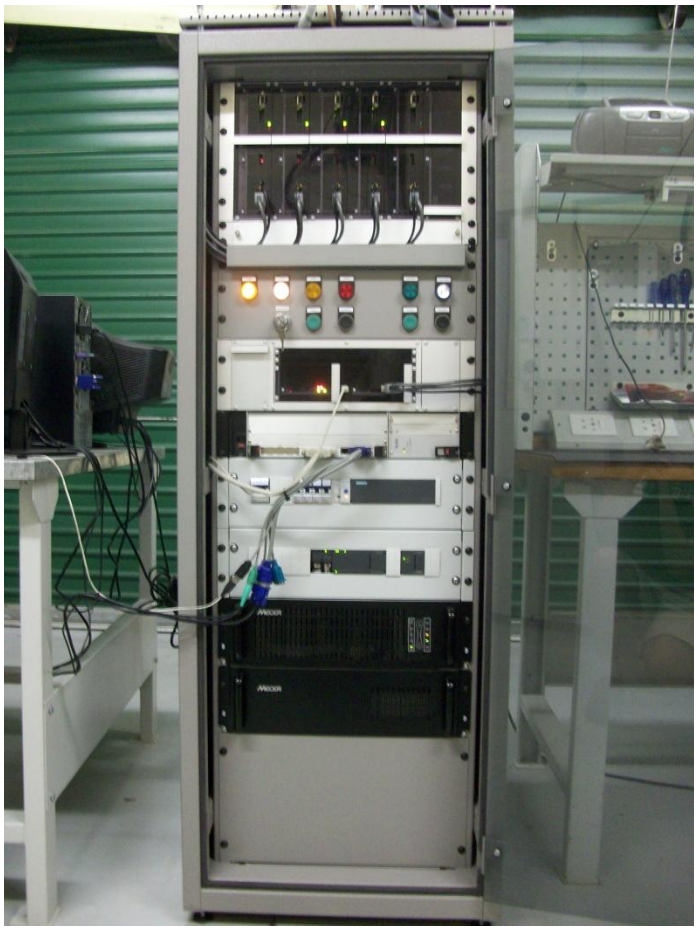


Figure C-2: ADES control enclosure

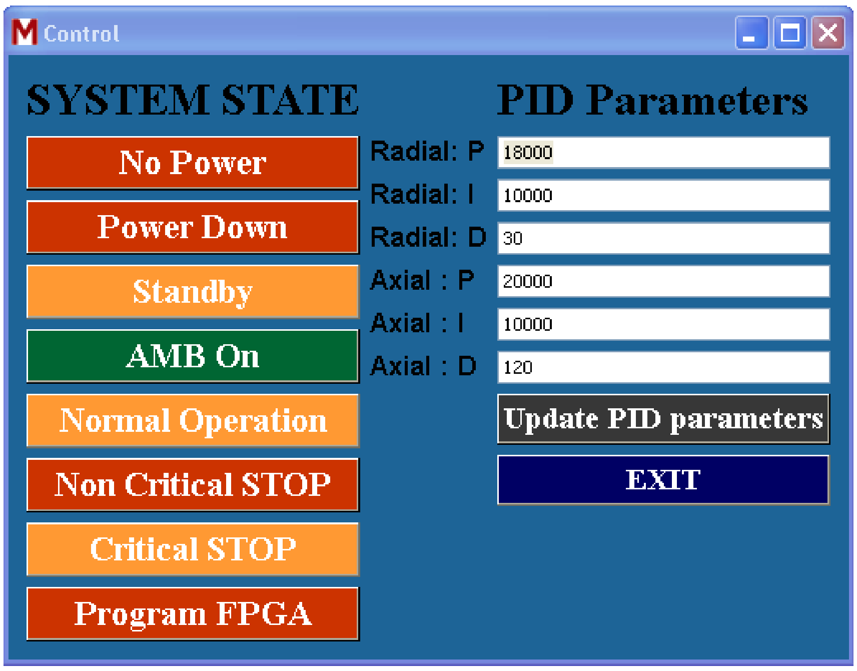
Appendix E: Graphical user interface

The graphical user interface (GUI) was developed using an open-source cross-platform application development framework called Qt[39]. The development environment used was Microsoft® Visual Studio® 2005 Professional edition. The development of the GUI was a rapid prototype to illustrate functionality. The functionality of the GUI will be replaced by the PLC and remote access interface.

The control window is seen in Figure D-1. The buttons on the left-hand side of the window is used to change the states of the system. The colour coding scheme used on the buttons are:

- Green indicates the current state
- Yellow indicates a possible state transition
- Red indicates an invalid state transition

The PID parameters are updated using the right-hand side textboxes. These parameters may only be updates while the system is in the "*Standby*"-state



Radial: P	18000
Radial: I	10000
Radial: D	30
Axial : P	20000
Axial : I	10000
Axial : D	120

Figure D-1: Graphical user interface: control window

The graphs indicating the position of the rotor in the x and y-axis of both radial AMBs, are shown in Figure D- 2. The orbital movement of the rotor is because the screens were captured at 4,000 r/min.

[39] Pronounced : "cute"

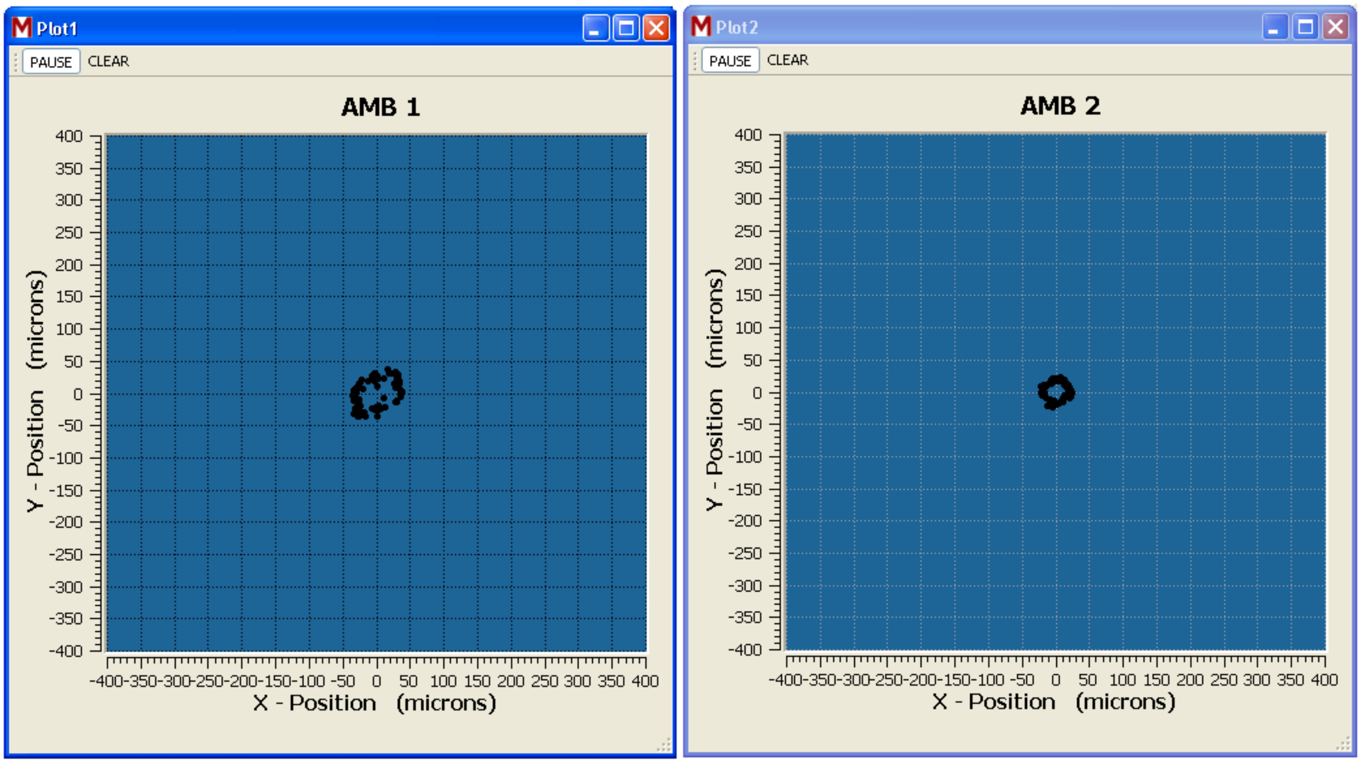


Figure D- 2: Graphical user interface: rotor position graphs

Figure D- 3 illustrates the true currents sent from the power amplifiers. As can be seen, 10 currents are received. Current 1 and 2 are received from power amplifier 1, current 3 and 4 are received from power amplifier 2, etc. The ADES was not used to suspend the rotor axially and thus are current 6 and 7 equal to zero. The axial position is also monitored and displayed.

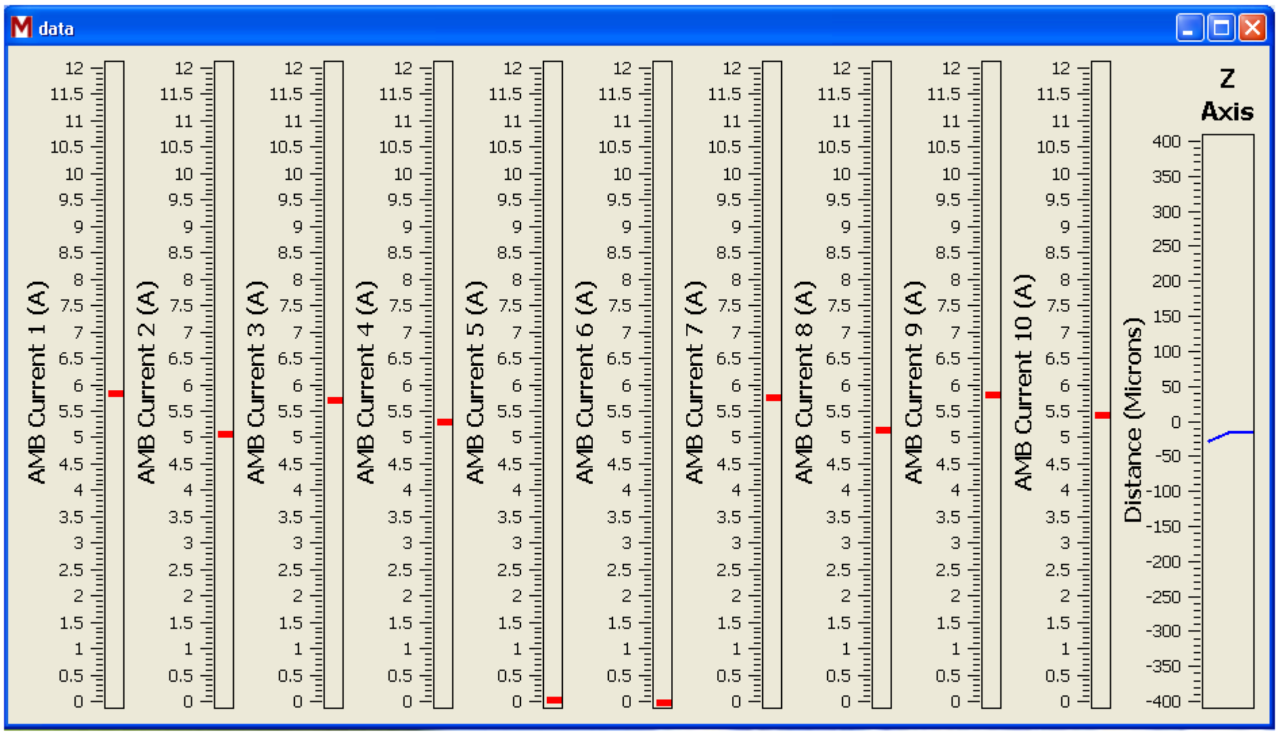


Figure D- 3: Graphical user interface: True currents and z-position

Appendix F: Data disc

The structure of the data on the disc is as follows:

F.9 Documentation

F.9.1 Project proposal

F.9.2 Dissertation

F.10 Photos

F.11 References[40]

[40] Not all references are available on the disc.