



MASTER OF SCIENCE
IN ENGINEERING

Hes·SO

Haute Ecole Spécialisée
de Suisse occidentale

Fachhochschule Westschweiz

University of Applied Sciences and Arts
Western Switzerland

Master of Science HES-SO in Engineering

Av. de Provence 6

CH-1007 Lausanne

Master of Science HES-SO in Engineering

Orientation : Technologies industrielles (TIN)

A real-time FPGA based monitoring and fault detection processing system for the Beam Wire Scanner instruments at CERN

Fait par

Jonathan Emery

Sous la direction de

Prof. Dr. Yann Thoma, REDS, HEIG-VD

Avec la collaboration de

Prof. Maurizio Tognolini, IAI, HEIG-VD

Fait en collaboration avec

l'Organisation Européenne pour la Recherche Nucléaire (CERN)

Dr. Christos Zamantzas, Beam Instrumentation Group, CERN

Expert externe

Prof. Dr. Marco Mazza, HEIA-FR

Genève, HES-SO//Master, 2017



Accepté par la HES-SO//Master (Suisse, Lausanne) sur proposition de

Prof. Dr. Yann Thoma, conseiller de travail de Master

et

Jonathan Emery, candidat.

Prof. Dr. Yann Thoma

Conseiller

Prof. Dr. Guido Frosio

Responsable de la filière

Contents

1	Introduction	11
1.1	Beam wire scanner instrument description	12
1.2	Thesis motivation	13
1.2.1	Position determination failure during beam time	13
1.3	Thesis subject	14
2	System description	15
2.1	Architecture	15
2.2	Electromechanical scanner and its motion	16
2.3	Sensors	18
2.4	Control electronics	19
3	Failures mode analysis	21
3.1	Failure report from the previous generation of Wire-scanner system (2009-)	22
3.2	Procedure of the FMA analysis	22
3.3	Actuation system description and items definition	23
3.3.1	Functional description	23
3.3.2	Item definition	25
3.3.3	Interfaces and interdependencies of the items	25
3.3.4	operational modes	25
3.3.5	mission phases	26
3.4	FMA classification for the system under consideration	27
3.4.1	Severity category and consequences	27
3.4.2	Occurrence probability	27
3.4.3	Detection probability	27
3.5	Failure modes and effects worksheet	28
3.5.1	Worksheet filling recommendation	28

3.5.2	Worksheet	29
3.6	Fault Tree Analysis	31
3.6.1	Existing system	31
3.6.2	Future system using sensor combination	32
3.7	Failure detection methods description	33
3.7.1	Resolver	33
3.7.2	Incremental Optical Position Sensor (IOPS)	34
3.7.3	Current transducer	34
3.7.4	Sensor fusion	34
3.8	Conclusion of the failure mode analysis	35
4	Methodology and design tools chain	37
4.1	Scripting tool for algorithms prototyping	38
4.2	Implementation method	39
4.3	VHDL development and verification	40
4.3.1	Verification alternative: Cosimulation Modelsim - MatLab	42
4.4	Tool for synthesis and target implementation	43
4.5	Numerical representation on the target	43
4.6	Fixed point library	44
5	Available field data sets	47
5.1	Example of data sets	48
5.2	Failure cases artificially generated	50
6	Data processing overview	53
6.1	Input value monitoring principle	53
6.2	Data flow hierarchical view	54
6.2.1	Resolver	55
6.2.2	IOPS	55
6.2.3	Motor phases current	56
6.2.4	Sensors fusion	57
7	Solid rotor resolver based absolute encoder	59
7.1	Technology	60
7.2	Resolver connections	61
7.3	field data characterisation	62

7.4	Processing algorithm prototyping	64
7.5	VHDL Implementation	65
7.6	Synthesis	66
7.7	Verification	69
7.8	Algorithm implementations comparison	70
7.8.1	Failure case simulation	71
8	Optical fibre based high precision incremental encoder	73
8.1	principle	73
8.1.1	Incremental and absolute encoding technique	77
8.2	Optical disk pattern optimisation	78
8.2.1	Simulation of the optical process	78
8.2.2	Patterns optimisation	81
8.2.3	Absolute angular position encoding into the track	82
8.2.4	Finalisation of the simulator for the detection algorithm development	84
8.2.5	Disk mask generation for the photolithography process	86
8.3	field data characterisation	88
8.4	Processing algorithm prototyping	90
8.4.1	fixed threshold based decision	91
8.4.2	adaptive threshold based decision	93
8.4.3	Adaptive detection of the maxima	94
8.4.4	Derivative based variable period detection	96
8.4.5	Summary of the algorithms development	99
8.5	VHDL implementation	100
8.6	Synthesis	102
8.6.1	Verification	104
8.7	Algorithm implementations comparison	106
9	Hall effect based current transducers	109
9.1	Technology and accuracy	109
9.2	field data characterisation	111
9.3	Processing algorithm prototyping	111
9.4	VHDL implementation	112
9.5	Synthesis	113
9.6	Verification	114
9.7	Algorithm implementations comparison	115

10 Sensors data fusion	119
10.1 Algorithm design	120
10.2 Architecture	121
10.3 VHDL Implementation	122
10.4 Verification	123
11 Top sensors processing	125
11.1 Verification	126
11.1.1 Failure cases simulations	129
11.2 further work	131
12 Common components and implementation summary	133
12.1 Common components	133
12.1.1 data flow control and standard interface between components	133
12.1.2 Convolution core for filtering	135
12.1.3 median filter for raw and sfixed data	137
12.1.4 Monitoring of raw and sfixed values	140
12.2 Implementation summary	140
13 Module integration into the PSB prototype firmware	143
14 Project management	147
15 Conclusion	151

Acknowledgements

I would like to express my very great appreciation to my thesis director Prof. Thoma of the Reconfigurable and Embedded Digital Systems institute (REDS), part of the School of Management and Engineering Vaud (HEIG-VD) for his wise guidance during this thesis and all the advanced and valuable suggestions in digital systems design and verification he gave me along the way. In addition, this thesis would not be the same without having followed his module Vérification des Systèmes Numériques (VSN), part of the Master of Science in Engineering (MSE) of the University of Applied Sciences of Western Switzerland (HES-SO).

I would also like to thanks for the collaboration of Prof. Tognolini of the institute of Industrial Automation (iAi), part of the HEIG-VD for his expert advices in the domain of signal processing, electronics and sensors during this thesis. In addition, this thesis would not be the same without having followed his module Traitement statistique du signal et modélisation (T-StatDig), part of the of the HES-SO.

I wish to especially thanks Dr. Zamantzas, expert in digital systems and section leader at the European Organization for Nuclear Research (CERN), who gave me very valuable organisational and technical suggestions during this thesis.

This thesis would not be possible without the support and environment of the CERN Beam Instrumentation group (BE-BI) and its leader Dr. Jones and the Profile Measurements section (BE-BI-PM) with its leader Dr. Roncarolo, where I could realised this thesis.

Last but not least, I would like to warmly thanks my family for their support which made this thesis and the whole Master time possible.

Abstract

The CERN Beam Instrumentation group (BE-BI) is designing a new generation of an instrument called Beam Wire Scanner (BWS). This system uses an actuator to move a very thin wire through a particle beams, back and forth with a movement stroke of π [rad]. To achieve very fast speed when touching the particle beam with such a small stroke, large torque is applied while the expected smoothness of the displacement is crucial.

This system relies on a resolver to determine the angular position and power correctly its Permanent Magnet Synchronous Motor (PMSM). In 2016, a failure of the position acquisition chain has highlighted the severe consequences of such problem, which resulted by 24 hours downtime of the Super Proton Synchrotron (SPS) accelerator with significant financial consequences.

This work mitigates this single failure point by taking advantage of the existing redundancy in the sensors embedded on the system. The resolver is compared to two Incremental Optical Position Sensor (IOPS) developed in-house and redundant motor currents sensor are allowing failure detection of this part. The processing capability of the large Field-Programmable Gate Array (FPGA) present on the actuation controller is also taken advantage of. The resulting system is more robust against sensor failures and provides more reliable measures.

An emphasis is made to the methodology of the design and development to ensure the high confidence in the results. The prototyping algorithms have been developed using the software MatLab prior to their implementation in VHDL Hardware Description Language (VHDL). The simulation vectors are coming from a field measurements recorder prior to this thesis and extra failure modes have been simulated by creating artificial data starting from these existing sets. The output produced by the algorithms written in the two languages have been finally compared to detect differences due to eventual implementation issues or number representation limitation.

The processing architecture developed during this project is able to detect numerous sensor acquisition chain failures. By processing and monitoring each sensor individually, many issues can already be detected. The final detection is made by comparing the value delivered by these sensors. In addition, one sensors fusion architecture is implemented to provide position and speed based on redundant information.

In addition to the work on the algorithms, during the design for the processing of the IOPS, a new type of optical disk was developed which could overcome the limitation of this sensor. By embedding the rotation direction and the absolute position directly into the unique mirrors track, this incremental encoder could be turned into a absolute encoder and improved the system reliability.

The outcome of this work will allow to deliver a more reliable and robust system based on complex technologies when compared to previous Beam Wire Scanner (BWS) generations. Giving to the accelerators operators and users high performance with better availability.

Chapter 1

Introduction

The CERN Beam Instrumentation group (BE-BI) is designing a new generation of an instrument called Beam Wire Scanner (BWS). This system uses an actuator to move a very thin wire through a particle beams, back and forth with a movement stroke of π [rad]. To achieve very quick speed when touching the particle beam with such a small stroke, large torque is applied while the expected smoothness of the displacement is crucial.

Previous generation of this instrument have had only one position sensor, usually a resistive potentiometer, to control the angular shaft position of the motor and provide an estimated wire position over time.

The new system is embedding one resolver for the feedbacks and one custom, high accuracy Incremental Optical Position Sensor (IOPS) to determine the position of the wire. In addition, since the actuator uses a Permanent Magnet Synchronous Motor (PMSM), three currents are generated to move the system, which makes it also possible to estimate position.

The finality of this thesis is to find an efficient way to monitor and detect eventual failures of these sensors. A second goal is to possibly combine their measures in order to, increase the reliability of the position and speed determination using available redundancy, and to improve the quality of the estimation fed to the control algorithm.

This introduction chapter will present the basics of the BWS, describe the motivation for this work and finish with the thesis subject as presented for its acceptance.

In Chapter 2, a detailed description of the BWS system will be given, with its various parts introduced such as the mechanics and its motion, the sensors and the control and power units.

Chapter 3 will present the Failure Mode and Effects Analysis (FMEA) performed at the sensors functional level. This analysis includes various table definitions and material required by the methodology introduced in the document ECSS-Q-ST-30-02C [1] edited by the European Cooperation for Space Standardization (ECSS). This chapter concludes with recommendations for the design of the detection and mitigation processes developed in this thesis.

The development methodology and the tools to achieve the goals of the thesis are described in Chapter 4. In addition to describe the steps followed for designing, developing and implementing the design, the Computer-Aided Design (CAD) tools are introduced. The last part of the chapter introduce the representation of numbers which is a key challenge when implementing signal processing inside Embedded systems.

The Chapter 5 is presenting the available data sets previously recorded with an early version of the system. As no data sets presents certain failures that the processing should detect, simulation of sensors failures based on real data is performed and described in this chapter.

The main concept of the data processing developed in this thesis is available in Chapter 6. This processing is presented in an hierarchical way, the processing modules are defined, named and numbered to ease the referencing in later chapter.

The three next chapter are describing in details the work preformed on each type of sensors, Chapter 7 for the resolver, Chapter 8 for the IOPS and Chapter 9 for the currents transducers. Each of these chapters are

presenting the physical sensor, available data, processing prototyping with MatLab, implantation in VHSIC Hardware Description Language (VHDL) and verification, to finish with the synthesis for the target device.

Chapter 10 presents the fusion of the approximation of the position and speed obtained from the pre-processing of the sensors.

The Chapter 11 describes the verification of the complete processing core developed.

The components which are used for multiple purposes are described in Chapter 12 along with an summary table of the modules developed in VHDL.

The Chapter 13 concludes the technical part by presenting the integration of the top processing unit inside the Field-Programmable Gate Array (FPGA) of the early design of the BWS

Chapter 15 will conclude this thesis dissertation by presenting the achieved work and the perspectives for further evolution of this subject.

1.1 Beam wire scanner instrument description

The Beam Wire Scanner (BWS) belongs to the category of interruptive beam transverse profile measurement instruments. It uses electro-mechanical parts to move a thin wire of about $30\mu\text{m}$ through a particle beam as shown on the top-centre of Fig.1.1. The secondary particles generated by the interaction of the beam with the wire escapes the beam pipe and are detected by a sensor based on scintillation material associated with a photomultiplier (right of the figure). The generated current is proportional to the beam intensity at the position of the wire, which allows the reconstruction of the transverse beam intensity profile (bottom-left). The raw data are processed to obtain the beam size, by fitting it with a Gaussian function and extracting the value of the sigma.

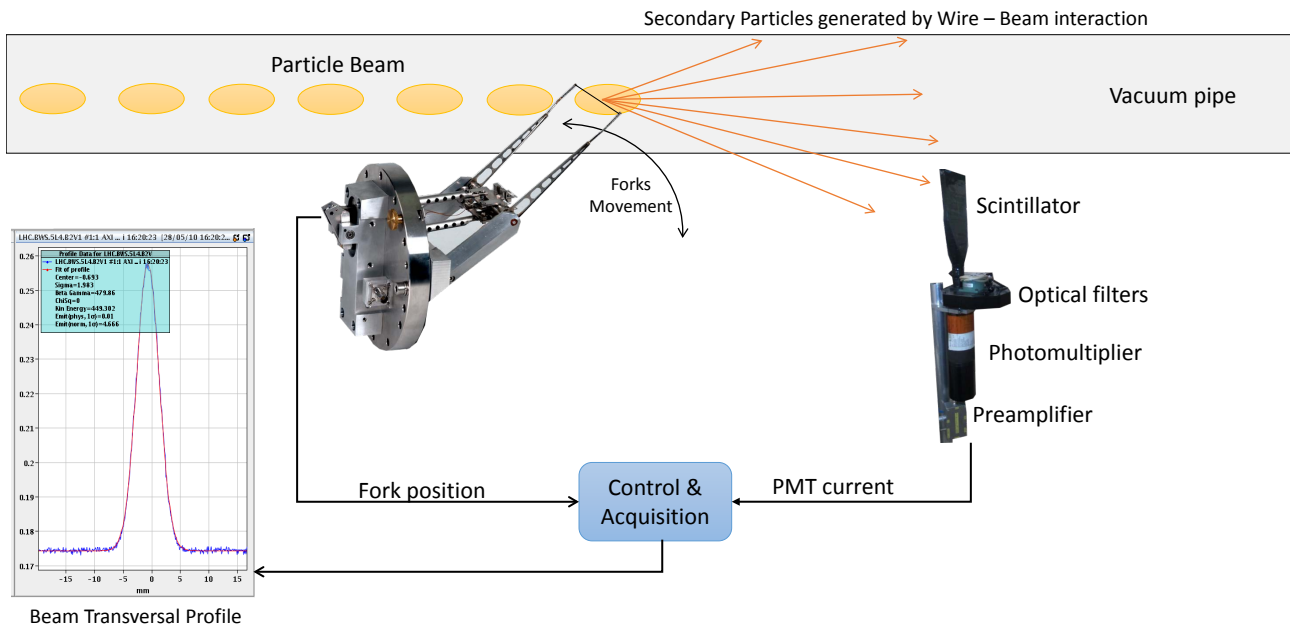


Figure 1.1: Wire-scanner working principle

1.2 Thesis motivation

The first implementation of the FPGA control system for the prototypes scanner was developed in 2014 - 2015 and was successfully used for beam measurements in the Super Proton Synchrotron (SPS). Multiple tests campaign have been performed and the recorded data have been used for this thesis. Nevertheless, one unexpected failure described in 1.2.1 had important consequences for the accelerator and should be avoided in the future.

1.2.1 Position determination failure during beam time

In 2016, a campaign of measurements with the SPS prototype scanner and its associated electronics was conducted. Unfortunately, on the 9 September 2016, an electrical fault provoked an incorrect position measure while being under position control mode. Due to this miss understanding, it believed it was off by multiple rotations and then started to move backwards at maximum speed for multiple revolutions. This fast spinning provoked a loss of material (piece of wire) from the forks which stopped in the middle of the beam pipe. This inhibited particle beams injection inside the accelerator pipe. The SPS had to stop for about 24 hours to open the vacuum and remove this obstructing material, which resulted by a large financial loss for the CERN accelerator complex.

Report from the investigation:

1. While testing the prototype scanner without beam, an error occurred in the processing of the shaft position.
2. This problem had never been observed during laboratory testing. The cause is to be found in interconnection failure between electronics components and/or FPGA code corner case. It could be linked to new electrical connection to an external system for hardware synchronisation (triggers).
3. This error induced the control system in believing the position was off by multiple rotations forwards.
4. Due to this situation, the position control feedback spin the scanner by multiples rotations backwards at the highest speed as possible, with a top speed of 170 [rad/s].
5. Due to this many rotations, the cables (measuring the thin carbon wire) have been bend to breakage (See left of Fig.1.2) and one part was ejected away from the forks.
6. This cable stopped at the edge of the tank in the middle of the beam pipe as illustrated on the right of Fig.1.2.

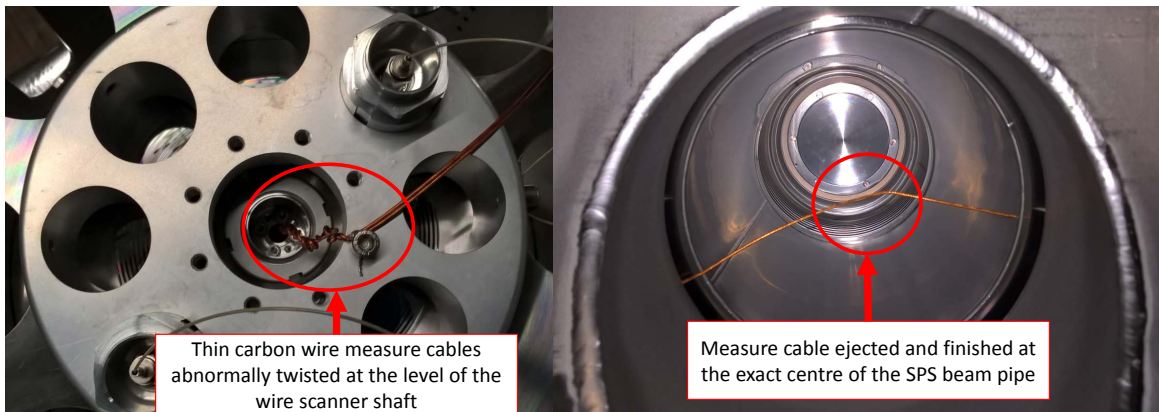


Figure 1.2: Illustration of the wire-scanner breakage due to loss of correct position

Mitigations:

1. Detection and limitation implemented in the code of the prototype: Firmware release to avoid processed position to be larger that 2 rotations (12 rad). The scanner is not made for multiple rotations. The scan does only half revolution and returns to its initial position.

2. Position coherence check and filtering implemented in the prototype code

Code re-factorisation for improved immunity to noise and ground bounce between the FPGA and the resolver chip.

3. Electrical fault minimisation by separating grounds with all external systems providing signals, such as triggers signalisation.
4. Investigate the use of redundant sensors for the angular measurement.

This last point is the starting point of this thesis.

1.3 Thesis subject

The thesis title is "A real-time FPGA based monitoring and fault detection processing system for the Beam Wire Scanner instruments at CERN". The main goal is to develop signal processing and algorithm to determine the health of the system using the sensors (monitoring). The second goal is to combine the sensors information to improve the reliability of the position and speed determination and avoid system failure as presented in subsection 1.2.1.

Fig.1.3 presents a blocs diagram of the sensors combination which results by an single output of the position and speed based on all the sensors. this approach improves the overall reliability of the system by providing a verified position and speed between the sensors, as well as detecting failure and incoherences.

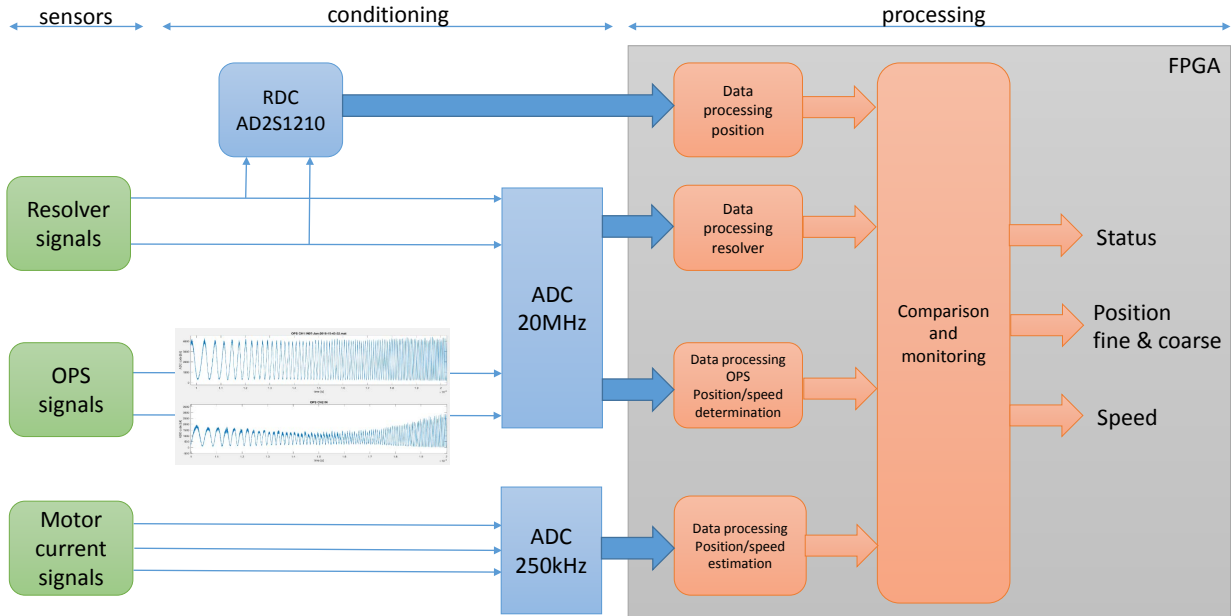


Figure 1.3: Sensors combination overview

The methods used to do this project are:

1. Analysis of the system architecture (sensors and control system) and identification of possible failures modes and scenarios, severity and consequences.
2. Develop ways to detect failures bases on sensors data signal processing
3. Implement signal processing prototypes with real measurements using MatLab tools
4. Translate them in VHDL to be integrated into a FPGA
5. Build a verification environment which compares the two implementations (MatLab and VHDL)

Chapter 2

System description

This Chapter describes the Beam Wire Scanner (BWS) architecture, from the physical part and its motion to the sensors and from the control electronics to the power part.

2.1 Architecture

The architecture of the BWS system is visible on Fig.2.1. The electro-mechanical part (lower-right) and the secondary particle shower detector (lower-left) are installed in the accelerator underground tunnel where the beams are travelling. The intelligent drive, which controls and power the scanner (top-right) and the acquisition and supervision (top-left) are situated on the surface buildings, fare away from the destructive effect of the ionizing particles. The distance between these two parts can be as fare as 300 meters and introduce perturbations and parasitic coupling on the sensors measures. there is no semiconductor material foreseen close to the mechanism to avoid any failure due to destruction of these type of components.

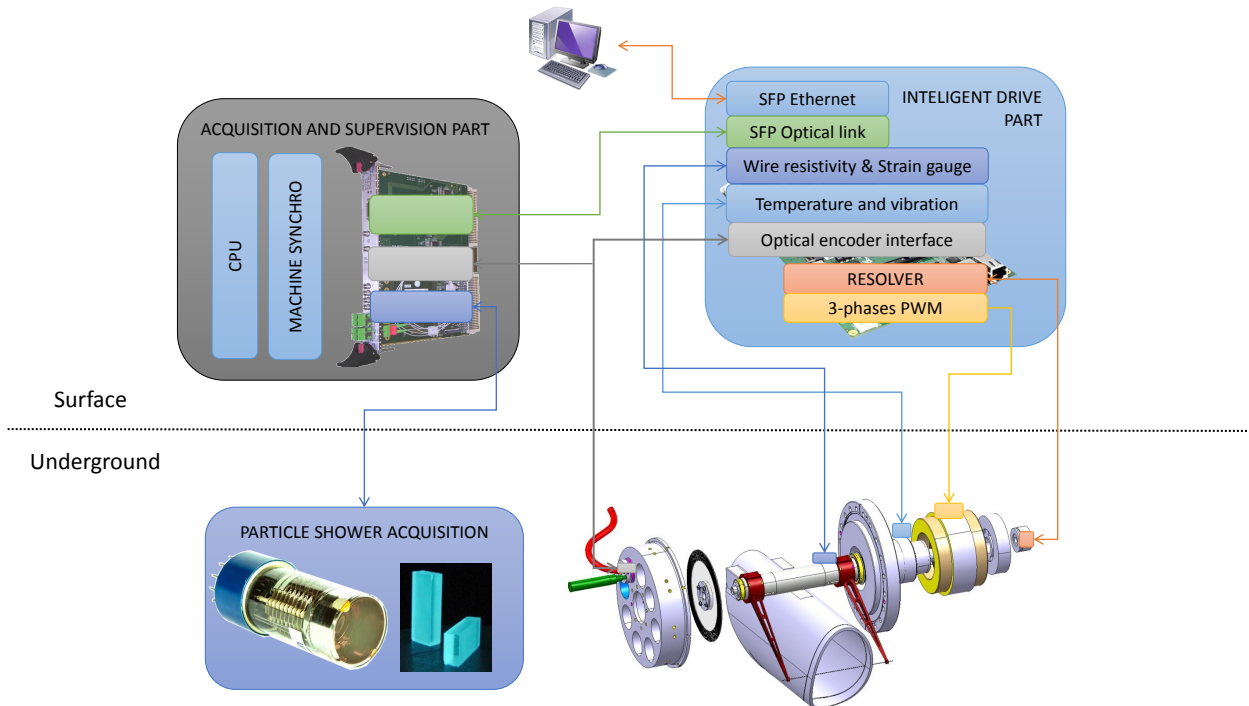


Figure 2.1: Wire-scanner control and acquisition architecture

2.2 Electromechanical scanner and its motion

An important part of the wire-scanner is the mechanism responsible to move the wire through the beam. The motion is produced with a motor and is transmitted to the forks with a shaft. A crucial difference between this design and the previous generations is that the motion is produced directly inside the vacuum

Previous generations of the wire-scanner are using vacuum bellows to transfer the movement from atmosphere side where the motor is located to the vacuum side. The concept of the wire-scanner new generation is that all the moving part are located in the vacuum where the particle beam is travelling. The stator of the motor is situated in the air side and the magnetic energy is transmitted through the protecting thin wall to the rotor.

The motor type is a Permanent Magnet Synchronous Motor (PMSM) for its high power/inertia ratio. The system needs to achieve very high acceleration, so an optimum between torque and inertia has to be found. A study dedicated to the design of a custom motor has determined the optimum geometry and technological choice as well as producing a survey of existing devices [2]

The angular position of the shaft need to be know for two purposes. Firstly, the absolute position of the motor rotor need to be known in order to correctly power the three phases of the stator and control the magnetic flux to produce the right torque. Secondly, since the shaft holds the forks from where the wire is fixed, its angular position needs to be determined very precisely. Since these two characteristics, absolute position and high precision, are difficult to find in a single sensor, decision was taken to use one sensor for each functionality.

The absolute position is given by a resolver which uses the variation of the coupling between two windings on the stator to determine it. This technique avoids rotor transformers making it compatible with ultra high vacuum.

The incremental encoder used is the IOPS which uses optical signal travelling through single mode optical fibre to get information of the shaft angle. the laser light and detection is performed far away from the mechanism. A glass disk coated with very thin reflecting slits produces signals when the shaft is rotating [3]. This encoder is able to produce very precise incremental information to measure accurately the beam size.

The design of this mechanism allows very low static friction which is very good for the system precision. The drawback is that the shaft is able to move when the position control is off. For this reason, a magnetic brake using the reluctance principle was developed in [2] to fix two stable position. One is called "HOME" and is far away from the beam, the other is "IN" the beam (the particle beam passes between the forks). If for any reason the shaft is left in the middle of these two stable position, the shaft will be "attracted" by the closest stable position.

The thin carbon wire is conductive with a resistance of few kilo-ohm. It is possible to measure this value to determine its integrity and view the effect of the interaction beam-wire. For this, vacuum compatible shielded copper cables are routing the potential of each side of the thin wire to an electrical feedthrough to get the signal out of the vacuum region to the air side.

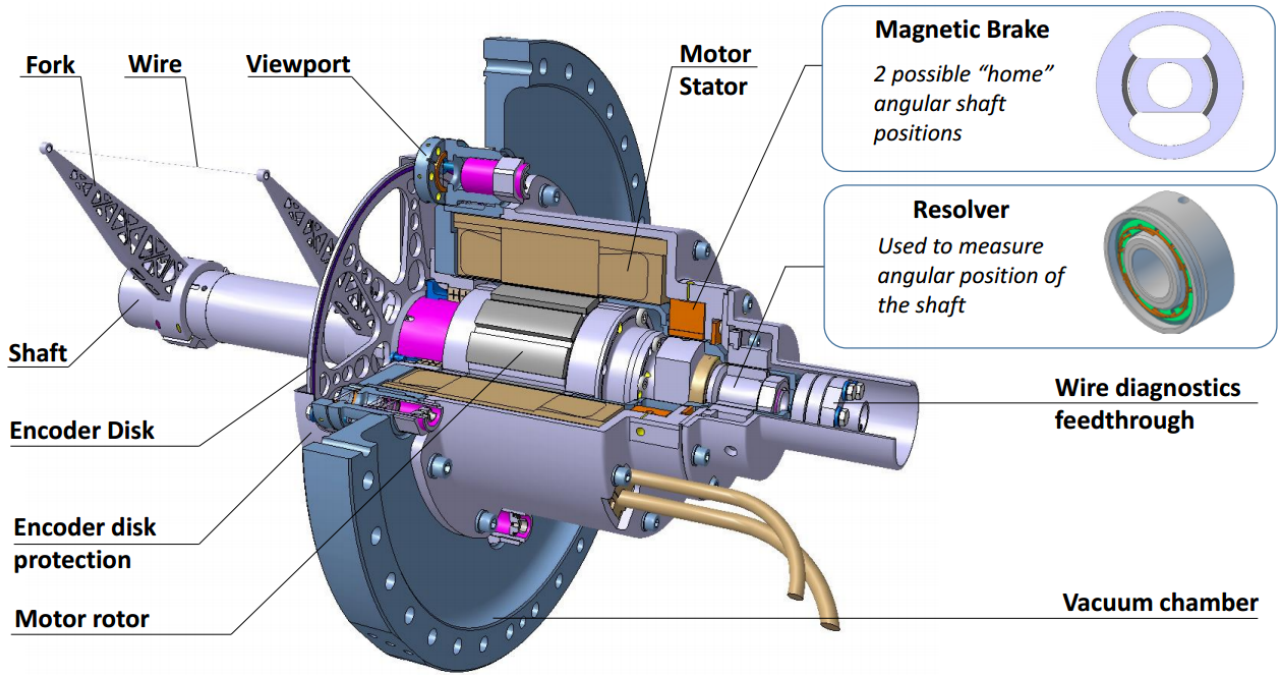


Figure 2.2: Electromechanical mechanism of the wire-scanner (Courtesy of D. Gudkov, BE-BI-ML)

the motion performed during a so called measurement cycle is a half rotation in one direction to cross the beam and then to return to its initial position. Fig. 2.3 shows the motion profiles for a displacement of π , from 'HOME' to 'IN' positions. The return movement is produced by reading the tables backwards. The movement is created from a sinusoidal jerk (top plot). This profile is then integrated to give the acceleration profile (second plot). The speed of the third plot is obtained by integrating the acceleration. Finally the position of the fourth plot is calculated by integrating the speed.

These curves are stored inside the motion controller based of FPGA and are accessible to the monitoring processing developed in this thesis.

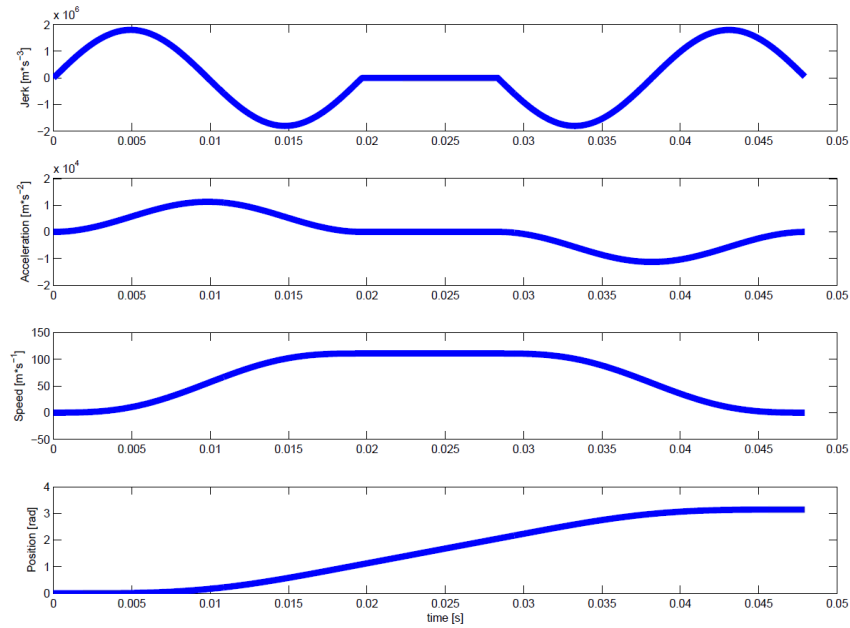


Figure 2.3: Motion profiles of the wire-scanner

The motion described consider a the tangential speed of the wire of 20 m/s. The wire travels at this speed turning around a circle, but an observer will not see this speed. The projection of this motion on the perpendicular plane of the beam direction will show a lowering speed as the fork goes away from an horizontal

position. This effect is presented in Fig.2.4 with the projected position (top) and projected speed (bottom). This means that the wire will stay longer in the particle beam as it enters with a angle different than 90 [deg].

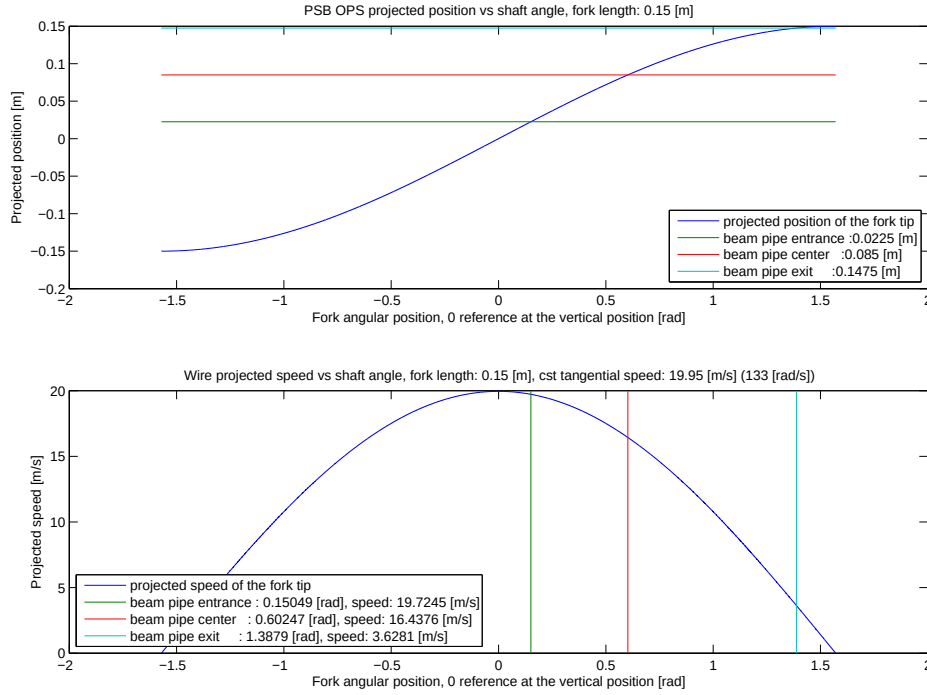


Figure 2.4: PSB wire scanner projected position (top) and projected speed (bottom) vs angular position of the scanner shaft

2.3 Sensors

The sensors available for processing in the frame of this thesis are of three types. There are dedicated chapter for each of them in this thesis.

The solid rotor resolver is an angular measurement sensor which is providing a measure of the shaft angle. It provides an absolute information in the range of one complete rotation, i.e. it can't provide an information over multiple rotations. This information is used to correctly power the motor. To obtain the multi-rotation information, the measure needs to be integrated by the processing unit to overcome position jump from $2 \cdot \pi$ [rad] to 0 and 0 to $2 \cdot \pi$ [rad]. Chap.7 is dedicated to this sensor.

The high precision incremental encoder called IOPS is an in house sensor which was developed at CERN. the principle uses optical fibre reflection to determine when a disc has moved of an small angle. The resolution of such device can be very small with the drawback of not having the absolute position. This needs to be determined by processing the signal. The digitalisation of the return light power is available to the control system based on FPGA which allows real time processing of this signal. This subject is treated in Chap.8

The last sensor type available to the control system is the current transducers measuring the three motor phases currents. These informations are used to actuate the motor properly, as we need to accurately control the stator flux to regulate the torque produced. The scheme with three sensors have some redundancy that can be used to detect eventual failure of these sensors. Chap.9 is dedicated to this sensor.

2.4 Control electronics

The control electronics is based on an FPGA platform with the Altera Arria V product and an in-house mezzanine performing the analogue conditioning and digitalisation of the various signals. It is this type of board that has been used to record the various data sets that are treated in this thesis. Fig.2.5 shows a picture of it.

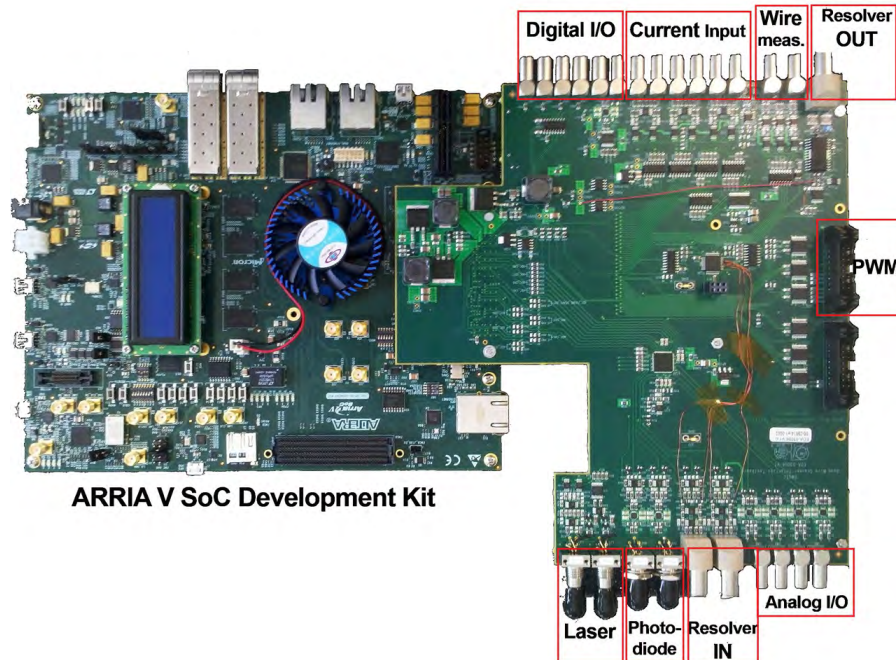


Figure 2.5: Prototyping electronics used for controlling the wire-scanner system and recoding the data sets processed in this thesis

Chapter 3

Failures mode analysis

The purpose of this chapter is to analyse the wire-scanner actuation design using a systematic approach developed for critical application such as space missions, airborne systems or military products.

Failure Mode Analysis (FMA) is a methodology used to determine the reliability of a system or process by analysing its parts, components, assemblies, and subsystems. It can be used to determine the likeliness of problem and its effects on a complete system based on the single failure using FMEA worksheet or combination of them using Fault Tree Analysis (FTA).

The FMEA is focussed only on the effect of a failure while the Failure Modes, Effects and Criticality Analysis (FMECA) introduces the criticality of such effects. The reference document for this analysis in the frame of this work is "Space product assurance - Failure modes, effects (and criticality) analysis (FMEA/FMECA)" (ECSS-Q-ST-30-02C [1]) edited by the ECSS.

As such analysis can be a complete thesis subject of its own, which could include the complete system analysis and its implementation down to single component failure modes, a focus is given to the functional analysis of the actuator system. This part is responsible of moving the scanner and has potentially the ability to brake part of itself in certain cases.

The motivation to follow such investigation is to use of formal method to determine failures scenarios, the severity of these events and to be able to develop ways to detect them, or even avoid them by the use of redundancy.

The FMEA, by which each potential failure mode in a product (or function or process) is analysed to determine its effects is used. In this analysis, the potential failure modes are classified according to their severity. The FMECA, is a FMEA extended to classify potential failure modes according to their criticality.

The standard defines also functional analysis, where the functions of the system, rather than the items used in their implementation, are analysed. This approach will be used for the wire-scanner.

Before going into the analysis of the system under consideration, the next section resume the failures that the operational BWS system have suffered these last years. this analysis has driven some technical choices for the redesign that is under consideration in this thesis

Events	Failure description	Failure Origin	Failure severity	Mitigation	Redesign objectives
1x LHC, 2x PS	Mechanical bellow failure	Used outside specification	High, operation stopped for 24h	Change of bellow type	Remove need for bellows
2x LHC	Wire broken	CPU crash	High, Unavailable for weeks	Wire exchange	Change system structure
2x SPS	Wire broken	VME protocol incompatibility	Medium, Unavailable for weeks	Delay CPU upgrade	Change system structure
2x LHC, 1x PS	Movement not possible	Multiplexer relay failures	Medium, Accelerator access needed	Relay exchange	No multiplexer
16x various	Movement stopped	Linear driver burned	Medium Unavailable for hours	Component exchange	Use of inverter
Often	Wrong measurement	Bad acquisition settings	Low, new measurement required	Better measurement protocol	Extend dynamic range for the acquisition

Table 3.1: Operational scanners failures report (2009-2012) from [4]

3.1 Failure report from the previous generation of Wire-scanner system (2009-)

The major scanner failures that occurred during last years of operations, between 2009 and 2012, can be classified into four different severity levels as reported in [4]

The highest severity means that the accelerator has to stop until the issue is fixed, which in the case of the wire-scanner, is typically due to a vacuum leak.

At the next highest severity level, the instrument needs an intervention inside the vacuum and is not operational for a long period until access is granted.

Lower severity levels are linked to failures on the surface electronics that can be fixed in a matter of hours. The lowest severity is allocated when the scanner scans but fails to acquire data. This occurs regularly and requires a repetition of the mechanical movement that accelerates the system ageing, reducing the time interval for regular maintenance.

Table 3.1 lists the principal failures. Each of these failure scenarios is taken into account in the new design. To address mechanical failure, the system has been completely re-designed in order to eliminate the need for bellows, which are a source of high friction, play, backlash and fast ageing, and so incompatible with the high availability demands of today's accelerators.

For the electronics, the challenge is to reduce as much as possible the sensitivity of the control system to external perturbations. The first decision was to remove the multiplexers that allowed only one electronic system to control multiple scanners sequentially. The influence of the Central Processor Unit (CPU) on critical functions that can lead to failure of mechanical or electrical sub-systems is also avoided. The result is to have a self-contained independent control system that is able to survey the status of the scanner for continuous condition monitoring that can take protective actions when necessary. The sensitivity of the system to the linear power drivers used will be avoided through the use of a design based on inverters. The final major change concerns the type of secondary shower detection adopted, which will in future use sensors and electronics with an extended dynamic range.

3.2 Procedure of the FMA analysis

As described in the document [1], the FMEA analysis should follow theses steps:

1. Description of the product (i.e. function or hardware) to be analysed, by providing:
 - (a) functional descriptions,
 - (b) interfaces,
 - (c) interrelationships and interdependencies of the items which constitute the product,
 - (d) operational modes,

- (e) mission phases.
- 2. Identification of all potential failure modes for each item and investigation of their effect on the system under analysis. Assuming that each single item failure is the only failure, implying that combination of failures are not considered.
- 3. Evaluation of each failure mode in terms of the worst potential consequences and assignment into a severity category.
- 4. Identification of failure detection methods.
- 5. Identification of existing preventive or compensating provisions for each failure mode.
- 6. Provide for identified critical items corrective design or other actions (such as manual actions) necessary to eliminate the failure, or to mitigate, or to control the risk.
- 7. Document the analysis and summarize the results and the problems that cannot be solved by the corrective actions.
- 8. Record all critical items into a dedicated table as an input to the overall project Critical Item List (CIL).

Each of these steps will be developed in the following sections.

3.3 Actuation system description and items definition

This section is the first part of the FMEA analysis which describes functionality the system under consideration, identify its items, its operational modes and mission phases.

- (a) functional descriptions,
- (b) interfaces,
- (c) interrelationships and interdependencies of the items which constitute the product,
- (d) operational modes,
- (e) mission phases.

3.3.1 Functional description

The scope of this analysis is the actuation system of the so called Beam Wire Scanner (BWS) represented on the lower part of Fig. 3.1. The purpose of this electromechanical system is to perform movement cycles constituted of half rotation (π rad) in one direction, and half rotation in the opposite direction to return to its initial position. One complete revolution would be allowed as long as an complete revolution is performed in the other direction after it.

The mechanical power is given by a Permanent Magnet Synchronous Motor (PMSM) which requires the use of an external angular position sensor to correctly power its three phases.

The FPGA is responsible of processing the sensors inputs, storing them for post mortem analysis and calculate the outputs to apply to the controlled system.

The available sensor to perform this function is an resolver which provides two analog signals in quadrature which is an estimation of the absolute angular position. A commercial Application-Specific Integrated Circuit (ASIC) allows the Resolver to Digital Conversion (RDC) given to the FPGA. An alternative digitalisation path is also available to provide a direct digitalisation of the quadrature signals for monitoring purposes.

An second angular encoder called the Incremental Optical Position Sensor (IOPS) is also coupled with the shaft for high precision measurement of the shaft position targeting the accurate determination of the particle beam size, the primary purpose of this system. This encoder uses continuous low power laser coupled to an optical fibre to focus light at a wavelength of 1310 μ m to an optical disk featuring micrometer size mirror

reflecting the light back. The returning light is converted into voltage by means of photo-diodes and is digitalised at 20MHz. The maximum repetition rate of the mirror is close to 1MHz.

The last sensor type considered in the analysis is the motor phases current transducers which are providing the measure used for the feedback. It is to control the voltage applied to these phases by means of Pulse Width Modulation (PWM). There are one sensor for each phase while the only necessary are two device, the third phase is calculated as it is the sum of the two others.

The connection of the FPGA to the power stage is going through a small programmable device called Complex Programmable Logic Device (CPLD). This device is in charge of controlling the power stage and verifying the coherence of the PWM signals generated by the FPGA. this device in function as soon as power is up while the FPGA needs to load its internal configuration from an external memory device.

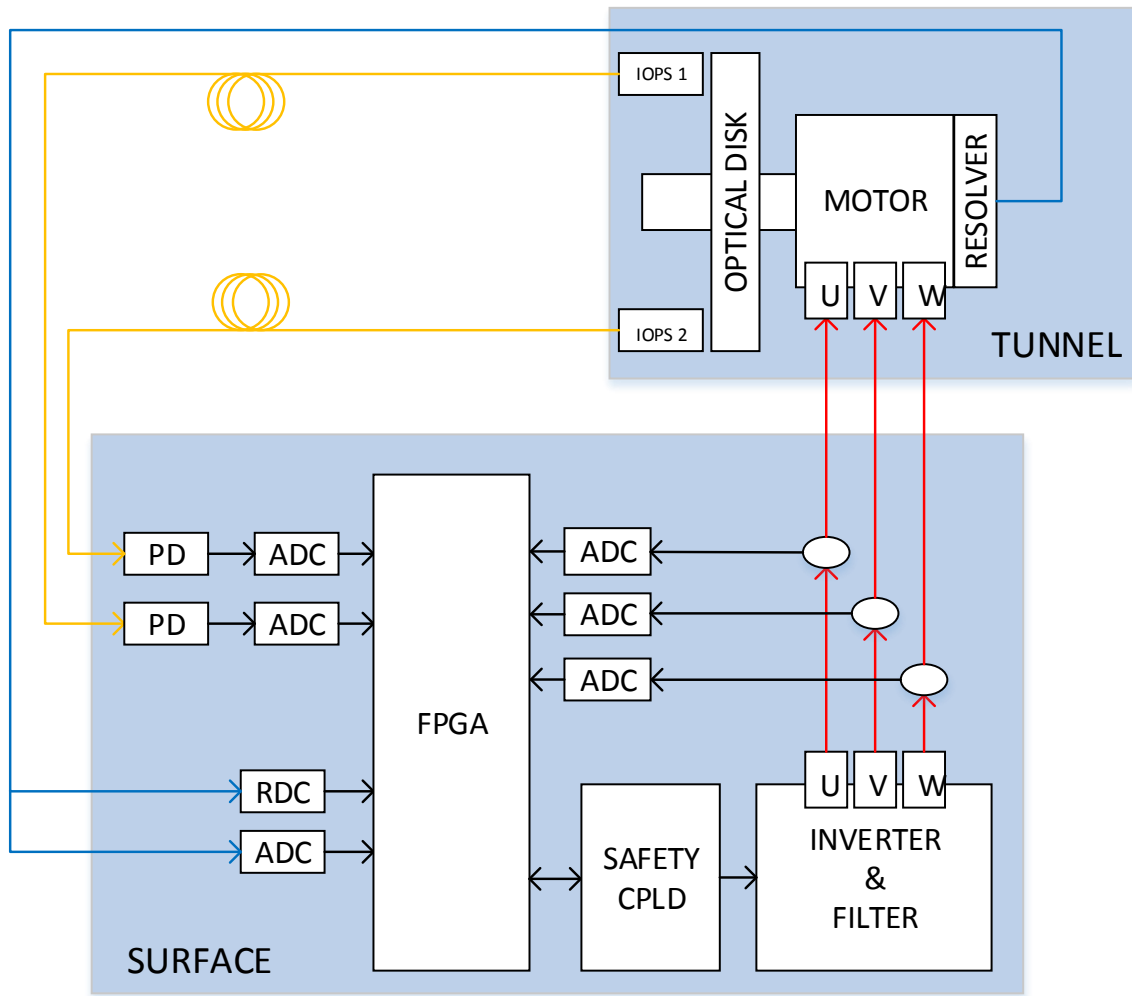


Figure 3.1: Block diagram of the actuator with its sensors

3.3.2 Item definition

The definition of the items of the actuation system is given in table 3.2.

Table 3.2: Item list for the Failure Mode Analysis

Item	Part name	Description
I01	RESOLVER	Absolute angular position sensor
I02	OPS 1	Incremental optical position sensor (Channel 1)
I03	OPS 2	Incremental optical position sensor (Channel 2)
I04	CT 1	Current transducer motor phase U
I05	CT 2	Current transducer motor phase V
I06	CT 3	Current transducer motor phase W
I07	FPGA	Central programmable logic controller
I08	SAFETY CPLD	Controller of the power stage
I09	POWER INVERTER	Three phases power bridge
I10	MOTOR	Motor mechanical and electrical part
I11	SHAFT	Connects the motor with the sensors

3.3.3 Interfaces and interdependencies of the items

The interfaces overview is presented in Fig.3.2. The interdependencies between the items is very strong, the FMA analysis will determine how to lower the interdependencies between these parts if possible.

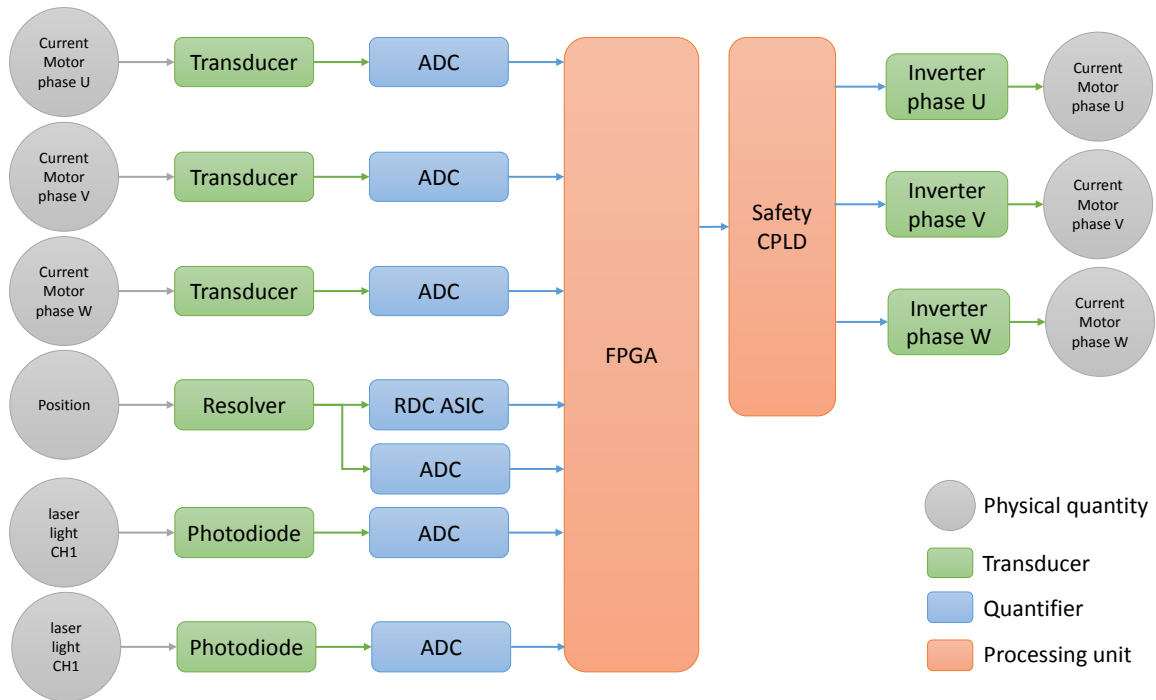


Figure 3.2: Block diagram of the actuator with its sensors

3.3.4 operational modes

The wire scanner has different mode of operation. These modes do not all require the sensors. This means that the severity of the failure depends in which mode the scanner is when the event happens and is detected. As defined in the reference document [1], only "the record of results is limited to the phase or mode in which the worst failure effects occur." The following list of operation modes highlights the operational mode under analysis.

- (a) OM1: feedback OFF

- (b) OM2: feedback ON

Cycle phases:

- (a) PH1: System at 'HOME' position, waiting for start trigger
- (b) PH2: System go to 'IN BEAM', acceleration phase
- (c) PH3: System go to 'IN BEAM', constant speed phase
- (d) PH4: System go to 'IN BEAM', deceleration phase
- (e) PH5: System at 'IN BEAM' position, waiting for return trigger
- (f) PH6: System go to 'HOME', acceleration phase
- (g) PH7: System go to 'HOME', constant speed phase
- (h) PH8: System go to 'HOME', deceleration phase

3.3.5 mission phases

The standard defines the following phases:

- 0 : Mission analysis or requirements identification
- A : Feasibility
- B : Preliminary definition
- C : Detailed definition
- D : Production or ground qualification testing
- E : Utilization
- F : Disposal

The phase of this thesis project is presently in the "**Phase B: Preliminary definition**"

and therefore the FMEA/FMECA shall:

1. Support the trade-offs from the dependability and safety point of view
2. Support the definition of the requirements to be implemented in the product as redundancies, inhibits, operations to be followed to avoid hazards or loss of mission, and others, such as fail-safe, leak before burst, and maximum time allowable before compensation activation.

3.4 FMA classification for the system under consideration

3.4.1 Severity category and consequences

Severity category defined for the system under consideration has been firstly described in [4]. By extending it to the context of this FMEA analysis, table 3.3 has been build.

Table 3.3: Severity category definition for the system

Category	SN*	Effects	Consequences
Catastrophic	4	Loss of system	Accelerator stopped (24h and more), beam pipe obstruction or vacuum leak
Critical	3	Loss of system	Accelerator not stopped, Repair requires vacuum beam pipe opening
Major	2	System stopped	Repair requires access to underground facility
Minor	1	System unstable	Repair requires physical or remote access, precision degradation
Negligable	1	Measure lost	Repetition of the measure

*SN : Severity Number

3.4.2 Occurrence probability

The probability of occurrence is defined in the standard as a number from the lowest probability (1) to the highest (4). Table 3.4 summarize these levels and proposed an interpretation for the system under consideration.

Table 3.4: Probability of occurrence definition for the system

Category	PN*	Limits	Interpretation
Probable	4	$P > 1E-1$	ones every 100 scans
Occasional	3	$1E-3 < P < 1E-1$	between 100 - 10k
Remote	2	$1E-5 < P < 1E-3$	between 10k - 100k
Extremely remote	1	$P \leq 1E-5$	between 100k - 1M

*PN : Probability Number

3.4.3 Detection probability

The detection probability defines the likelihood of discovering a problem.

Table 3.5: Probability of detection definition for the system

Category	DN*	Interpretation
Extremely unlikely	4	No visible consequences on system operation
Unlikely	3	Detectable only by looking at post-mortem data
Likely	2	Detectable during a measure cycle
Very likely	1	Detectable before a measure cycle

*DN : Detection Number

3.5 Failure modes and effects worksheet

This section describes the core of the analysis. Its purpose is to "identify all potential failure modes for each item and investigate their effect on the item under analysis and on the product and operation to be studied."

It assumes that each single item failure is the only one in the product and implies that combination of failures are not considered. The analysis is evaluating each failure mode in terms of the worst potential consequences and assign a severity category.

3.5.1 Worksheet filling recommendation

The following part is the recommendation given in the standard [1] to fill the FMEA worksheet. The FMEA worksheet should contain:

1. **Header information** Identity of the product (hardware or function) and the identity of corresponding equipment, subsystem, and system (as applicable)
2. **Identification number** for traceability purposes.
3. **Item/block** name of the item or function being analysed, and the block of the reliability block diagram that is applicable to the analysis entry.
4. **Function** concise statement of the function performed by the item.
5. **Failure mode** identification and description of all potential failure modes of the item or function under analysis.
NOTE With reference to 4.5d, end effects of lower level FMEA are failure modes of the higher level FMEA.
6. **Failure cause** When requested, the FMEA worksheet shall contain the identification and description of the most probable causes associated with the assumed failure mode.
NOTE 1 With reference to 4.5e failure modes of lower level FMEA are failure causes of the higher level FMEA.
NOTE 2 The failure cause are generally not identified when components are analysed (equipment level FMEA).
7. **Mission phase/Operational mode** concise statement of the mission phase and operational mode in which the failure is assumed to occur.
NOTE These elements can be addressed in the header of the worksheet. Although all of the different mission phases or operational modes are taken into account, the record of results is limited to the phase or mode in which the worst failure effects occur.
8. **Failure effects** identification of the consequences of each assumed failure mode at local effects and end effects levels.
NOTE 1 Local effects concentrate specifically on the impact of the failure mode on the operation, function, or status of the item identified in the second column of the worksheet. The local effects are recorded when different from the failure modes. The purpose of defining local effects is to provide a basis for evaluating compensating provisions and for recommending corrective actions.
NOTE 2 End effects define the effect that the analysed failure mode has on the operation, function, or status of the product under investigation and its interfaces, such that it allows integration into the next higher level FMEA.
9. **Severity classification** severity classification category assigned to each failure mode according to the worst potential end effect of the failure (see clause 4.1 and 4.2).
10. **Failure detection method and Observable symptoms** identify the failure detection method and the observable symptoms.
NOTE The failure detection means include telemetry (exact label), visual or audible warning devices, sensing instrumentation, other unique indications (e.g. the failure effect itself), or none.

11. **Compensating provisions** identify the existing compensating provisions, such as design provisions or operator actions, which circumvent or mitigate the effect of the failure.

NOTE 1 Design provisions: Compensating provisions are considered design provisions when they feature a design that nullifies the effects of a malfunction or failure, control, or deactivate product items to halt generation or propagation of failure effects, or activate backup or standby items. Design compensating provisions include:

- redundant items or alternative modes of operation that allow continued and safe operation, and
- safety or relief devices which allow effective operation or limit the failure effects.

NOTE 2 Operator actions: Compensating provisions are considered operator actions when the operator circumvents or mitigates the effect of the postulated failure mode.

12. **Recommendations** for corrective actions shall be noted. Each recommendation shall have a non ambiguous identifier for tracking purpose.
13. **Remarks** any pertinent remarks relevant to and clarifying any other column in the worksheet line.

3.5.2 Worksheet

Table 3.6: FMA Worksheet sensors, Part A

Product: Beam wire scanner							
Project/Phase: Phase B (Preliminary definition)							
System/Subsystem/Equipment: Actuator sensors							
Id	Item Block	Function	Failure mode	Failure cause	Local effects	End effects	Severity
1	I01	Resolver	FM1 Pos noise	signals too small	Position noise	feedback unstable	Major
2	I01	Resolver	FM2 Pos error	coils shorten**	Position error	Motor torque issue	Major
3	I01	Resolver	FM3 Pos jump	RDC link, FPGA integration	Jump to unknown position	Undesired multiple rotations	Catastrophic
4	I01	Resolver	FM4 Pos lost	coils grounded RDC broken	pos stuck to ?	Loss of feedback ctrl	Catastrophic
5	I02	OPS* 1	FM1 signal low	laser power low / wrong distance Lens-disk	Pulses count errors	position uncertainty	Minor
6	I02	OPS* 1	FM2 signal overflow	laser power too high	Pulses count errors	position uncertainty	Minor
7	I02	OPS* 1	FM3 signal lost	laser or fibre or circulator or disk or photodiode broken	signal stuck	sensor off	Major
8	I03	OPS* 2	FM1 signal low	laser power low / wrong distance Lens-disk	Pulses count errors	position uncertainty	Minor
9	I03	OPS* 2	FM2 signal overflow	laser power too high	Pulses count errors	position uncertainty	Minor
10	I03	OPS* 2	FM3 signal lost	laser or fibre or circulator or disk or photodiode broken	signal stuck	sensor off	Major
11	I04	CT* 1	FM1 signal lost	multiple sources	Phase U curent not available	Loss of current feedback	Critical
12	I05	CT* 2	FM1 signal lost	multiple sources	Phase V curent not available	Loss of current feedback	Critical
13	I06	CT* 3	FM1 signal lost	multiple sources	Phase W curent not available	Loss of current feedback	Critical

*OPS: Optical Position Sensor, *CT: Current transducer, ** as described in [5],

Table 3.7: FMA Worksheet sensors, Part B

Prepared by: Document ref.:								
Approved by: Issue:								
Date:								
Id	Failure detection method/ observable symptoms	Compensation provisions	SN*	PN*	DN*	CN*	Corrective actions	Recommendations and Remarks
1	Secondary coils signals too low	filtering	2	2	2	8	measure sig. amplitude	-
2	Secondary coils signals mismatch	-	2	2	3	12	monitor pos. coherence	-
3	Unphysical new position Dpos/dt	-	4	3	1	12	monitor pos. coherence	-
4	Position stuck to one value	-	4	1	2	8	monitor pos. coherence	-
5	Signals amplitude too low	-	1	3	1	3	switch to CH2	-
6	Signal with long saturations	-	1	3	1	3	switch to CH2	-
7	Signal stuck, noise only	-	2	3	1	6	switch to CH2	-
8	Signals amplitude too low	-	1	3	1	3	switch to CH1	-
9	Signal with long saturations	-	1	3	1	3	switch to CH1	-
10	Signal stuck, noise only	-	2	3	1	6	switch to CH1	-
11	Current balance between phases	-	3	1	3	9	disable sensor U	calc. U with V & W
12	Current balance between phases	-	3	1	3	9	disable sensor V	calc. V with U & W
13	Current balance between phases	-	3	1	3	9	disable sensor W	calc. W with U & V

*SN: Severity Number, *PN: Probability Number, *DN: Detection Number, *CN: Criticality Number

3.6 Fault Tree Analysis

After the identification of the failure modes for each of the items, the fault tree analysis (FTA) is used to identify the end effects due to combination of failures.

3.6.1 Existing system

The first step of this analysis is preformed for the current configuration of the system, i.e. without the combination of sensors. Each sensor is analysed individually since no link exist yet. Fig.3.3 presents the tree for the resolver, Fig.3.5 shows the tree for the optical position sensor and Fig.3.4 for the current transducers.

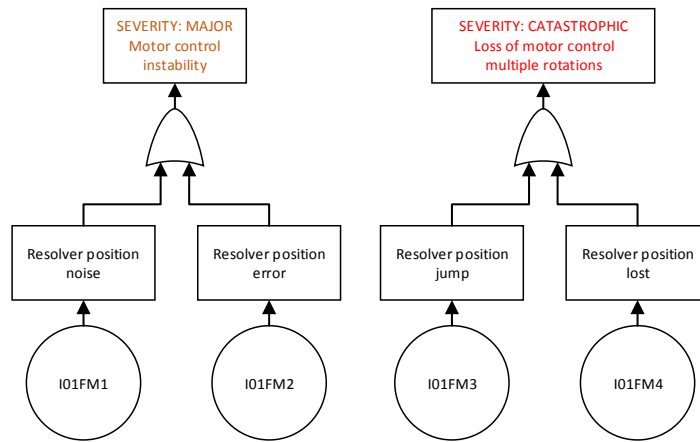


Figure 3.3: Fault tree of the resolver

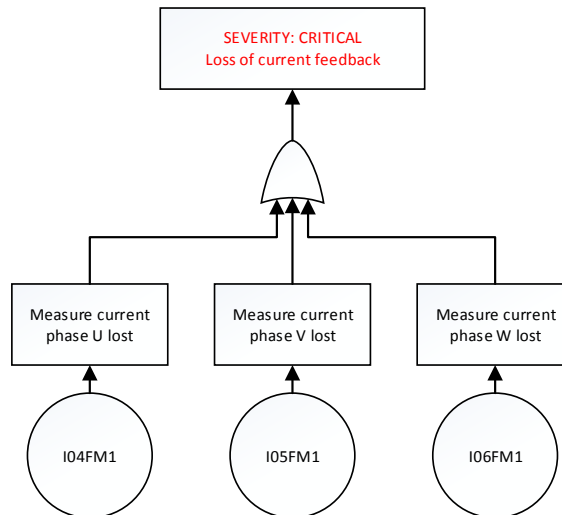


Figure 3.4: Fault tree of the current sensors

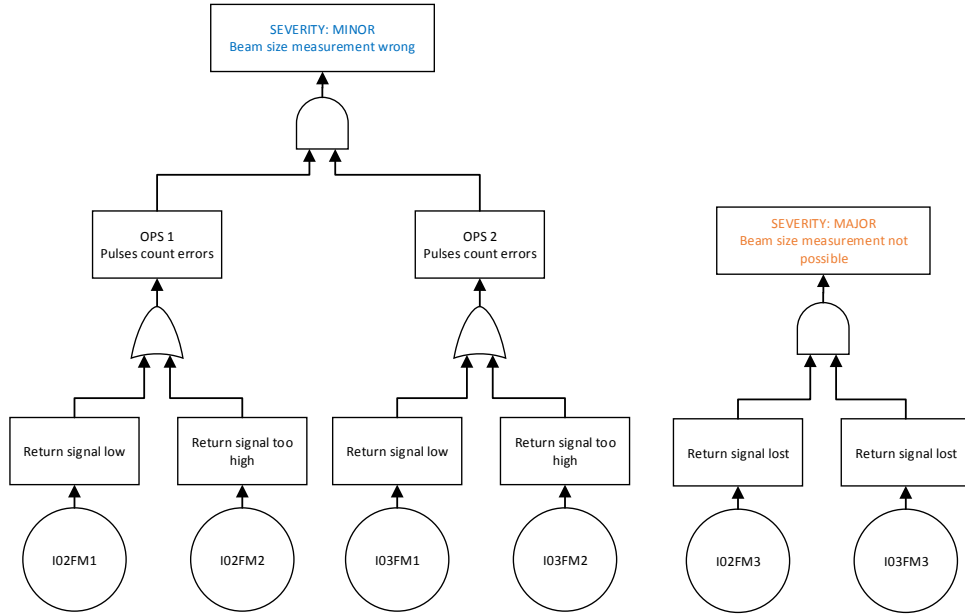


Figure 3.5: Fault trees of the optical position sensor

3.6.2 Future system using sensor combination

Fig.3.6 presents the fault tree analysis for the system after the combination of the sensors. This version suppose that the current control is absolutely needed and if 2 failures occurs at the same time, the system would be lost.

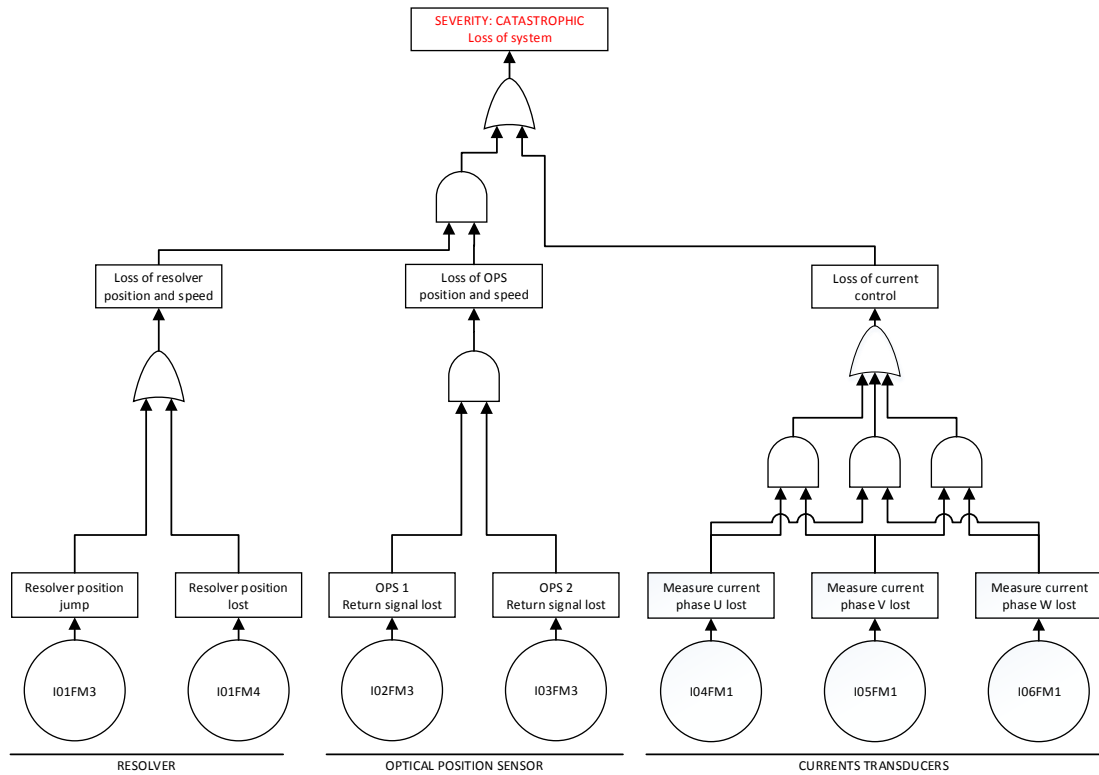


Figure 3.6: Fault tree in case of sensors combination

Fig.3.7 in the other hand suppose that the failure of the current transducers would be compensated and the

system would still be in control. In this latest version, the system would be lost only if the three sensors group would be failing. If implemented, this version would be the most reliable for the system.

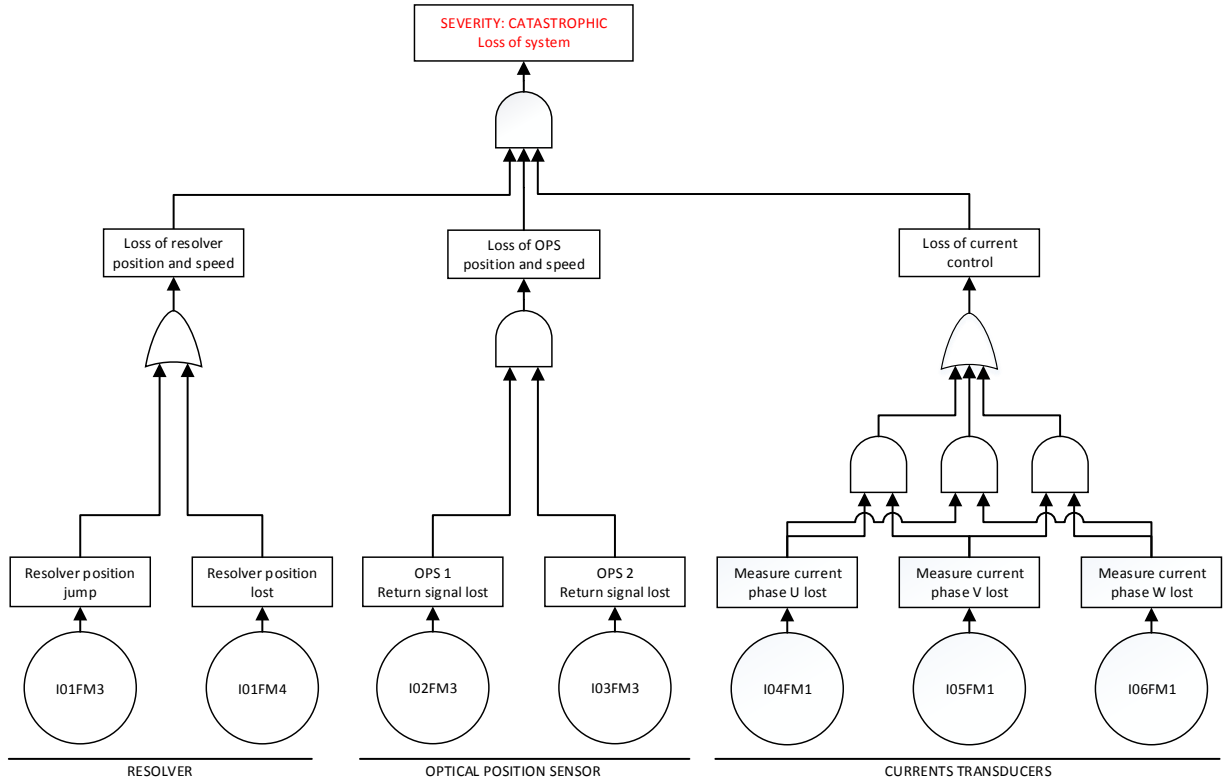


Figure 3.7: Fault tree in case of sensors combination

3.7 Failure detection methods description

The detection method treated in this thesis is data driven. It is proposed to detect system failures by processing the sensors data flow. This approach has the advantage of avoiding any dedicated electronics circuits. This allows fast prototyping and future evolution of the detecting algorithm to new failure case that would not be taken into consideration during the hardware design.

Other possibilities of detection exist. For example, some of the failure cases that are described above can be detected by means of dedicated electronic system. In one hand, the failure of the IOPS laser (reference to item and FM) can be detected by means of current sensing of the monitoring photodiode encapsulated with it. But on the other hand, this requires a dedicated circuit to be tuned and maintain, while a data processing approach does not require dedicated hardware and any change in that system is more practical to adapt.

3.7.1 Resolver

The resolver sensor is processed by an external ASIC called RDC which provides an digital estimation of the position.

The first failure detection method is to monitor the raw signal transient coming from the ASIC. It is expected that the delivered position is not able to make a large jump, with the exception of a complete rotation which is not considered in this work. The proposed implementation is the use of a median filter, in fact this type of filter has the ability to reject extreme values of a vector. This vector is constructed using a First In, First out (FIFO) of a certain depth. By making the difference between the input and output of this filter, it is possible to generate one value equal to the jumps that are filtered out. By comparing this value against a threshold, a status bit is generated.

The second detection method proposes to monitor the absolute value of the position which should stay in a defined zone, this can be done by comparing the incoming position against a threshold.

The third method is based on the derivative of the position, which is the system speed. As this physical quantity has to be in a defined range, it is possible to monitor its absolute value and generate a status using a threshold. This method differs from the first methods which uses a median filter. They are both computing the derivative in a sense, to the difference that the speed is the result of a smoothed derivation. So here we look at a slower process.

In parallel to the RDC, an independent digitalisation of the raw resolver signals is planned, but not implemented at the time of this thesis. This new hardware functionality has the potential of detecting the signal level coming from the resolver directly, allowing the detection before any processing. This would allow to detect early potential problem, by monitoring the trend of the signals, by generating the peak-to-peak values for example.

This sensor and the detection is discussed in details in Chap.7.

3.7.2 IOPS

The Incremental Optical Position Sensor (IOPS) is an sensor having two independent channels. Each channel gives an analog signal proportional to the reflected light by moving small mirrors. This signal is digitalised by an 14 [bit] Analog-to-Digital Converter (ADC). In normal case the signal provides an almost sinusoidal signal of variable period. Each period correspond to a precise angular displacement of the shaft.

The first failure detection method is to monitor the amplitude of this sinusoidal signal. When the system is operated, a variation of this analog signal must be seen. when the actuation system is not powered, the signal is stable. Due to the different mode of operation of the system, this method is dependant of external status and can't be only data driven.

A second approach is to compute the position based on the periodic signal and to compare it with the other sources. In this case the processing is independent and can run by itself.

This sensor and the detection is discussed in details in Chap.8.

3.7.3 Current transducer

The current transducer are used to measure the three motor phases. Each of the sensor are producing an analog voltage digitalised with an 16 [bit] resolution ADC at 100kHz.

The first detection method is similar to the previous sensors, it uses an input median filter. The difference between the input and output is monitored using an threshold comparison. If this difference is too high, this means there is a possible issue with the acquisition chain.

The second monitoring is looking at the physical quantity measured by this sensor. As we know in advance what is the maximum possible value, we can set a threshold to verify that this level is never reach.

The third method proposes to detect eventual failure by combining the information of the three currents. These three currents are connected to an PMSM type motor where all the 3 phases are connected together in a star topology. This means that the sum of the currents should be close to zero. By monitoring the result of the sum of these three current, it is possible to verify the coherence of the measure, to the limit of the acquisition chain precision and noise.

This sensor and the detection is discussed in details in Chap.9.

3.7.4 Sensor fusion

The fusion of the sensors measures is a very powerful method to detect failure of sensor or its acquisition chain. If two sensors are providing the same physical quantity, directly or indirectly, it is possible to compare this quantity and detect failure. The sensor fusion has also the potential to overcome one or multiple sensors failure.

By selecting one functioning source instead of a corrupted one, it can provide more robust output. A last possibility is that the sensor fusion is providing an output of better 'quality', meaning that this technique has the potential of providing a better output than the input sources taken individually.

This is a large subject by itself and one possible implementation is proposed in this thesis to detect certain failures and at the same time to provide an robust output to further processing. This output has the potential of providing an output of better quality as well.

3.8 Conclusion of the failure mode analysis

The Failure Mode and Effects Analysis (FMEA) method performed in this chapter has identified possible functional failure modes of the actuation system of the BWS. The method is based on the document "Space product assurance - Failure modes, effects (and criticality) analysis (FMEA/FMECA)" (ECSS-Q-ST-30-02C [1]).

The outcome is a categorisation of the possible functional failures of the actuation system. Single failure are described in Failure Mode and Effects Analysis (FMEA) worksheet in section 3.5 and the combination of them is investigated using the Fault Tree Analysis (FTA) in section 3.6.

The last part in section 3.7 is the description of the detection method which will be implemented in this thesis.

By the use of this methodology, functional failure mode of the BWS have been identified and ways of mitigating them have been proposed.

Chapter 4

Methodology and design tools chain

This chapter introduces to the methodology selected for this thesis. It allows the design, development and verification. It has to be efficient to design various algorithms in the limited time of this thesis while powerful enough to verify complex algorithms. The target processing uses Hardware Description Language (HDL), VHDL for instance, which makes prototyping code uneasy, so a second language was selected for this purpose. It was also used for the verification of high level entity and for result presentation.

The design flow of this thesis uses theses majors steps:

1. Algorithms design and prototyping using a scripting tools such as MatLab or equivalent.
2. Translation of the algorithms into the target language (VHDL).
3. Register Transfer Level verification of small entity (components) using (VHDL) test bench
4. High level verification of large processing entity using comparison between MatLab-VHDL outputs
5. Final code synthesis (compilation for the target FPGA) and optimisation.

The development of the first version of the control system for this wire-scanner has similarly used comparison of multiple design steps resulting by higher confidence in the developed processing [6]. Such systematic approach allows fast prototyping with quick and flexible scripting or visual language and a way to verify the final implementation.

Fig.4.1 shows an detailed overview of the design tools chain that was used during this work. From existing real world data sets, algorithms are developed using the MatLab environment [7]. These input data are also translated into text file to be read by the verification tool called ModelSim [8].

The VHDL version of the algorithms are written inside a text editor and then compiled by ModelSim. Then test benches are written around the developed entities to be fed by either artificially generated or file based stimuli. The output is saved into files and later compared with MatLab version of the same processing code. Once the VHDL code has been verified, the vendor specific tool, Quartus II [9] is used to synthesis the hardware description for the specific target and provides statistics about the resources utilisation (i.e. number of elements used for the specific code, estimated maximum execution speed, etc.).

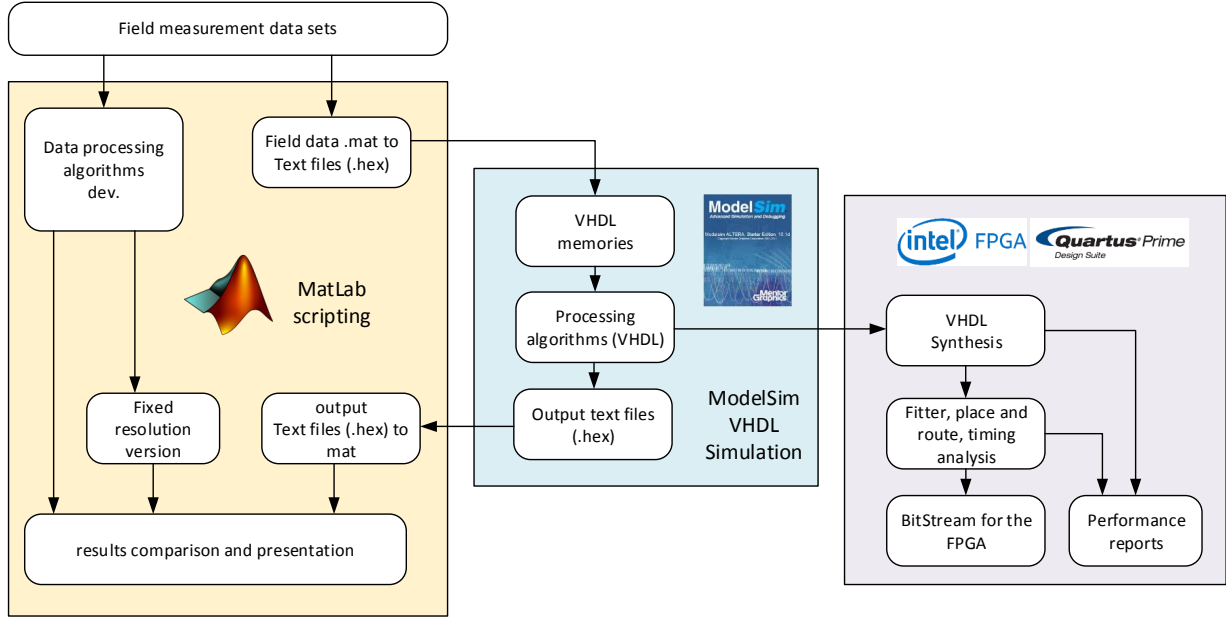


Figure 4.1: Tool chain structure for this thesis

Each of the sensors treated in this thesis have a dedicated chapter where these design steps are developed along after a detailed introduction on the physical sensor itself. A last chapter presents the combination of the sensors using the same systematic approach.

4.1 Scripting tool for algorithms prototyping

The selected tool to perform algorithms prototyping is MatLab [7]. It is widely used for this purpose and has a lot of extension packages. The particularity of this tool is the availability of powerful and optimised matrix and vector manipulations which allows quick coding of complex algorithms. In the case of this thesis, the goal of the prototyping is to find ways to process the data efficiently in real time, so most of the complex functionality have to be avoided, to fit as close as possible to the hardware capability.

4.2 Implementation method

Once the algorithms have been prototyped with MatLab, they need to be translated into Hardware Description Language (HDL) to be synthesised for a target FPGA. Some options are today existing to perform this translation:

- (a) Manual code transcription
- (b) MatLab/Simulink automatic generation
- (c) Automatic generation tools, such as Intel HLS Compiler, Open CL

The first method is the manual transcription of the prototype code into the target language. This method relies on the ability of the designer to accurately describe the algorithm using the specificity of the VHDL. It results for the designer a deep knowledge of the implementation, allowing a complete control of the code. It also avoids additional software tools, increasing the maintenance complexity. The downside is probably the investment in time to completely rewrite the code from one language to another very different, with its risks of errors. For each modification of the algorithm, two versions have to be modified and compared.

For this thesis, the manual transcription is going to be used, but alternatives exist which are using automated tools to perform this task.

MatLab/Simulink has its own tool to perform the generation of VHDL code using a module called HDL Coder™ [10] which generates Verilog or VHDL for Xilinx or Intel FPGA and ASIC design. This tool converts automatically the MatLab code written in floating point into fixed point version and generates implementable VHDL or Verilog. It is also possible to generate code written in the visual language Simulink. One of the advantages to use this option is that the link between the prototype code and the implemented one is kept, which simplifies the compliance to standards such as DO-254 [11] [12].

An alternative to commercial software is the initiative of the REDS institute of the HEIG-VD which proposed a code generator called Math2mat [13].

"The aim of the Math2mat software is to automatically generate a VHDL description of a mathematical function using floating point operators. It also generates a full SystemVerilog testbench that allows to validate the hardware description by comparing it to the initial mathematical description. The initial description corresponds to the Octave syntax (compatible with Matlab), and allows operations such as addition, subtraction, multiplication, division, and structures such as if/then/else and for loops."

The main features of Math2mat are:

1. Editor for typing Octave code
2. Dynamic code analysis, during typing
3. Generation of synthesizable VHDL code
4. Floating point operations
5. SystemVerilog testbench generation
6. Fully automated test suite for the generated code
7. Graphical interface with internal structure viewing

Math2mat provides a graphical interface as shown in the figure below. Command line operations using a subset of the Java classes are also possible, and are documented in the developer guide [14]

Other initiatives are existing in this domain such as "Programming an FPGA-based Super Computer Using a C-to-VHDL Compiler: DIME-C" [15]

4.3 VHDL development and verification

The industrial standard tool for simulation of VHDL code is called ModelSim from Mentor [8]. It comes with free and paid distribution of the vendor specific tools for FPGA designs.

This tool is used to compile the VHDL code written in the text editor, providing help messages in case of syntax or logical errors. Once fixed, the tool saves the entities compiles into the 'work' library for later use.

the goal of code verification is to ensure that the developed code is behaving as expected. this behaviour should be verified against the specification, this for normal cases as well as for corner cases and unexpected circumstances.

The test benches are the verification code, also written in VHDL which are wrapping the code to be verified. It allows stimulation from various sources and the verification of what the Device Under Verification (DUV) outputs, i.e. if it does correctly the defined functionality. These test benches don't have to be 'synthesized' (able to fit into a FPGA), which allows the use of more powerful instructions.

In this thesis, the verification of the small blocs are preformed using VHDL test benches and are providing error messages in case of unwanted behaviours, i.e. 'assert' instruction.

Fig.4.2 presents the architecture of the test benches for the VHDL code. We can see at the centre the Device Under Verification (DUV) encapsulated inside an entity which stimulate its inputs and check its output. The use of the library Transaction Level Modeling VHDL Methodology (TLMVM) [16] allows a step further in the verification by introducing the Transaction Level Modeling (TLM) which abstracts low level signal controls to focus on higher level parts of the verification, the transactions. This concept was used for instance in the verification of the convolution core developed during this thesis.

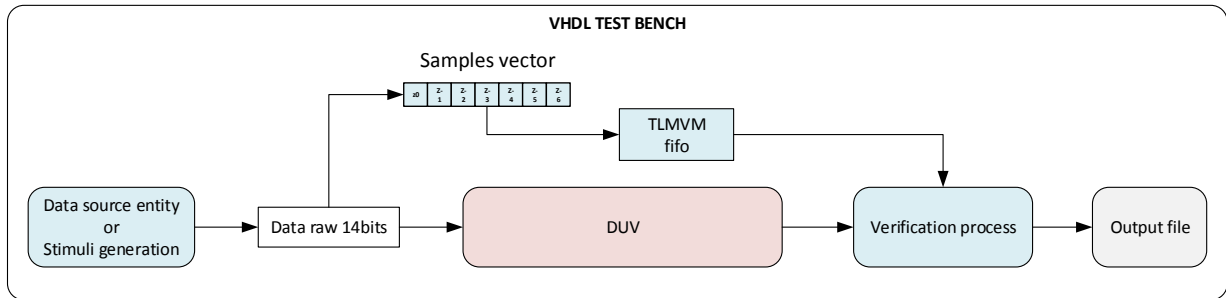


Figure 4.2: Test bench structure for the verification of individual modules

When dealing with complete data processing units made of multiple modules, the test benches complexity becomes difficult to manage for small project. The selected alternative for this thesis uses file based verification. The stimuli are generated from text file containing real measurements (that are also used with MatLab) and the outputs of the DUV are recorded inside text file and later compared with the prototypes algorithms outputs.

This method requires to translate the existing data set (field data) into readable file using the VHDL language. This method has the advantage to use a simpler verification setup, limiting it to one software application. As long as the data set is not too large to fit into a text file, this method provides an optimal setup.

The counterpart of using a predefined file for the input vector is that the sequence is defined before the simulation. The code to verify can't interact with its inputs generation, i.e. it is not possible to simulate a control system which directly has an action to the input vector through feedbacks.

To integrate these files into the simulation, one entity has been created. This component loads the data contained in the files into its memories at the execution through the use of 'generics'. This approach simplifies the use of multiple data set for testing the same entity, i.e. there is no needs to recompile the DUV and the test bench. The component uses specific types for its sensors outputs. These types were developed during this work and are described in the "data flow control and standard interface between components" in section 12.1.1.

The source data raw component is defined as

```

entity source_data_raw is
  generic(
    IOPS_1_DATA      : STRING := "data_in_scan_ops_1_07_Jan_2016_15_43_32.hex";
    IOPS_2_DATA      : STRING := "data_in_scan_ops_2_07_Jan_2016_15_43_32.hex";
    IOPS_DEPTH       : integer := 2**20-1;
    RESOLVER_DATA    : STRING := "data_scan_07_Jan_2016_15_43_00-pos-res-16k129Hz.hex";
    POSITION_DATA     : STRING := "data_scan_07_Jan_2016_15_43_00-pos-ref-16k129Hz.hex";
    POS_DEPTH        : integer := 10000;
    CURR_U_DATA      : STRING := "data_scan_07_Jan_2016_15_43_00-current-U-100kHz.hex";
    CURR_V_DATA      : STRING := "data_scan_07_Jan_2016_15_43_00-current-V-100kHz.hex";
    CURR_W_DATA      : STRING := "data_scan_07_Jan_2016_15_43_00-current-W-100kHz.hex";
    CURR_DEPTH       : integer := 62000;
    START_OFFSET     : natural := 0);
  port(
    clk_i              : in      std_logic; -- update OPS data at 20MHz, 100KHz and 16.129KHz
    rst_i              : in      std_logic;
    update_output_i    : in      std_logic;

    pt_data_iops_o     : out     std_logic_vector(19 downto 0); -- 1048576 positions
    channel_iops_1_o   : out     sensor_raw_type;
    channel_iops_2_o   : out     sensor_raw_type;

    pt_data_res_o      : out     std_logic_vector(13 downto 0); -- 16384 positions
    pos_reference_o    : out     sensor_raw_type;
    pos_resolver_o     : out     sensor_raw_type;

    pt_data_cur_o      : out     std_logic_vector(15 downto 0); -- 65536 positions
    current_u_o        : out     current_sensor_type;
    current_v_o        : out     current_sensor_type;
    current_w_o        : out     current_sensor_type
  );
end source_data_raw;

```

As the data rate are different for each type of sensors, three pointers are defined. One for the IOPS, one for the resolver and one for the currents. The data are output at the input clock speed (20 MHz) for the IOPS, at the input clock divided by 1240 for the resolver and at the input clock divided by 200 for the currents.

The module view of this entity is

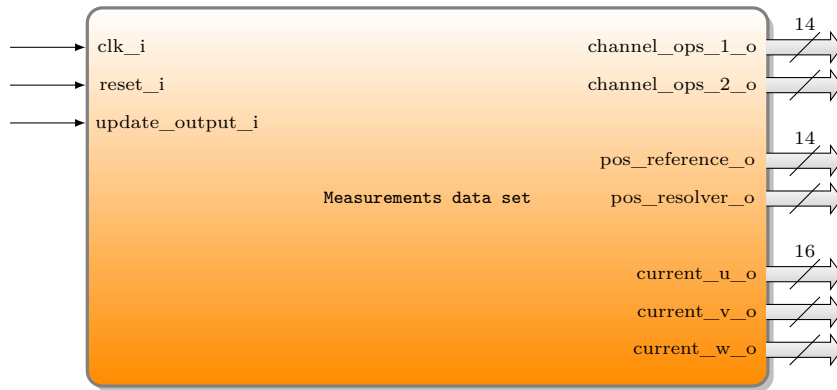


Figure 4.3: Field measurement player for the VHDL simulation

This module can easily be integrated into any test bench where the DUV needs to be stimulated with real world data sets, such as the processing units developed in this work.

Fig.4.4 presents the resulting data output of this module when running in the simulation software ModelSim. The outputs of the IOPS are the two first lines, the third is the reference position, the fourth is the resolver measure of the shaft angle. The three last lines are representing the three phases current names u, v and w.

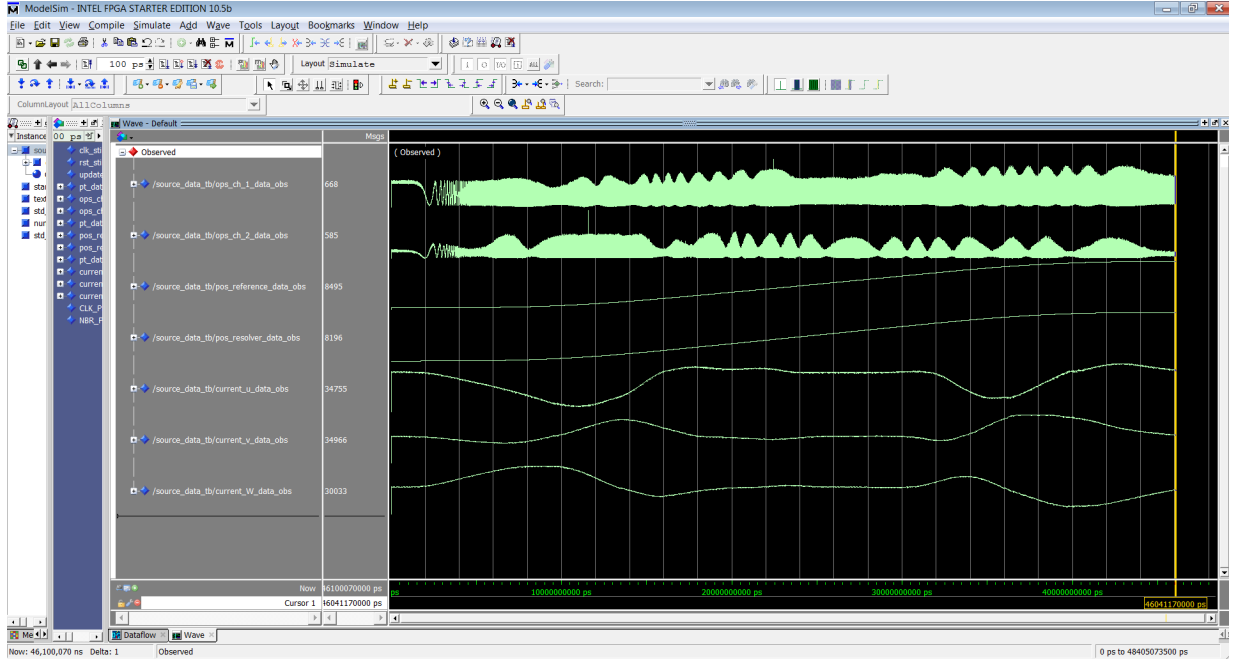


Figure 4.4: Integration of field measurement of the IOPS, resolver and motor currents into the RTL simulation

4.3.1 Verification alternative: Cosimulation Modelsim - MatLab

The verification setup should be able to read the data set of field measurements. These data have already been used to develop the algorithms prototypes using MatLab in Chap.6. It is a natural step to investigate the possibility to use again MatLab or Simulink as vectors generator. This is possible through the use of co-simulation. Two software are working in parallel and are exchanging simulation information. In our case this would be Simulink for the stimulation and Modelsim for the VHDL simulation.

Another major advantage of such method is that this link enables digital feedback system functional behaviour using complex MatLab or Simulink models, and this without the need to translate models in VHDL or SystemVerilog or other modelling language.

A previous beam wire-scanner project has been already using Co-simulation. Between Simulink and Modelsim. This setup has allowed the functional behaviour check of a position-speed-current controller written in VHDL using the Simulink modelling of the scanner actuator [6]. Fig.4.5 is extracted from this reference and presents this co-simulation setup where the motor model is replicated in Simulink. Each copy is controlled with a different implementation of the algorithm, one in Simulink, and the other in VHDL by using ModelSim. In this case it is an implementation of a vector controller for motor feedback.

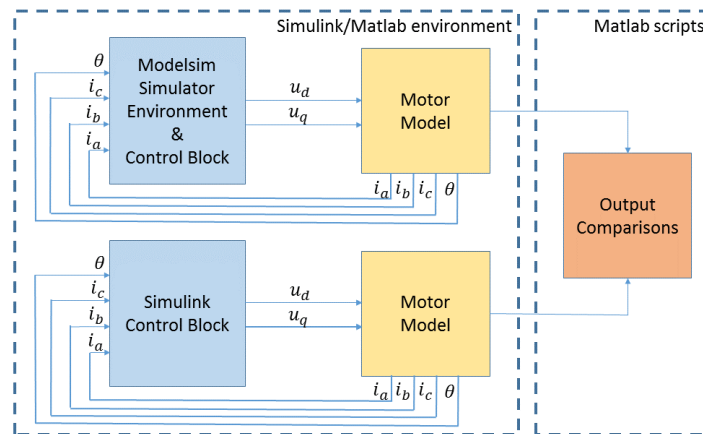


Figure 4.5: Co-Simulation setup for the functional verification of feedback systems [6]

This powerful method has not been used in this thesis, as the developed code did not have any interaction with the real system, therefore there was no need to model the system, and a simpler solution could be used to integrate recorded real world data .

4.4 Tool for synthesis and target implementation

The implementation of these algorithms will take place inside a Field-Programmable Gate Array (FPGA). This type of processing unit is widely used in the CERN Beam Instrumentation group (BE-BI) due to its massively parallel processing, high performance and versatility.

The FPGA used for the target board is based on the 5th generation of Altera device, and a mid-range device called Arria. It features 176k Adaptive Logic Module (ALM) having eight inputs and eight outputs each, 6.5 Gbps transceivers and 23Mbit of internal memory. It also features 1k of Digital Signal Processing (DSP) blocks for arithmetic operations.

The company making this FPGA is called Altera, and has just been bought by Intel. The tool to develop code for these devices is called Quartus 2 [9]. This tool has the ability to take VHDL code and synthesis hardware dedicated for the target under consideration. At this stage the necessary resources for a particular function is known.

The next steps of the design compilation are the assignment of specific chip location to particular function, it is called the place, route and fitting phase. The last step is the assembly of these information into one file to program the device.

One additional and very useful step is the estimation of the maximum frequency at which the FPGA will be able to be operated. This parameter is dependent on the way the processing is described in VHDL. For example, an extremely large combinational function can slow down the whole device due to the propagation time between numerous gates until the next flip-flop. For this reason, long processing needs to be pipelined, i.e. to insert additional registers to introduced in to out delay, but increasing the throughput capability of the device having a faster operating frequency.

For instance, the first version of the convolution core was using multiplication and summation in one go, which slow down the fmax to 30MHz. Changing the architecture of this core by inserting pipeline improved the maximum frequency up to 70MHz, above the target frequency of 50 MHz. See subsection 12.1.2 for details.

4.5 Numerical representation on the target

The number representation is a key characteristic of the implementation that will define the resolution of calculated outputs and the ease of the development. Three type of representation is envisaged for the implementation of the algorithms.

The first one is the straight binary format, without any complication. this has the advantage to very simple and low resources are needed for its implementation. The major limitation is the risk of loss of precision and overflow in the calculation.

Secondly, the floating point representation, such as the Institute of Electrical and Electronics Engineers (IEEE) standard for Floating-Point Arithmetic (IEEE 754) [17], has the advantage of being very robust for complex processing. This representation is able to handle very small to very large numbers with a range adaptation, called mantis. With the disadvantage of variable precision depending on the range of the number. The resources needed for this type of representation is large, but it also ease the development.

And finally the fixed point representation, on the other side, has a constant precision over the full range of the number, allowing a much lighter implementation. The downside is the risk of overflow with a this type of representation. By carefully planning the processing with sufficient precision, i.e. number of bits, this solution is the optimal one. It will be used in this thesis.

The input data representation and the ranges coming from the sensors is summarised in Table 4.1.

Sensor	Resolution (bits)	Representation	Update frequency	Position resolution
Resolver	analog	-	-	291 uRad
Resolver + RDC	14	straight binary	16 kHz	378 uRad
Resolver + ADC	14	straight binary	20 MHz	tbd
Optical encoder	14	straight binary	20 MHz	214 uRad / mirroir pitch
Motor currents	16	2's complement	16 kHz / 200 kHz	tbd

Table 4.1: Sensors output characteristics

The output specification to provide the position-speed information must have at least the resolution and precision of the output from the resolver which is used at the moment.

The specification for the position and the velocity is: resolution precision better than: ≤ 291 uRad.

Destination	Res. (bits)	Representation	Update freq.	Position resolution
Motion control	min. 14	fixed point(16,16)	16 kHz	min. 378 uRad
Optical encoder coarse	14.8	fixed point(16,16)	16 kHz	214 uRad (mirroir pitch)
Optical encoder fine	min. 18.2	straight binary	40 MHz	min. 21.4 uRad (mirroir pitch / 10)

Table 4.2: Algorithms output requirement

4.6 Fixed point library

The use of the fixed point representation is simplified by the use of a dedicated library which overload the mathematical operators to allow fixed-point operations as with other types.

The VHDL 2008 version includes fixed-point as well as floating point. Unfortunately, not the whole tool chain supports already this feature. For instance, Modelsim is able to compile and simulate VHDL 2008. The first components of this work were made in 2008 version, but the Intel tools are not integrating this feature for the target FPGA. The Arria V is only supported by the standard Quartus edition while the floating point and fixed point library is only available with Quartus pro edition, which only support Arria 10 [18].

The solution is to use an open source library commonly called ieee-proposed, written in VHDL-2002 by David Bishop. This library introduces a new type for fixed manipulation: `ufixed` and `sfixed`. This library is still available for downloading [19] [20].

To monitoring eventual overflow during mathematical operation with this type, one method was investigated. At each fixed point operation, the resulting number size becomes larger to be able to represent the result. To limit the representation of the numbers, after each operation, an instruction (`resize`) is available in the library to resize numbers to determined sizes. By default the overflow method is set to saturate. This means that a large number resized to with a smaller representation will result to an small number with its value set to the maximum/minimum possible. This makes the detection easy for overflow, the resized number should be verified after each operation. If this number is equal to the maximum or to the minimum of the defined range, there is an overflow and further processing can be notified with a flag.

To experiment this method, an entity has been developed which sums two `sfixed`. If an overflow occurs, as defined above, the output will rise the flag:

```
sfixed_overflow : std_logic;    -- Overflow during a calculation somewhere in this channel with sfixed number.
```


Fig.4.6 presents the verification of this module within the verification software ModelSim. Different vectors are applied to the DUV and some of these operation are raising the flag 'sfixed-overflow'. Its indicates that the representation of the output is not enough for correctly representing the mathematical operation with these numbers, i.e. the output is saturated and wrong.

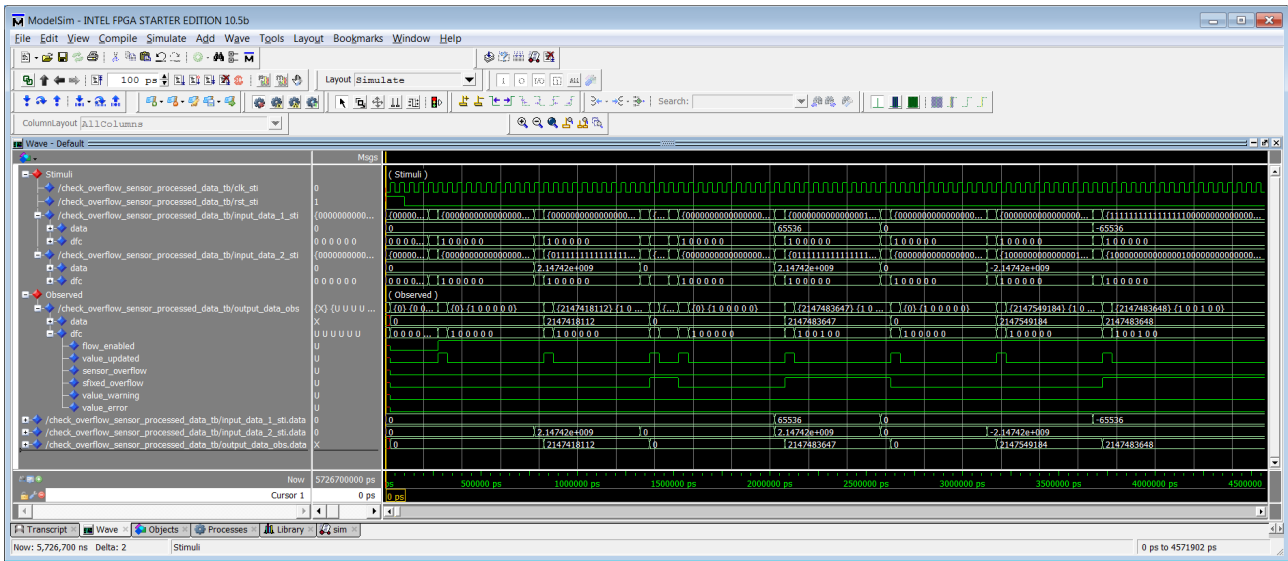


Figure 4.6: Verification of the entity generating the overflow signal

Chapter 5

Available field data sets

The first version of the Beam Wire Scanner (BWS) FPGA control system has produced numerous measurements which are available for the development of the processing algorithms during this work. These measurements were gathered into a list of twelve dates where tests were performed in the Super Proton Synchrotron (SPS) accelerator Fig.5.1. There are two types of files. The first one is a MatLab structure which contains 54 variables recorded at 16kHz, the feedback refresh frequency and thirteen variables at a speed of 100kHz, the ADC digitalisation period. The second type of data is also a MatLab structure which contains the information of the Incremental Optical Position Sensor (IOPS) signal recorded at 20kHz.

In addition to these sets of measurement, artificial failures were created by corrupting one of these data sets to simulate system failure. This allows the verification for unrecorded failure modes.

Table 5.1: List of the available fields measurements

Set Nbr	Date	Scanner type	Nbr Motion data	Nbr OPS data	OPS data format
S1	07 Dec 2015	SPS	29	-	-
S2	08 Dec 2015	SPS	37	-	-
S3	12 Dec 2015	SPS	43	15	float(4) - raw adc(11)
S4	07 Jan 2016	SPS	14	12	raw ADC bits
S5	07 Sep 2016	SPS	17	9	float
S6	09 Sep 2016	SPS	7	1	float
S7	14 Sep 2016	SPS	4	4	float
S8	25 Nov 2016	SPS	37	8	float
S9	02 Dec 2016	SPS	32	3	float
S10	07 Dec 2016	SPS	59	0	
S11	09 Dec 2016	SPS	20	0	

5.1 Example of data sets

The following three figures are showing one example of the available data sets for the development of the algorithms. They are used in the MatLab environment as well as in the VHDL domain. The nexts figure are representing the available data for the three type of sensors.

Fig.5.1 represents the resolver signal and the reference position (top plot). some statistics are presented in the three middle plot with, from left to right, the histogram of the resolver values, the linear fit of one part of the measure which is used to compute the distribution of the noise (middle right plot). The bottom plot presents the reference speed and the speed obtained by derivation of the position using a convolution with Savitzky-Golay coefficients, later described in section 12.1.2.

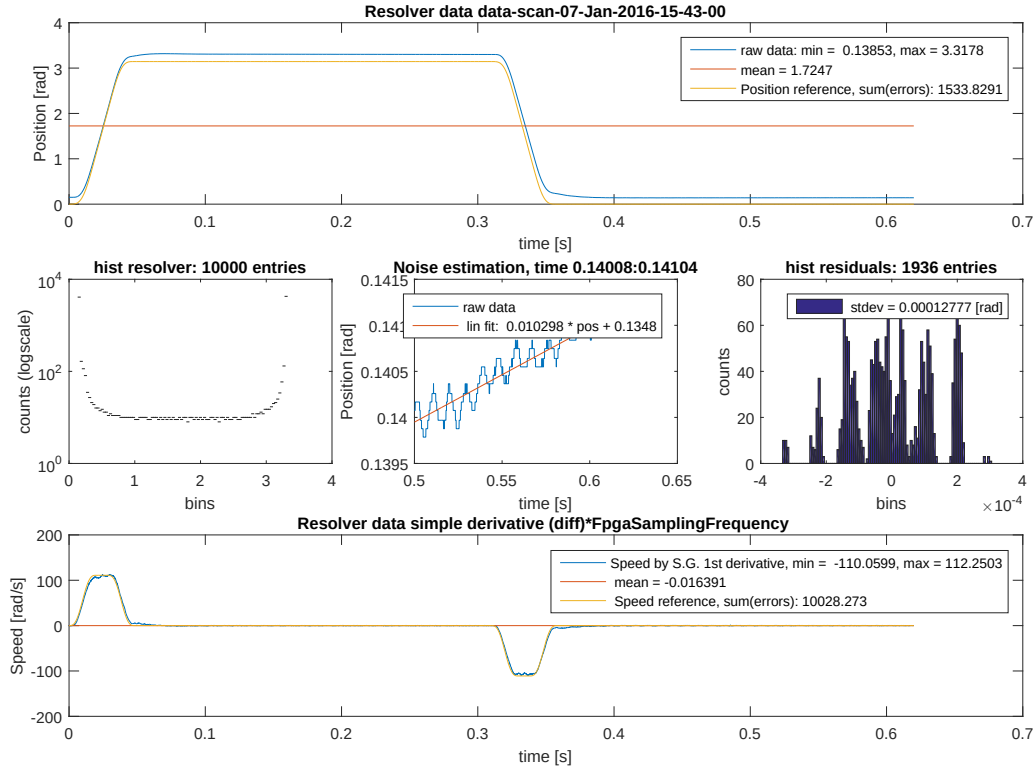


Figure 5.1: Resolver data from field measurement

Fig.5.2 is the plots of the IOPS data of one channel. The top plot represent the raw signal coming from the fast ADC. The bottom plots, from left to rights are a zoom at the beginning of the data vector, then a zoom at the middle of the data vector. The third plot is the histogram of the whole vector while the last plot is the histogram of the beginning of the vector where the system isn't moving, giving a good idea of the stochastic noise.

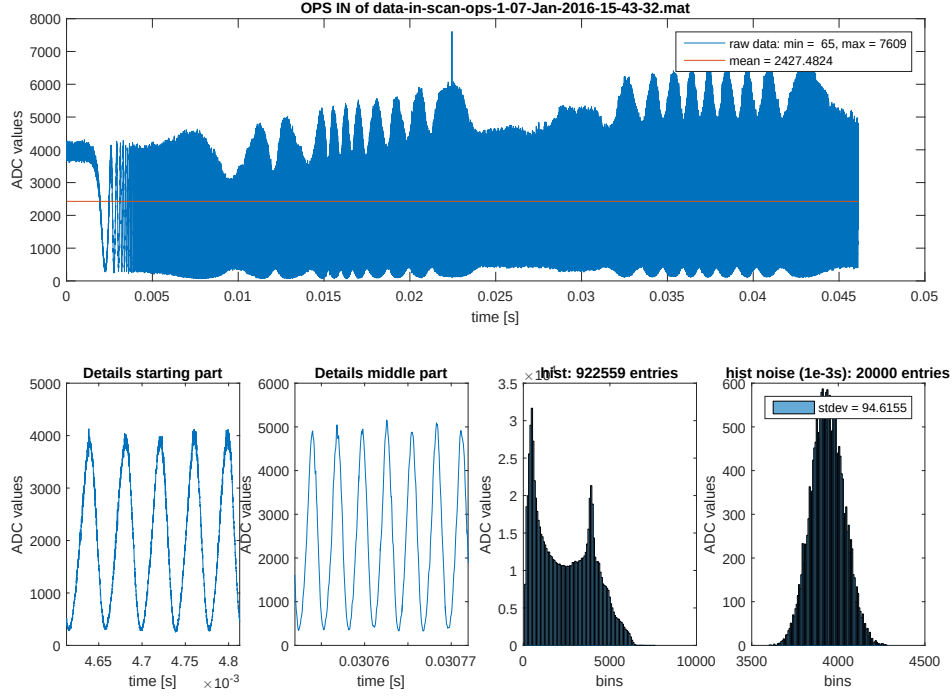


Figure 5.2: Incremental Optical Position Sensor (IOPS) data example

Fig. 5.3 represents the motor phases current. The top plot is a view of all the three currents during a complete measure cycle. We can clearly see two displacement of the system while the rest of the time, the currents are close to zero. The middle plots are the linear fit for each of the current to estimate the stochastic noise which is represented with an histogram for each phase on the bottom plots.

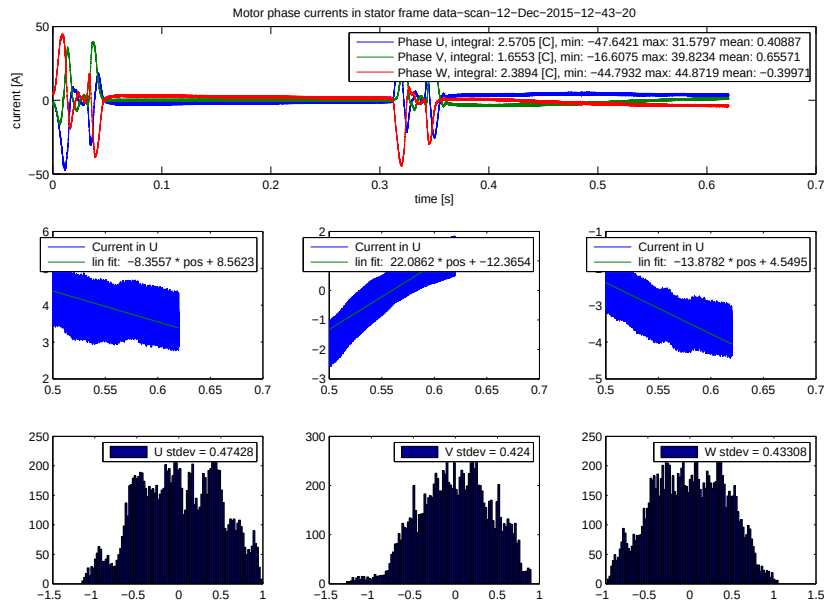


Figure 5.3: Motor phase currents data example

For each of the data sets, visualisation and statistic scripts were developed, details can be found under each

subsection dedicated to a particular sensor.

Out of all these measurement few measures were selected to be used as inputs to develop the algorithms. Table.8.3 list these data and details eventual specificity of each.

Table 5.2: List of selected fields measurements

Set Nbr	Date	time	Specificity
S3	12 Dec 2015	15:08	abnormal system friction introducing position error and large OPS parasitic modulation
S4	07 Jan 2016	15:43	Systems ok
S9	02 Dec 2016	14:49-15:50	OPS 2 data low quality

5.2 Failure cases artificially generated

Since the data available is not covering all cases that needs to be taken into account in this development, there is need of creating new sets of data to cover failure cases described in Chap.3.

The failure cases covered using signal generation are:

Table 5.3: List of sensors failures covered and their associated simulation files

FMA number	description	Simulation file	Figure
I01-FM1	resolver noise	data-scan-07-Jan-2016-15-43-00-noisy.mat	Fig.5.4
I01-FM3	resolver short or long jump	data-scan-07-Jan-2016-15-43-00-short-jumps.mat	Fig.5.5, Fig.5.6
I01-FM4	resolver stuck	data-scan-07-Jan-2016-15-43-00-short-jumps.mat	Fig.5.6
I02-FM1	OPS 1 signal low	failure-signal-low-data-in-scan-ops-1-07-Jan-2016-15-43-32.mat	Fig.5.7
I03-FM1	OPS 2 signal low	failure-signal-low-data-in-scan-ops-2-07-Jan-2016-15-43-32.mat	Fig.5.8
I02-FM3	OPS 1 signal overflow	failure-signal-too-high-data-in-scan-ops-1-07-Jan-2016-15-43-32.mat	Fig.5.7
I03-FM3	OPS 2 signal overflow	failure-signal-too-high-data-in-scan-ops-2-07-Jan-2016-15-43-32.mat	Fig.5.8
I02-FM4	OPS 1 signal lost	failure-signal-lost-data-in-scan-ops-1-07-Jan-2016-15-43-32.mat	Fig.5.7
I03-FM4	OPS 2 signal lost	failure-signal-lost-data-in-scan-ops-2-07-Jan-2016-15-43-32.mat	Fig.5.8
I04-FM1	CT1 signal lost		
I05-FM1	CT2 signal lost		
I06-FM1	CT3 signal lost		

The following figures are representing the sensors data which are artificially deteriorate to cover the failure case. Fig.5.4 shows the resolver data with additional normally distributed noise, which starts after the begging of the system displacement.

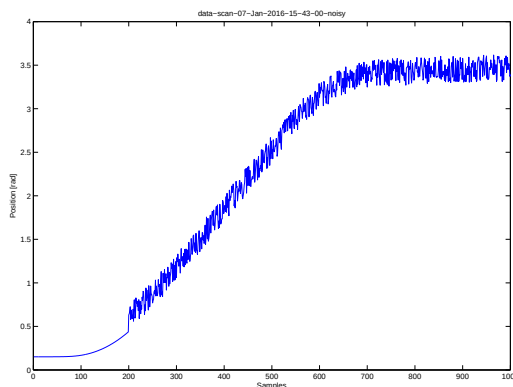


Figure 5.4: Resolver data from field measurement with additional noise apperaring after the start of the movement

Fig.5.5 shows the initial resolver data which is perturbed with artificially added one sample saturation to the minimum or the maximum possible value.

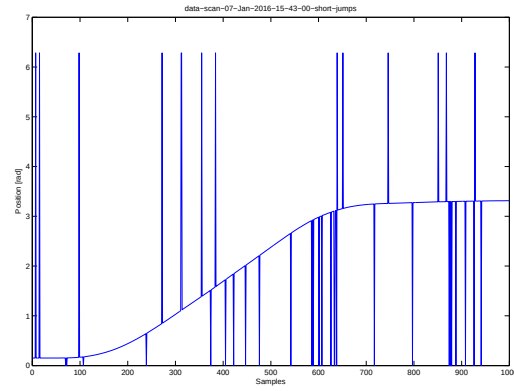


Figure 5.5: Resolver data from field measurement with short jumps artificially added (salt and pepper)

Fig. 5.6 shows an artificially perturbed resolver data with temporary unavailability of the measure for multiple samples, going to the minimum or to the maximum possible value.

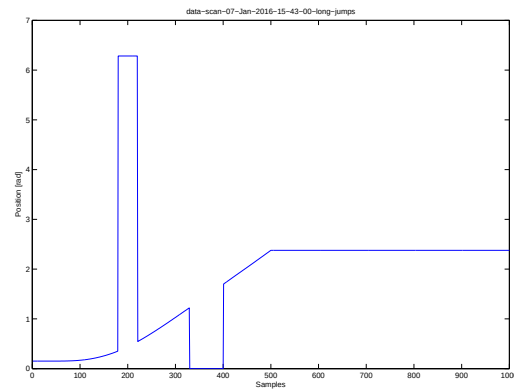


Figure 5.6: Resolver data from field measurement with long jumps and complete failure artificially added

The IOPS sensor has also been artificially perturbed with three different type of failures. Fig. 5.7 is the channel one, Fig. 5.8 is the channel 2. Both figures have an artificial progressive attenuation of the signal (top), progressive raise (middle) and loss of signal with its value stuck to the last value (bottom plot).

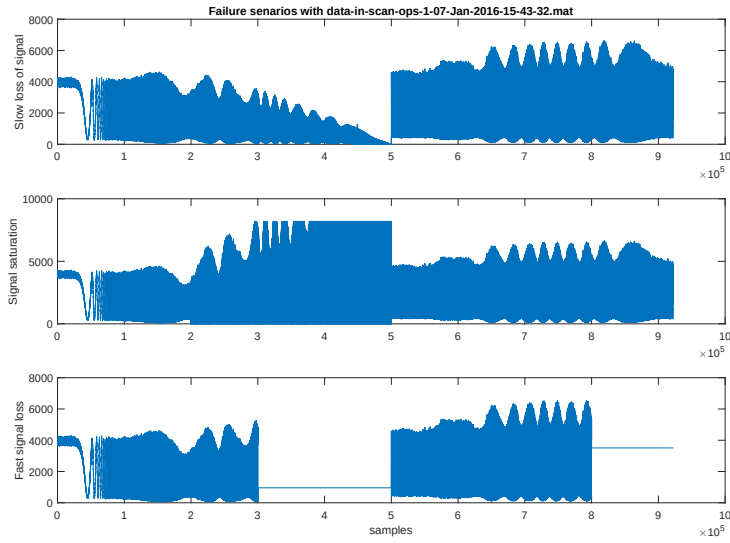


Figure 5.7: IOPS sensor failures simulation of CH1 based on field data

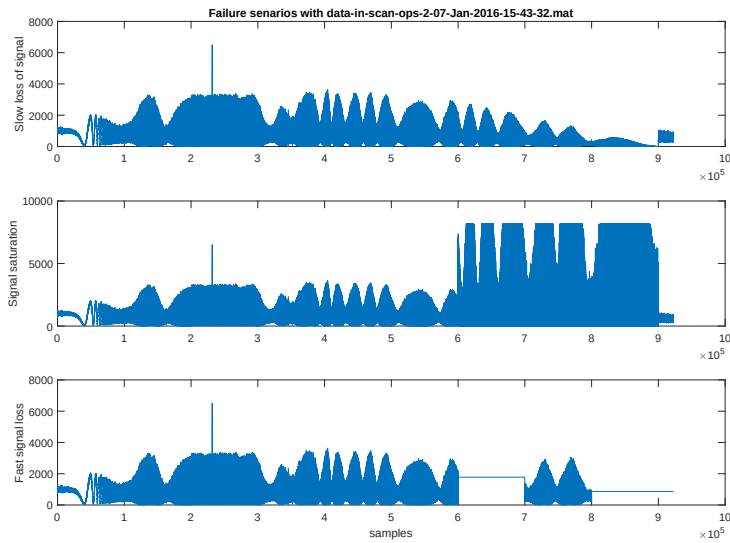


Figure 5.8: IOPS sensor failures simulation of CH2 based on field data

Chapter 6

Data processing overview

This chapter presents an overview of the manipulations applied to the sensors data on the final implementation into the processing target. In later chapters dedicated to each sensor, the prototyping of the processing algorithms done with MatLab will be described along with more details on the implementation.

The aim of this thesis is to perform monitoring of the system by means of sensors data stream processing. The detectability is an main component of an reliability strategy. Nevertheless, the mitigation of some defined failures listed in Chap.3 is taken into account. The achieve these goals, two ways are investigated in this work.

Firstly, the processing of incoming data flow of a particular sensor is treated individually to discover eventual problems, such as high noise, glitches or none physical rate.

Secondly, since some of the sensors are caring redundant information, the comparison of their absolute value can help to detect wrong measures, even mitigate them by switching between sensor stream or by generating an output calculated from multiple sources.

Fig.6.1 presents a sensor acquisition and processing chain, from the physical quantity (left side) to the delivery of a digital value representing the quantity (right side) to further processing inside the FPGA. The monitoring is performed on raw value and on values having the physical unit. In this work, the physical quantity processed are, radian [rad] for the position, radian per second [rad/s] for the speed and ampere [A] for the motor phases current.

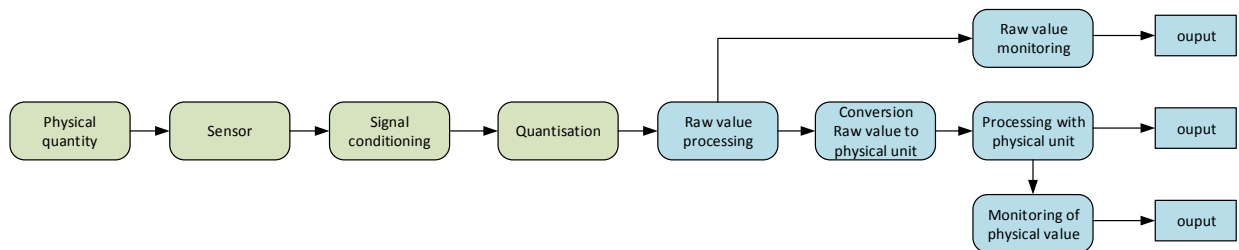


Figure 6.1: A physical quantity acquisition chain. the blue part represents the scope of this work

6.1 Input value monitoring principle

On each sensor data processing unit, the raw input value is filtered to reject large noise, and eventual jumps of the input lasting less than three value. this is performed using the architecture of Fig.6.2 where the data is travelling from the left side to the right side. The rejection principle uses a the median filtering which is detailed in the subsection 12.1.3. The output of this filter is sent to further processing, which will be the conversion of this value to a physical representation having si unit. The difference between input and output of the median filter will then be used for monitor the 'quality' of the input stream. For this, the difference is made absolute

and compared against a predefined threshold. If the difference is above, a status will be raised and a counter incremented. At each new incoming value, this processing is performed. Further processing can then decide based on the current status and on how many times the fail condition occurred.

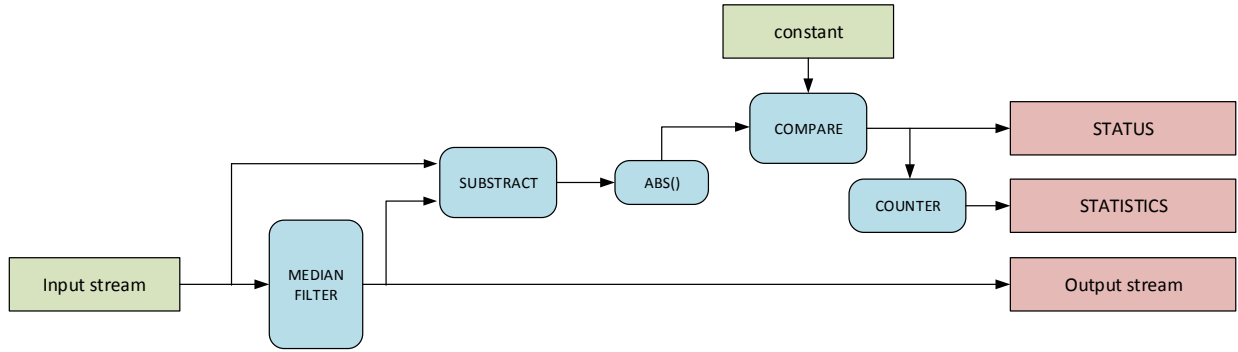


Figure 6.2: Input filtering and monitoring

6.2 Data flow hierarchical view

The data flow hierarchical view is presented here to summaries what will be detailed in the chapter dedicated to each sensor. Fig.6.3 presents the work realised during this thesis. The input stream from the sensors are represented in green (left of the figure) while the outputs are in red (right of the figure). The processing itself is blue at the centre of the figure. Each processing module have a number to ease the reference to it.

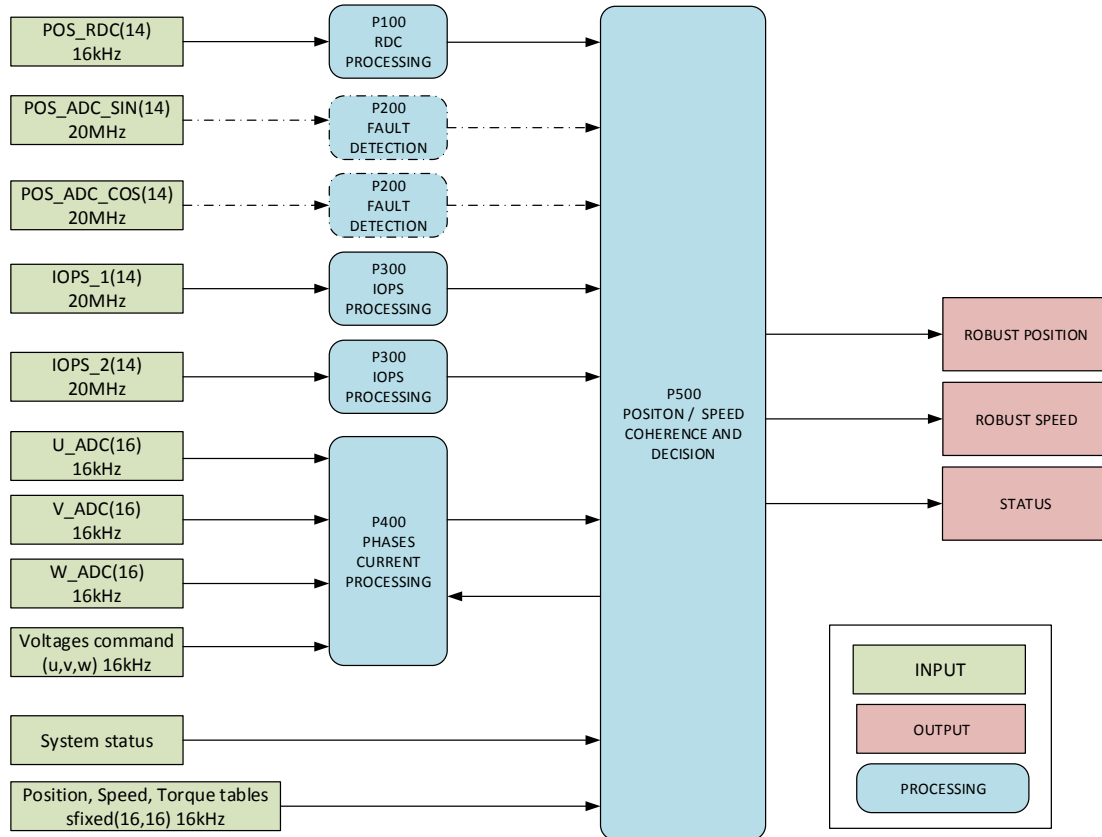


Figure 6.3: Sensors processing data flow overview

The following subsections are presenting the summary of the processing for each sensors.

6.2.1 Resolver

The resolver data flow is presented on Fig.6.4. the incoming data is firstly filtered using a median filter and the difference between input and output is monitored. Then the value is converted to correctly represent the physical quantity, the angular position in radian. This position is then integrated to be able to deal with multiple rotation, as the sensor can only represent one rotation. Finally the resulting position is processed by a convolution filter to obtain its derivative which is the speed of the system. All these resulting quantities are monitored against predefined thresholds and the status are send further.

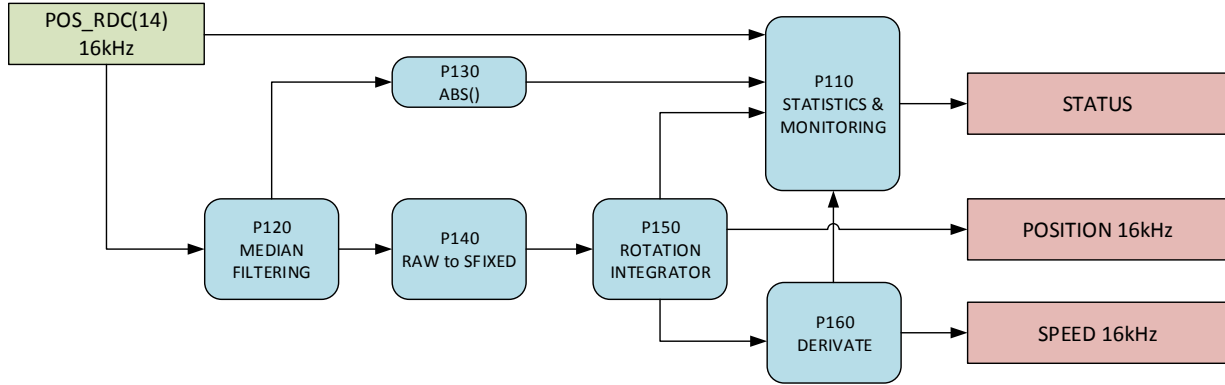


Figure 6.4: RDC data processing

6.2.2 IOPS

The Incremental Optical Position Sensor (IOPS) data flow is presented on Fig.6.5. The incoming data is filtered using a median filter and the difference between input and output is monitored. Then the processing is extracting the position information from the data. This is done by comparing the value against a calculated threshold to determine the length of the signal period, and the system angular position (P340). The mean of the signal and the system speed is calculated by the module P350. The thresholds are calculated at every new period by the module P360, using multiple signal periods to obtain smooth values. The calculated speed is filtered to exclude eventual peaks. All the resulting quantities are monitored against predefined thresholds and the status are send further.

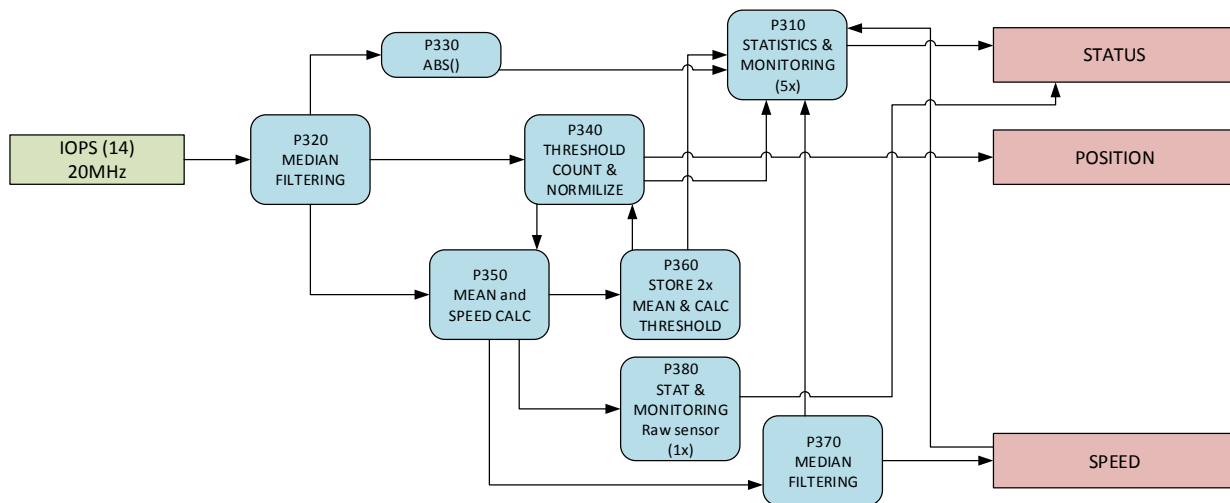


Figure 6.5: IOPS data processing

6.2.3 Motor phases current

The data flow of the currents is presented on Fig.6.6. The blue part as been implemented while the white part is foreseen for future work. Each of the three incoming data (u, v, w) is firstly filtered using a median filter and the difference between input and output is monitored. Then the values are converted to correctly represent the physical quantity, the currents in Ampere. To detect failure, the three current are sum (P440) and the result is monitored using a predefined threshold (P410). the absolute value of the currents are monitored as well against predefined thresholds and the status are send further.

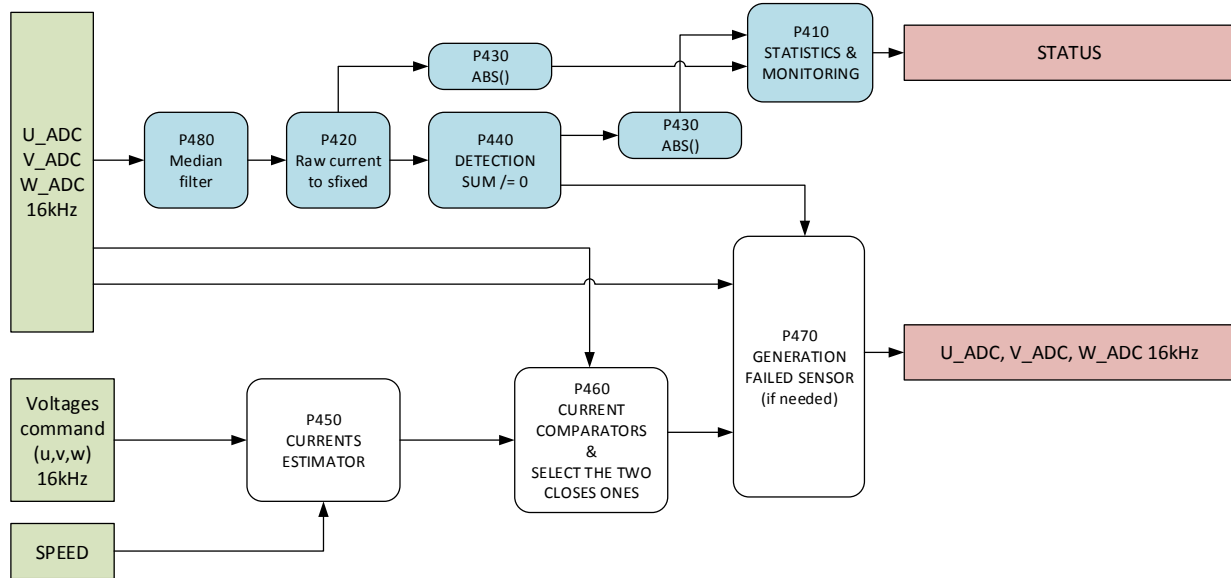


Figure 6.6: Motor current data processing

6.2.4 Sensors fusion

The last part of the processing is the fusion of all the information produced by individually processed sensors. Fig.6.7 represents this fusion. The blue part as been implemented while the white part is foreseen for future work. There is now four sources of information on the system which have to be combined. The position and the speed is processed using the same module (P510) which will described in details in Chap.10 dedicated to this subject.

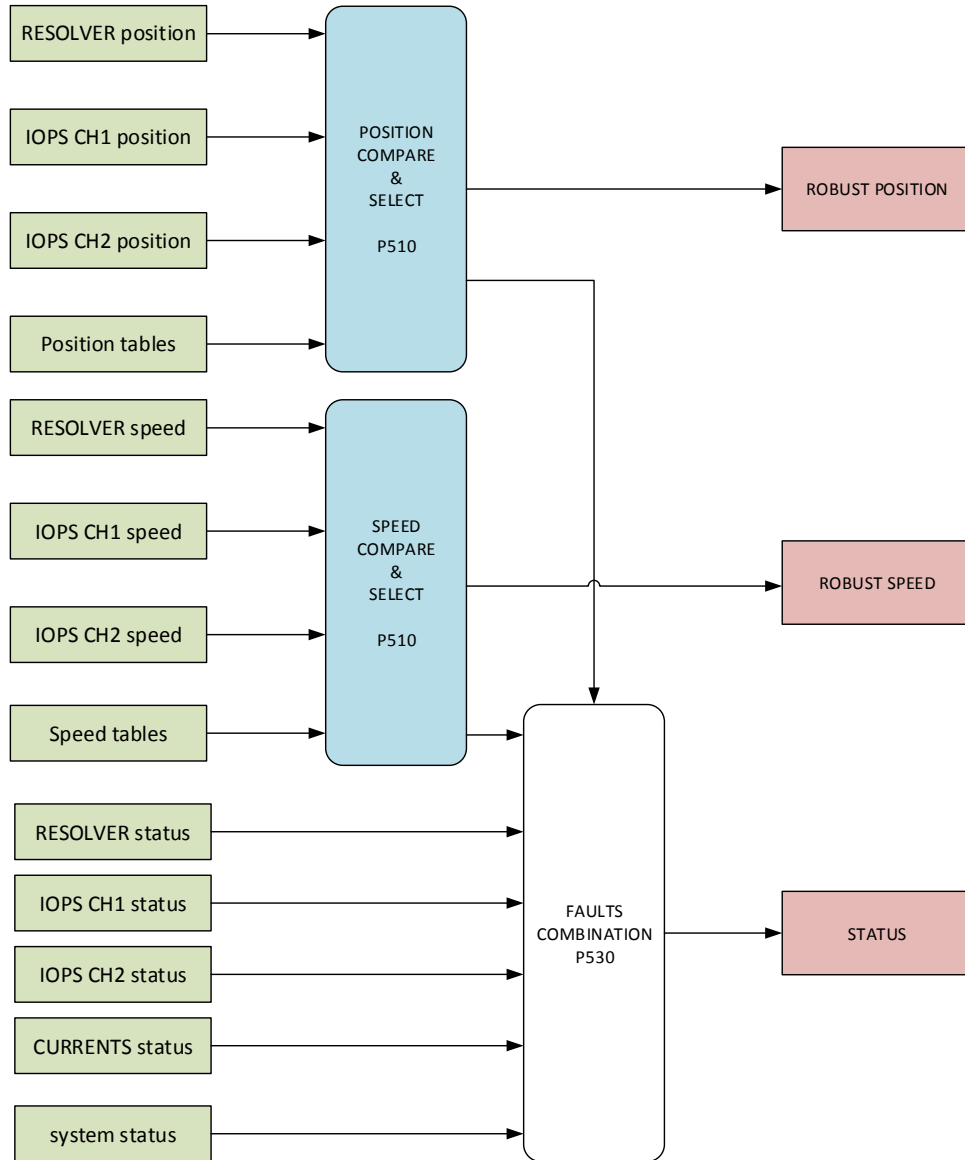


Figure 6.7: Sensors fusion processing

Chapter 7

Solid rotor resolver based absolute encoder

The solid rotor resolver is an angular measurement sensor which is providing an measure of the shaft angle. It provides an absolute information in the range of one complete rotation, i.e. it can't provide an information over multiple rotations. this sensor is firstly used to correctly power the Permanent Magnet Synchronous Motor (PMSM) phases. As this type of motor requires this information for functioning, it must always be able to find the real angular position of the shaft, even after a power cut that could erase information and displace the shaft without being noticed.

Secondly, the resolver measure is used by the controller to displace the shaft at the desired position with a given profile of position, speed and acceleration. During a measure cycle, the controller is following predefined profiles which is verified by the position and speed feedback, based on the resolver information.

The absolute position encoding is obtained by connecting the resolver to a Resolver to Digital Conversion (RDC) Application-Specific Integrated Circuit (ASIC) from Analog Devices which resolve the position and provides an encoded measure to the Field-Programmable Gate Array (FPGA) where the controller is implemented.

7.1 Technology

The technology used in this resolver allow the use of this rotor in Ultra High Vacuum. It uses the variation of the coupling between a primary coil and two secondary coils displaced by $\pi/2$ [rad]. The rotor is build with two materials, one a-magnetic, transparent to the magnetic field and the other which conducts magnetic fields. By careful mechanical design of these two pieces, it is possible to obtain primary-secondary coupling variation proportional to the rotor angle. Fig.7.1 is a technological view of this resolver principle, image sourced from [21]. The rotor is in the centre, primary coil on the left, in phase secondary (sine) at the bottom, while the displaced by $\pi/2$ [rad] secondary (cosine) is on the right.

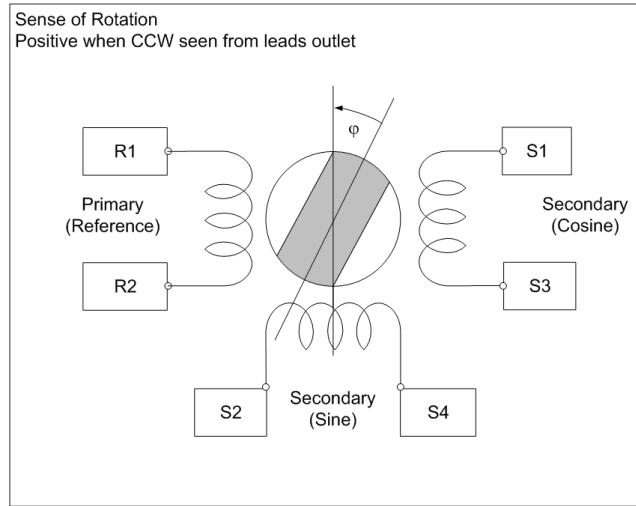


Figure 7.1: Solid rotor resolver technological view sourced from [21]

Fig.7.2 shows an example of the solid rotor resolver from Admotec. The stator and rotor are independent (frame less) as this type of device is usually mounted on the two sides of a tube which separate machine region. The rotor is in the inner part of the tube, being inside a liquid or under vacuum, and the stator on the outer side of the tube, at atmospheric pressure.

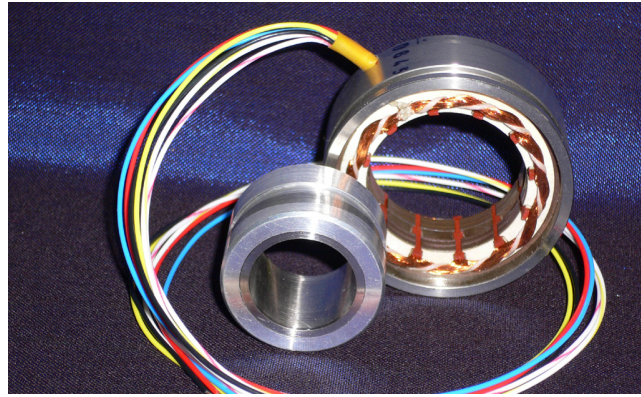


Figure 7.2: Solid rotor resolver form Admotec used for absolut angular measure of the shaft position

Each of these devices are coming with a test report where we can found the absolute accuracy measured. This value variates from device to device and is between 20 to 40 ['] (arcminute) or 5.8 to 11.6 [mrad].

7.2 Resolver connections

The resolver is driven by a custom made analogue conditioning circuit to be able to drive the low inductive coils of its primary circuit. The secondary coils are connected to custom made buffering circuit to condition their signals. The analogue conditioning circuits are connected to the Resolver to Digital Conversion (RDC) Application-Specific Integrated Circuit (ASIC) which resolve the position by processing the signals and tracking the position by means of a feedback tracking principle.

This circuit is connected the FPGA by a Serial Peripheral Interface (SPI) where various information are exchanged between them, including the absolute resolved position. The resolution at which the RDC is providing the position is selectable. The more resolution, the slowest the measure response is. For this project, the resolution is set to 14 [bit] providing a theoretical resolution of 0.383 [mrad],

In addition to this circuit, it is foreseen in the coming revision of the system to include a direct digitalisation of the secondary signals at a speed equal or higher than 20[MHz]. The aim of this part will be the direct monitoring of the signal for checking the integrity of the sensor before the RDC. It will also be possible to resolve directly the position by processing the signals inside the FPGA.

Fig.7.3 presents the schematics of the resolver connection to the FPGA. The resolver is situated in the underground accelerator tunnel while all the driving and processing part are situated far away in the surface building to be protected against radiation damages. This type of topology has been tested up to 250 [m].

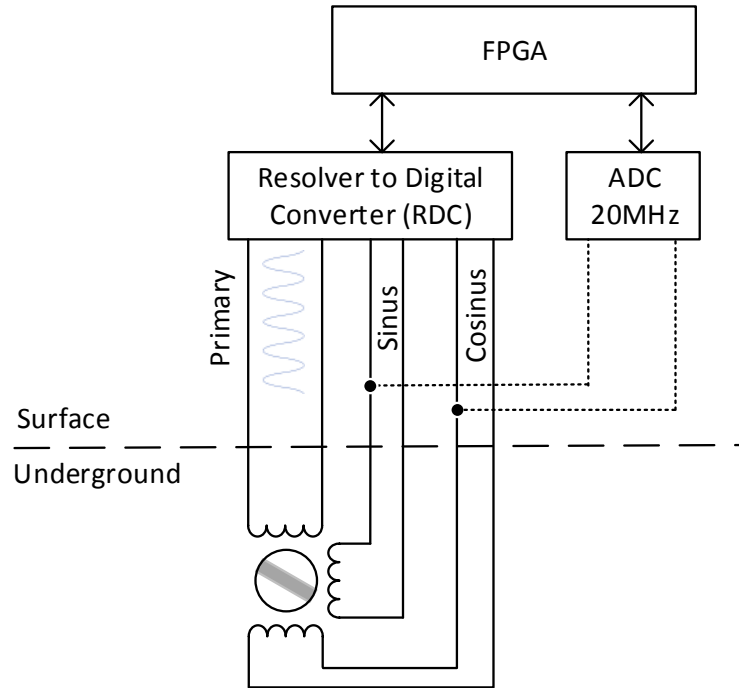


Figure 7.3: Resolver system overview

7.3 field data characterisation

The first step of the data processing was to develop a mechanism to look at the data sets and to characterise these measurements. Fig.7.4 shows an example of the analysis script output for one measure cycle. The position given by the resolver and the reference position is shown on the top while the bottom plot represents the derivative compared to the reference speed.

Multiple indicators are calculated such as the minimum, maximum and mean values for the position (top) and for its derivative (bottom). Finally, the noise is estimated by selecting a portion of the available measures and performing a linear fit (centre plot of the middle plot line). The standard deviation of the residue between the position and the linear fit is taken as reference value for the noise histogram (right plot in the middle). For instance, the standard deviation of the noise is 0.1 [mrad] in this example.

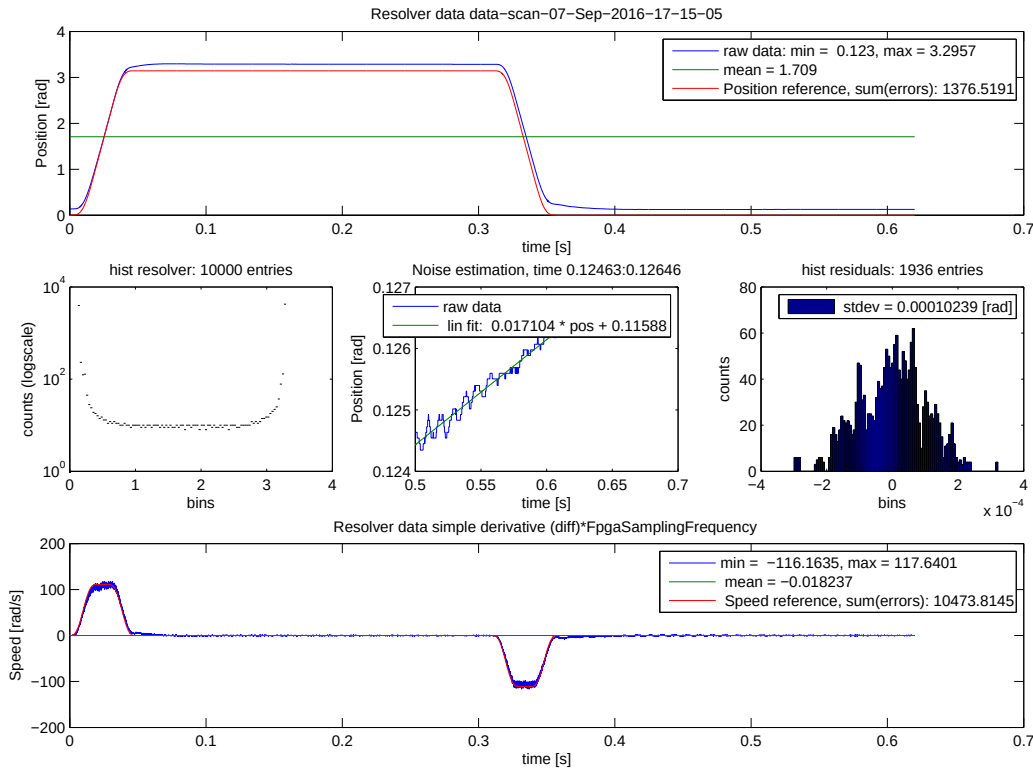


Figure 7.4: Resolver measurement characterisation from the 7 September 2016 at 17h15m05s

By processing these indicators for multiple measures, it is then possible to represent them on the same plot representing them for each measure cycle of a given data set. Fig.7.5 presents that result for the data set number 10 and 11 (7 and 9 December 2016). This kind of representation is inspired by the control charts developed from Statistical Process Control (SPC) in industrial manufacturing.

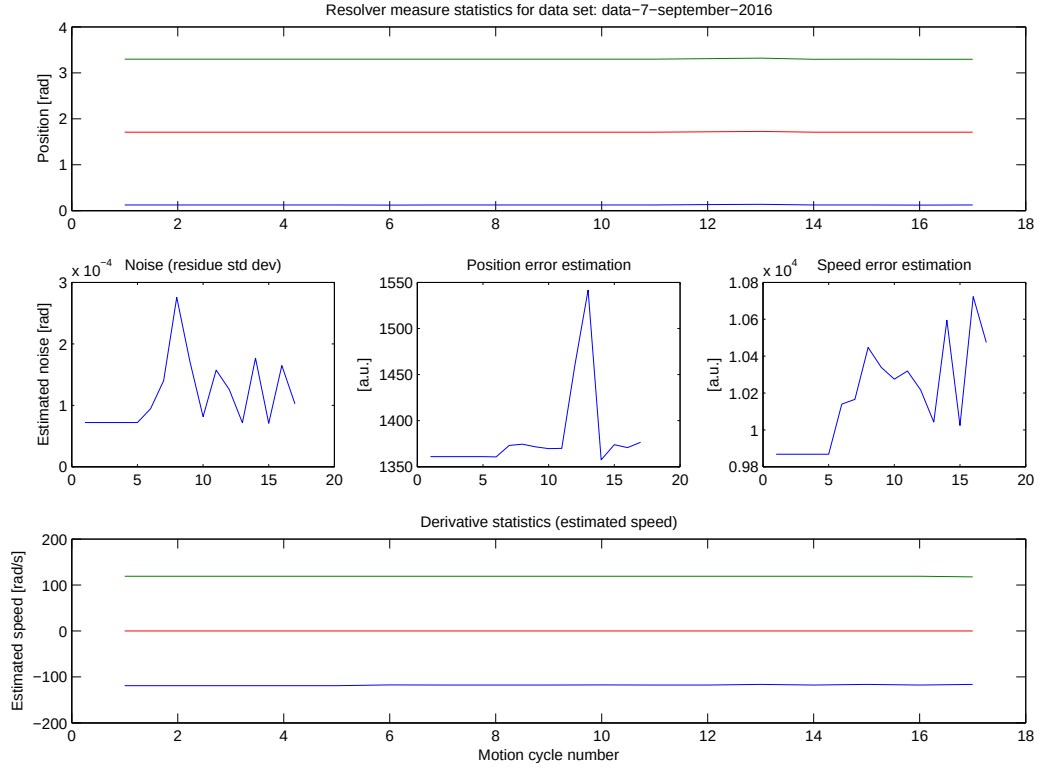


Figure 7.5: Resolver data indicators for the data set (7 September 2016)

The MatLab script to generate these statistics is integrated in the processing algorithm prototype which is described in the next section [7.4](#).

7.4 Processing algorithm prototyping

The input given to the algorithm is the shaft angle, in unsigned binary, 14 bits from 0 to $2^{14}-1$, representing an angle from 0 to $2 \cdot \pi$ in radian. To be noted that the raw value that the FPGA will be processing is not directly available in the data set. the only available position is already in radian. In order to prototype the future processing, the position value available in the data set is artificially converted in 14 [bit] unsigned integer representation. The MatLab code for this operation is

```
uint16pos = uint16(pos/(2 * pi) * 214 - 1);
```

This raw position needs first to be cleaned. Since discontinuities were observed, such as values going to the minimum position for one sample, a median filter with a length of seven samples is implemented. This allows 3 samples to be completely off the median value. The MatLab instruction is

```
mpos = uint16(medfilt1(double(uint16pos), 7));
```

Once this value is cleaner, it needs to be converted into an representation with physical unit allowing various calculations. The representation in signed fixed point 32 bits was chosen. To take into account an eventual multi-rotation situation, an rotation integrator needs then to be inserted.

The system also need to extract an approximation of the speed, various methods to calculate the speed are possible and many were investigated in [22]. Another way was investigated in this thesis by using a convolution filter to perform a Savitzky Golay filter [23]. By selecting the appropriate coefficients, it can calculate the first derivative of the position and obtaining the speed already smoothed. The following coefficients were selected for this filter: coefficients = [4 3 2 1 0 -1 -2 -3 -4] and the normalisation = 60.0/16129.0. With MatLab, this filter is implemented with

```
coefsgolay = [4, 3, 2, 1, 0, -1, -2, -3, -4];
```

```
speed = conv(coefsgolay, double(mpos))/60 * 16219;
```

The last addition to the processing is the monitoring which checks that values are not exceeding the expected. For this sensor, the monitored values are the integrated position, the speed and finally the difference between the median filter output and the input signal. This last check allows to see if largely diverging samples where rejected by the median filter. Fig.7.8 shows the result of the resolver measure processed with the prototyped algorithm.

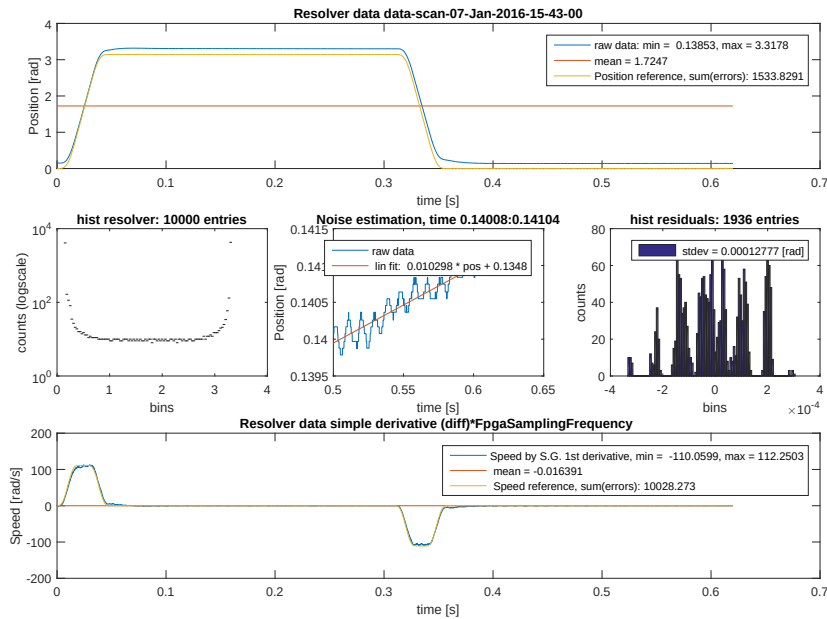


Figure 7.6: RDC processing - plots

7.5 VHDL Implementation

The processing algorithm for the resolver has been written as a self contained entity as defined in Chap.6. Fig.7.7 show the entity with its inputs and outputs.

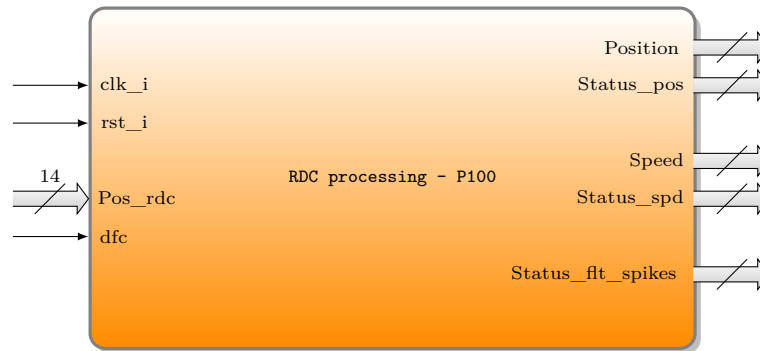


Figure 7.7: The resolver processing component

The conversion of the prototype algorithm written in MatLab to a VHSIC Hardware Description Language (VHDL) description was performed manually according to the defined procedure in 4.2. the following paragraph describing the implementation comes from the Data processing overview chapter in 6

The resolver data flow is presented on Fig.7.8. the incoming data is firstly filtered using a median filter and the difference between input and output is monitored. Then the value is converted to correctly represent the physical quantity, the angular position in radian. This position is then integrated to be able to deal with multiple rotation, as the sensor can only represent one rotation. Finally the resulting position is processed by a convolution filter to obtain its derivative which is the speed of the system. All these resulting quantities are monitored against predefined thresholds and the status are send further.

the following components are described in details in Chap.12: the median filter (P120), the statistics and monitoring (P110) as well as the derivative (P160) under the convolution section.

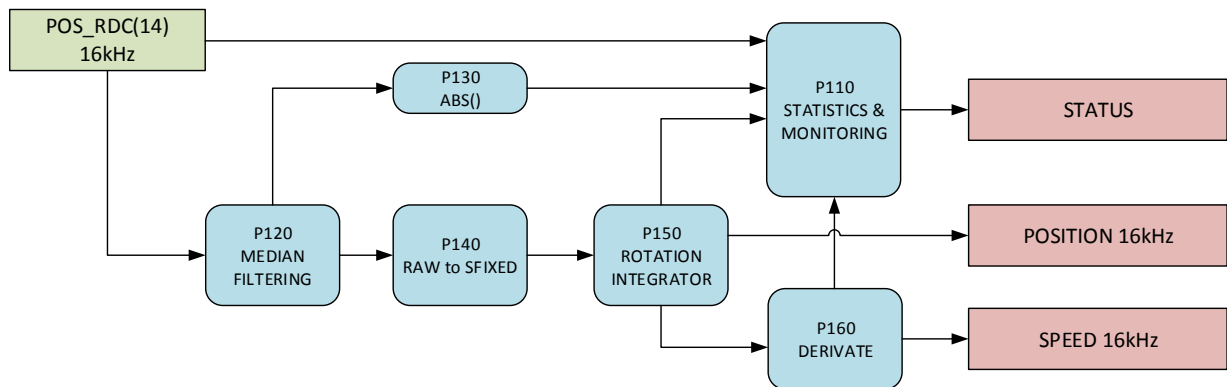


Figure 7.8: RDC processing - P100

7.6 Synthesis

The module 'rdc-processing-P100' is build with the following components that have been individually written and verified:

1. median-filter-raw-data
2. data-raw-to-sfixed
3. rotation-integrator
4. convolution-sfixed-data-generic-size
5. abs-sensor-processed-data
6. monitoring-processed-sensor-data

Fig.7.9 details the initial synthesis result of the module resolver. We can see for example multiple instances of the component 'monitoring-processed-sensor-data' as it is used to monitor multiple parameters and needs to be replicate in the FPGA for parallel operation.

Analysis & Synthesis Resource Utilization by Entity							
<<Filter>>							
	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Block Memory Bits	DSP Blocks	Pins	Virtual Pins
1	rdc_processing_P100	2547 {0}	1623 {0}	0	35	191	0
1	abs_sensor_processed_data:duv10	32 {32}	23 {23}	0	0	0	0
2	convolution_sfixed_data_generic_size:duv6	1888 {1888}	636 {636}	0	30	0	0
3	data_raw_to_sfixed:duv3	58 {58}	24 {24}	0	2	0	0
4	median_filter_raw_data:duv1	145 {145}	771 {771}	0	0	0	0
5	monitoring_processed_sensor_data:duv2	28 {28}	27 {27}	0	0	0	0
6	monitoring_processed_sensor_data:duv5	31 {31}	18 {18}	0	0	0	0
7	monitoring_processed_sensor_data:duv7	29 {29}	27 {27}	0	0	0	0
8	rotation_integrator:duv4	336 {336}	97 {97}	0	3	0	0

Figure 7.9: Synthesis summary of the resolver processing (P100) on Arria V FPGA

The challenge in the synthesis, place and route and fitter process was related to the maximum clocking frequency. As this is linked to the way the components are written, care must be taken to limit the combinational path slowing down the propagation of the signals from register to register. One optimisation of that type was performed on the convolution core in Chap.12. For the specific entity of the resolver the problem was lying in the 'rotation-integrator' component.

This component is responsible of integrating the position in case of multiple rotations of the system. Even if the scanner is not supposed to rotate multiple times, this case is covered. Fig.7.10 shows a simulation of this component with Modelsim. We can distinguish multiple rotation simulation applied to the component with the signal called 'input-stimuli-sti'. One rotation is simulated by a smooth ramp until the maximum position and a discontinuity from the maximum to the minimum position. the component reacts as expected and present an output visualised with the vector 'output-stimuli-obs' which constantly increases until the rotation direction is inverted.

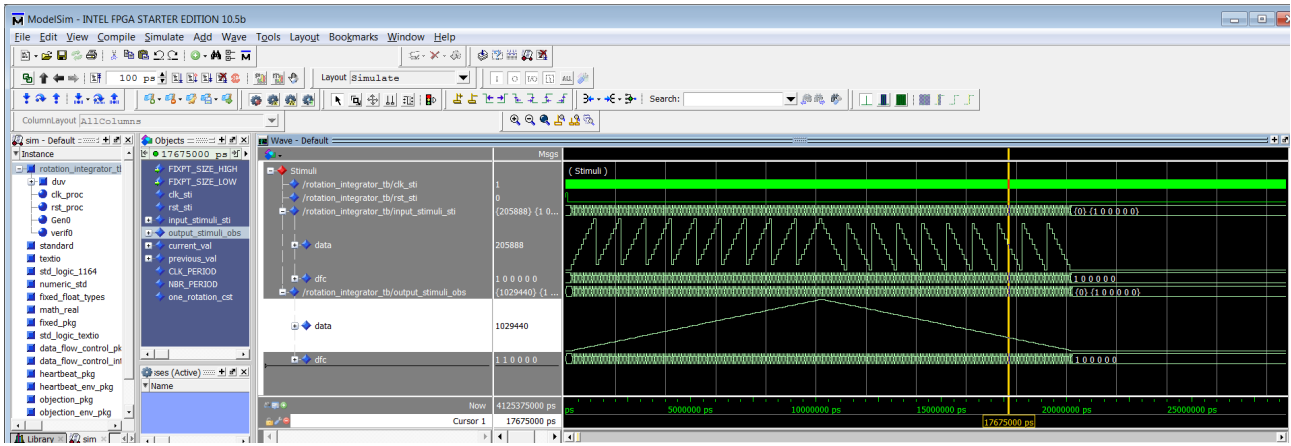


Figure 7.10: resolver rotation integrator test bench - P140

The maximum clocking frequency of the first implementation was only allowing this module to run at about 50MHz, which was producing too long path when it was integrated into the complete system. To tackle the worst case path, the tools was used. Fig.7.11 show a screen capture of the interface where we can identify this problem. The tool called TimeQuest provides in addition, a proposal for improving the situation, in this case, to lower the combinational logic.

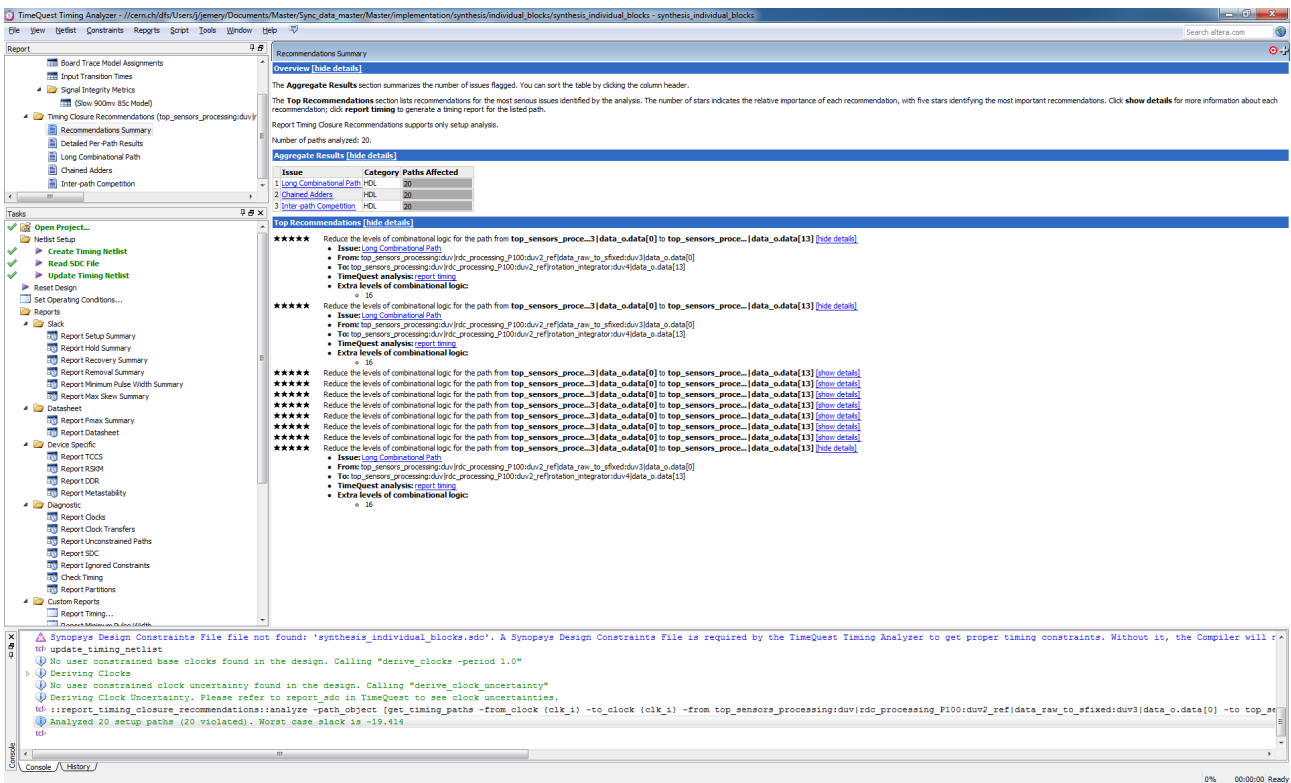


Figure 7.11: Intel TimeQuest Timing Analyzer used to identify the worst case path

The tool identified the exact path taking the longest time to propagate. As already said, this path for the P100 module was inside the rotation integrator where multiple mathematical operation were performed in a row. A subtraction with a multiply and finally an addition was done without pipeline as shown in the following VHDL code.

```

-- Calculate the difference between incoming position and previous position in fixed point
diff_pos := resize( to_sfixed(data_i.data, FIXPT_SIZE_HIGH - 1, FIXPT_SIZE_LOW) -
                    to_sfixed(previous_position.data, FIXPT_SIZE_HIGH - 1, FIXPT_SIZE_LOW),
                    diff_pos);
...
-- Calculate the new integrated position using the memorised number of turns and the current position
data_o.data <= std_logic_vector(resize( to_sfixed(data_i.data, FIXPT_SIZE_HIGH - 1, FIXPT_SIZE_LOW) +
                                        one_rotation_sfised *
                                        to_sfixed(rotation_memory, FIXPT_SIZE_HIGH - 1, FIXPT_SIZE_LOW),
                                        one_rotation_sfised));

```

A modified version which pipelines the mathematical operations and adapt signals delays (not shown here) was written to mitigate it:

```

-- Calculate the difference between incoming position and previous position in fixed point
diff_pos <= resize( to_sfixed(data_i.data, FIXPT_SIZE_HIGH - 1, FIXPT_SIZE_LOW) -
                    to_sfixed(previous_position.data, FIXPT_SIZE_HIGH - 1, FIXPT_SIZE_LOW), diff_pos);
...
-- Calculate the new integrated position, first by calculating the number of rotations in radians
integer_position_sfised <= resize( one_rotation_sfised *
                                   to_sfixed(rotation_memory, FIXPT_SIZE_HIGH - 1, FIXPT_SIZE_LOW),
                                   integer_position_sfised);
...
-- then by summing the integer number of rotation and the current fraction of it
data_o.data <= std_logic_vector(resize( to_sfixed(pipelined_2_data_i.data, FIXPT_SIZE_HIGH - 1, FIXPT_SIZE_LOW) +
                                        integer_position_sfised,
                                        one_rotation_sfised));
data_o.dfc <= pipelined_2_data_i.dfc;

```

The improvement in clocking speed is about 28 [%], from 48[MHz] to 65.5[MHz] when compiled inside the top entity. The drawback of this technique is the additional registers needed for storing the intermediate signals (the pipeline process). Fig.7.12 shows the synthesis results with the pipelined version of the 'rotation-integrator' module. Resource utilisation for the Logic Cell (LC) went from 97 to 240, an increase of 2.5 times. For this project, the free resources inside the FPGA are large and the impact of this increase is small compared to the gain in clocking speed of the overall system.

Analysis & Synthesis Resource Utilization by Entity								
	Compilation Hierarchy Node	LC Combinationals	LC Registers	Block Memory Bits	DSP Blocks	Pins	Virtual Pins	Full Hierarchy Name
1	└─ rdc_processing_P100	3214 (0)	2142 (0)	0	35	191	0	rdc_processing_P100
1	└─ abs_sensor_processed_data:duv10	32 (32)	23 (23)	0	0	0	0	rdc_processing_P100 abs_sensor_processed_data:duv10
2	└─ convolution_sfised_data_generic_size:duv6	2532 (2532)	996 (996)	0	30	0	0	rdc_processing_P100 convolution_sfised_data_generic_size:duv6
3	└─ data_raw_to_sfised:duv3	76 (76)	32 (32)	0	2	0	0	rdc_processing_P100 data_raw_to_sfised:duv3
4	└─ median_filter_raw_data:duv1	163 (163)	771 (771)	0	0	0	0	rdc_processing_P100 median_filter_raw_data:duv1
5	└─ monitoring_processed_sensor_data:duv2	28 (28)	27 (27)	0	0	0	0	rdc_processing_P100 monitoring_processed_sensor_data:duv2
6	└─ monitoring_processed_sensor_data:duv5	33 (33)	26 (26)	0	0	0	0	rdc_processing_P100 monitoring_processed_sensor_data:duv5
7	└─ monitoring_processed_sensor_data:duv7	29 (29)	27 (27)	0	0	0	0	rdc_processing_P100 monitoring_processed_sensor_data:duv7
8	└─ rotation_integrator:duv4	321 (321)	240 (240)	0	3	0	0	rdc_processing_P100 rotation_integrator:duv4

Figure 7.12: Synthesis summary pipelined version of the resolver processing (P100) on Arria V FPGA

7.7 Verification

The verification of the complete resolver processing was performed using stimuli generation from the real data sets. The output was then compared with the prototyped algorithm in Matlab.

To do this, the complete top entity 'rdc-processing-P100' was connected to the field data generator and the outputs were saved into individual text file. the test bench entity description below shows that many generics parameters are available. this is to be able to simulate the design for multiple set of input data without having to recompile it. The 'OUTFOLDER' generic allows to save multiple results set into separated folders.

```
entity rdc_processing_P100_tb is
  generic(
    OUTFOLDER          : STRING;
    IOPS_1_DATA        : STRING;
    IOPS_2_DATA        : STRING;
    IOPS_DEPTH         : integer;
    RESOLVER_DATA      : STRING;
    POSITION_DATA       : STRING;
    POS_DEPTH          : integer;
    CURR_U_DATA        : STRING;
    CURR_V_DATA        : STRING;
    CURR_W_DATA        : STRING;
    CURR_DEPTH         : integer;
    FIXPT_SIZE_HIGH    : integer := 16;
    FIXPT_SIZE_LOW     : integer := -16
  );
end rdc_processing_P100_tb;
```

the instructions to compile and run the simulation are located into a Tool Command Language (TCL) script started from the command line of Modelsim.

to compile one entity, the instruction 'vcom' is used as in this exemple:

```
vcom -2008 -work work $path_vhdl/rdc_processing_P100.vhd
```

The order of the compilation must be going from libraries, such as the one for the sfixed, to the small building modules to the large components using smaller ones.

the instructions to run ones simulation with a given set of data looks like:

```
vsim -novopt -GOUTFOLDER="out_data/" -GIOPS_1_DATA="data_in_scan_ops_1_07_Jan_2016_15_43_32.hex"
-GIOPS_2_DATA="data_in_scan_ops_2_07_Jan_2016_15_43_32.hex"
-GIOPS_DEPTH=1048575
-GRESOLVER_DATA="data_scan_07_Jan_2016_15_43_00_noisy-pos-res-16k129Hz.hex"
-GPOSITION_DATA="data_scan_07_Jan_2016_15_43_00_noisy-pos-ref-16k129Hz.hex"
-GPOS_DEPTH=10000
-GCURR_U_DATA="data_scan_07_Jan_2016_15_43_00_noisy-current-U-100kHz.hex"
-GCURR_V_DATA="data_scan_07_Jan_2016_15_43_00_noisy-current-V-100kHz.hex"
-GCURR_W_DATA="data_scan_07_Jan_2016_15_43_00_noisy-current-W-100kHz.hex"
-GCURR_DEPTH=62000
work.rdc_processing_P100_tb
run -all
```

Fig.7.13 shows the Modelsim simulator with the resolver processing component being stimulated with input vectors coming from files. the data are based on the set S4 and artificially modified with parasite peaks lasting one sample as in Fig.5.5 to simulate failure mode 'I01-FM3' (short jumps) see Table.5.3

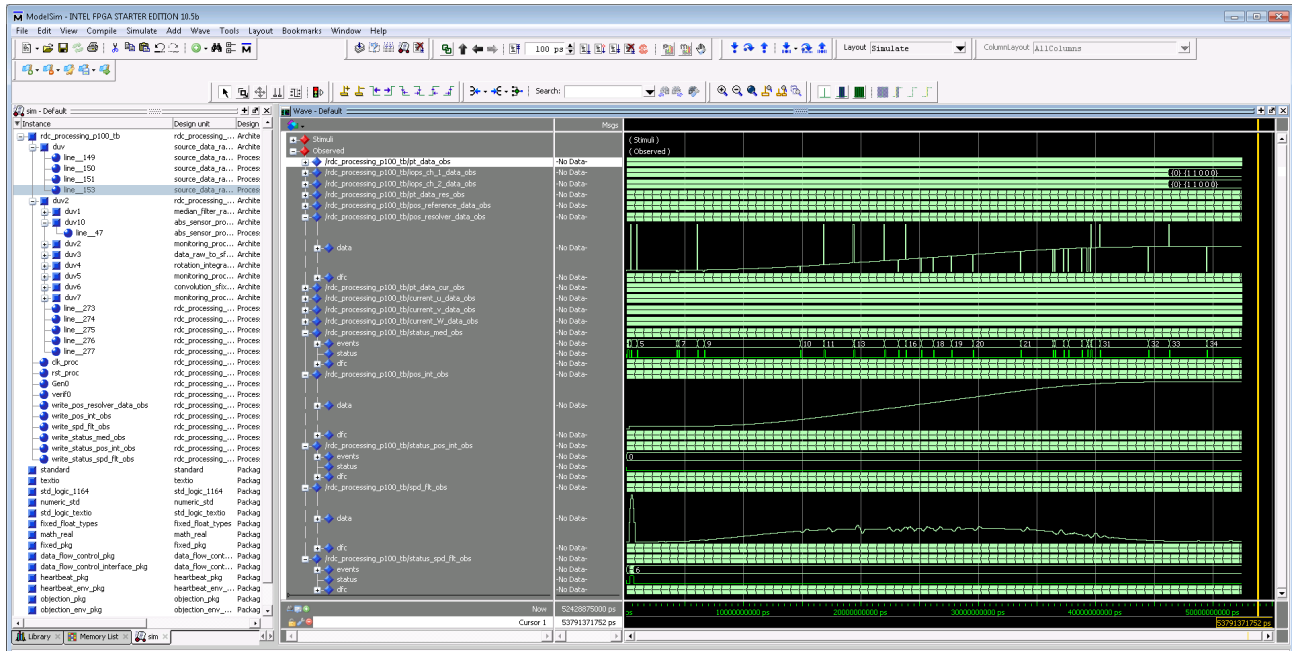


Figure 7.13: Modelsim simulation of the rdc component - P100 with its input stimulated with the data set S4 artificially parasited with peaks lasting one sample

7.8 Algorithm implementations comparison

The output files generated by the test bench have different forms depending on the output types, if it is a structured type or not. For each of these output type, one MatLab function was written to load them into the workspace. Then the script compares these results processed with the VHDL algorithms against the ones generated by the prototypes written with MatLab.

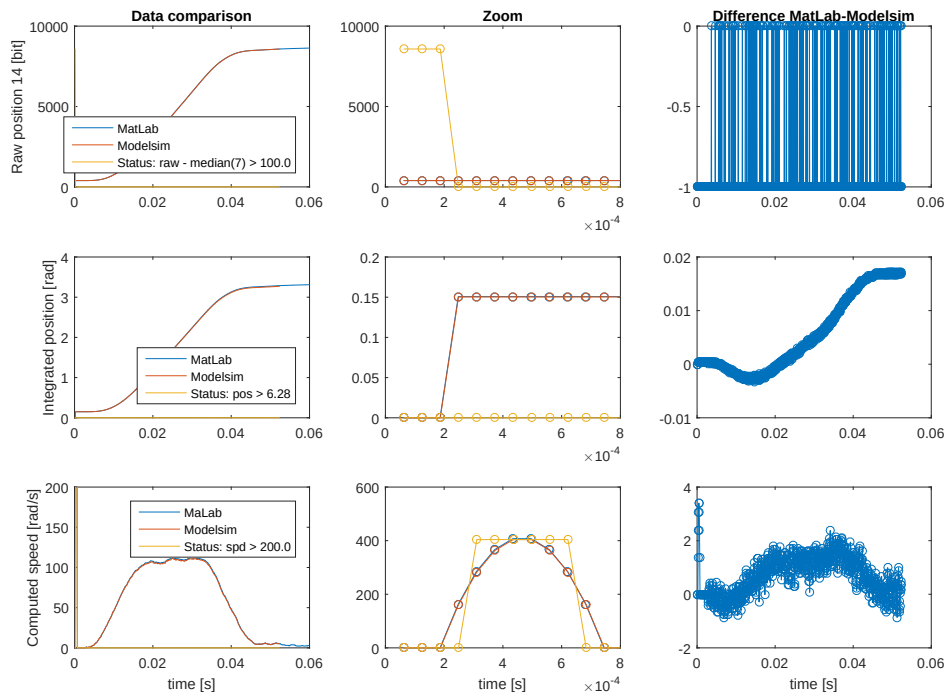


Figure 7.14: Comparison of resolver processing implementations, MatLab versus VHDL using the S4 dataset

Fig.7.14 presents the result of this comparison for the reference data set S4, see Table.8.3. the first line shows the comparison of the 'raw' position after the median filter. Apart from a difference at the beginning of the simulation due to different initial condition, the difference between the two implementation are in the order of one bit. the second line shows the integrated position and the last line the speed generated by the convolution. they are both presenting a difference between the two implementation of no more than 2 [%]. The source of these differences can come from multiple sources, rounding scheme, uncompensated delays and implementation differences. As time was running out, focus was given to complete this level of finalisation with the other core and this errors was considered acceptable for the purpose of this work which is focussed on detection and monitoring

7.8.1 Failure case simulation

To compare the two implementation with other data, another set was used with this module. The modified S4 dataset 5.5, which simulate failure mode 'I01-FM3' (short jumps) c.f.5.3 was compared.

Fig.7.15 shows the output of the VHDL only where we can see some additional details. The first line shows the input data presenting spikes of a duration of one sample and in red is the status which detects these discontinuities (top left plot). The sum of these spikes is given by the accumulated output of the status which is represented on the right. The middle line is the integrated position without the spikes, the processing has removed them and presents a smooth approximation of the position. The last line is the computed speed by the convolution filter. The suppression of the spikes have created some discontinuities not seen on the position, but visible on the derivation (the speed). The oscillation are the result of a very small dirac at the input of the convolution filter.

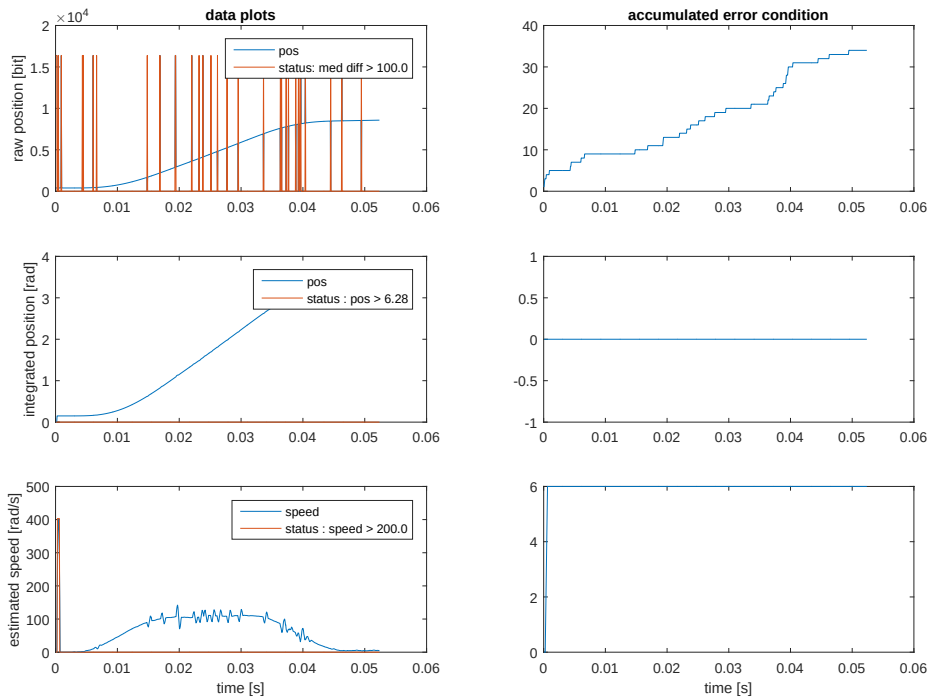


Figure 7.15: VHDL simulation results using modified S4 dataset: 5.5 to simulate failure mode 'I01-FM3' (short jumps) c.f.5.3

Fig.7.16 and Fig.7.17 are showing the comparison of the algorithm in MatLab and VHDL. The difference is sensibly the same, except for the speed. After a short investigation it seems that this difference is mainly due to the small time difference between these simulations. As the delays in MatLab is not completely modelled to fit the VHDL, the difference calculated for a particular time, will be more off as the slope is faster.

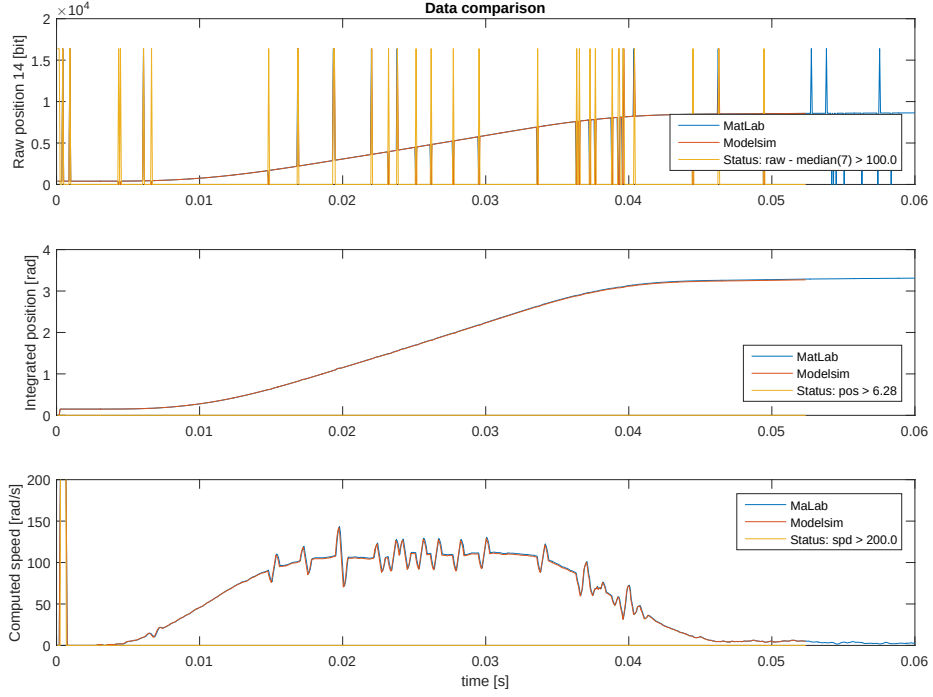


Figure 7.16: Comparison of resolver processing implementations using modified S4 dataset: 5.5 to simulate failure mode 'I01-FM3' (short jumps) c.f.5.3

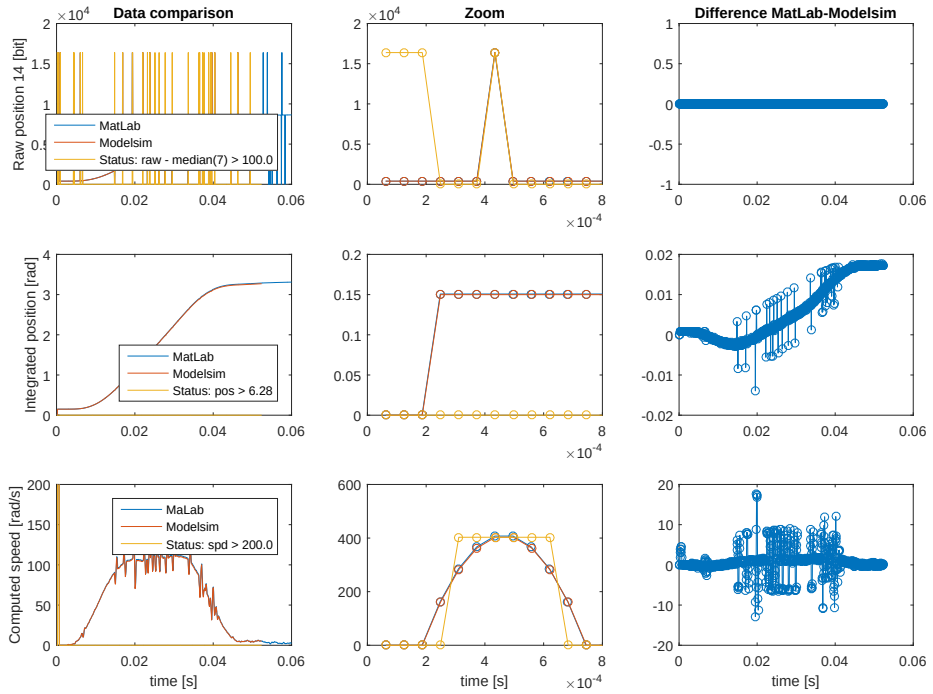


Figure 7.17: Comparison of resolver processing implementations using modified S4 dataset: 5.5 to simulate failure mode I01-FM3 (short jumps) c.f.5.3

Chapter 8

Optical fibre based high precision incremental encoder

The Incremental Optical Position Sensor (IOPS) is an high precision sensor which is used to find the accurate position of the shaft to be used in the determination of the wire position. This sensor is mainly used for measuring the beam, but can also be used as a redundant measure for the control system as developed in this thesis.

8.1 principle

The high precision encoder based on optical fibre was designed at CERN to combine high resolution and compatibility with radiation, vacuum and temperature [3].

The principle is to focus a continuous laser beam on a track of a glass disk having very small mirrors separated by non reflective material. When the scanner starts to rotate, the laser is modulated by the small mirrors and the resulting signal arriving to a photodiode is a quasi sinusoidal signal. Each period of this signal represent 20 μ m. Fig.8.1 shows a picture of one prototype glass disk having multiple mirrors track with different sizes and pitch.

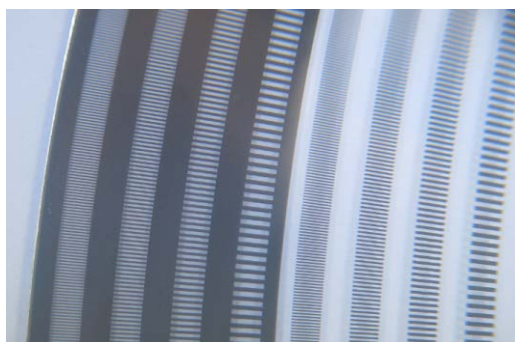


Figure 8.1: Prototype glass disk of the OPS encoder having multiple tracks of different mirror pitch size

This encoder has the ability to measure very small incremental angle under high radiation due to its architecture which places all the sensitive parts far away from the radiations.

Fig.8.2 presents the principle of this encoder. The signal is generated at the surface installation of the accelerator with a small continuous laser diode of 1310 μ m of no more than 2 mW (left side). This light is then sent through a fibre to the scanner and will be modulated by small mirrors (right side). The reflected light is recombined into the same fibre, an optical component called circulator allows to separate the returning light and route it to a photodiode. The analog signal of this photodiode is digitalised resulting with the signal shown of Fig.8.24

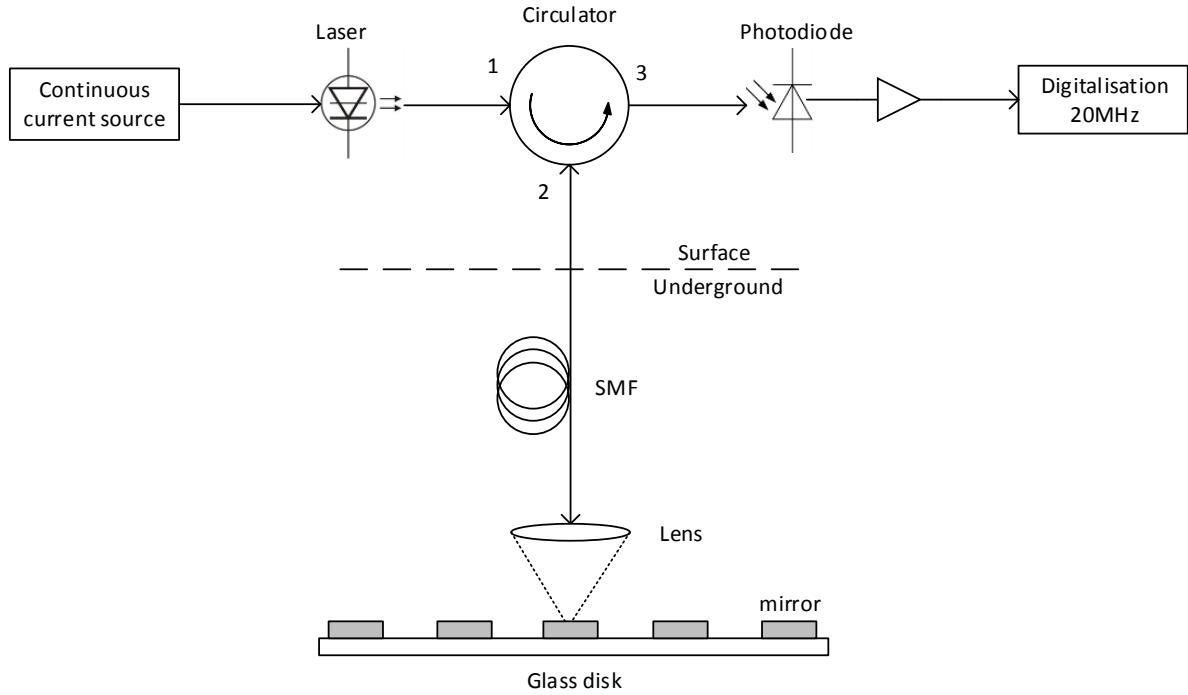


Figure 8.2: Optical incremental position sensor developed at CERN

This encoder is built with a glass disk having very thin reflective part made from Chrome. The process of photolithography ensures a precision of about $1\mu\text{m}$. Multiple prototypes with different patterns sizes have shown that the optimal pitch is $20\mu\text{m}$, $10\mu\text{m}$ chrome and $10\mu\text{m}$ glass. The type of fibre is single mode with a core size of $9\mu\text{m}$ and a wavelength of $1310\mu\text{m}$. The last piece of this encoder at the scanner side is the lens system used to focus the laser beam on the glass disk, and back to the fibre.

The fixation of the disk on the shaft is a crucial parameter. If the disk is off the axis of the shaft, the track will present some eccentricity and will induce error on the measure, as the track diameter will vary while rotating. To mitigate this problem, two measures are made at the same time, with two optical systems displaced by 180° . By making the difference between the two sensors, the eccentricity is compensated. These two sensors, apart from increasing the overall precision of the encoder, can be used as redundancy, if very high precision is not needed.

Fig.8.3 shows a worst case measurement of the IOPS. The bottom plot is the beginning of a scanner movement where the signal period is getting shorter as the system accelerates. The top plot shows the rotation of the disk of $\pi[\text{rad}]$. The variable period sinusoidal signal is perturbed by a modulation that can be caused by different factors such as:

1. Disk vibration which are changing the distance lens - disk
2. Possibly optical interferences since the light is coupled back into the same optical fibre
3. Disk track scratches

The purpose of the illustrated example of Fig.8.3 is only to display what kind of problem this type of encoder can suffer. It is not expected that the processing developed in this thesis will overcome such kind of behaviour. This kind of parasite modulation can be mitigated when discovered.

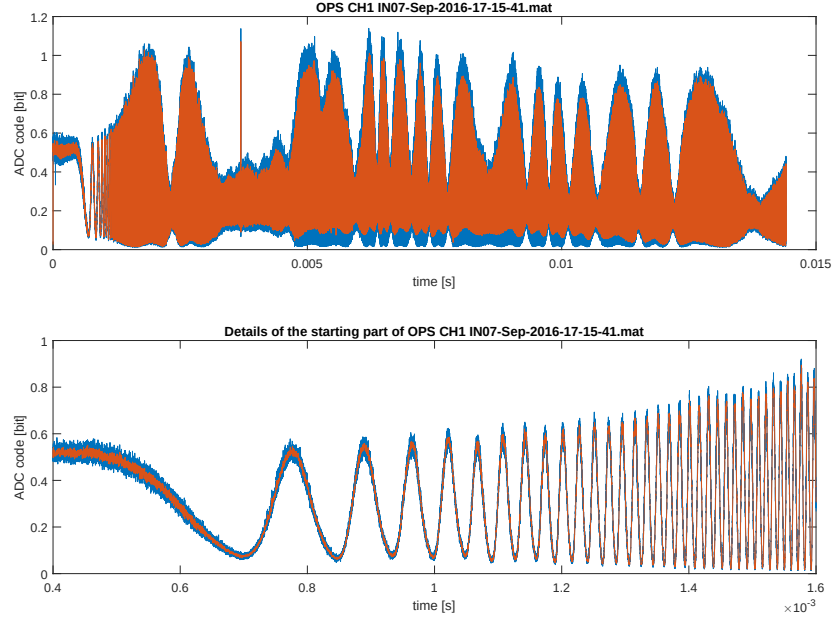


Figure 8.3: IOPS channel one signal recorded on the 7th September 2016 at 17h15m41s

The resolution of the measurement realised with the scanner will depend on the intrinsic resolution of the IOPS and also of the position of the measured beam. In fact, we like to measure a linear physical quantity (the beam transversal size) with a rotational movement. This means that the resolution will vary with the position of the fork. Fig. 8.4 presents the projected resolution versus the angular position of the scanner (top). The bottom plot presents the resolution versus the angular position. The optimal resolution is when the scanner has a movement almost perpendicular to the projected position (sides of the bottom plot), because the fork will mostly move in the perpendicular direction of the projected plan.

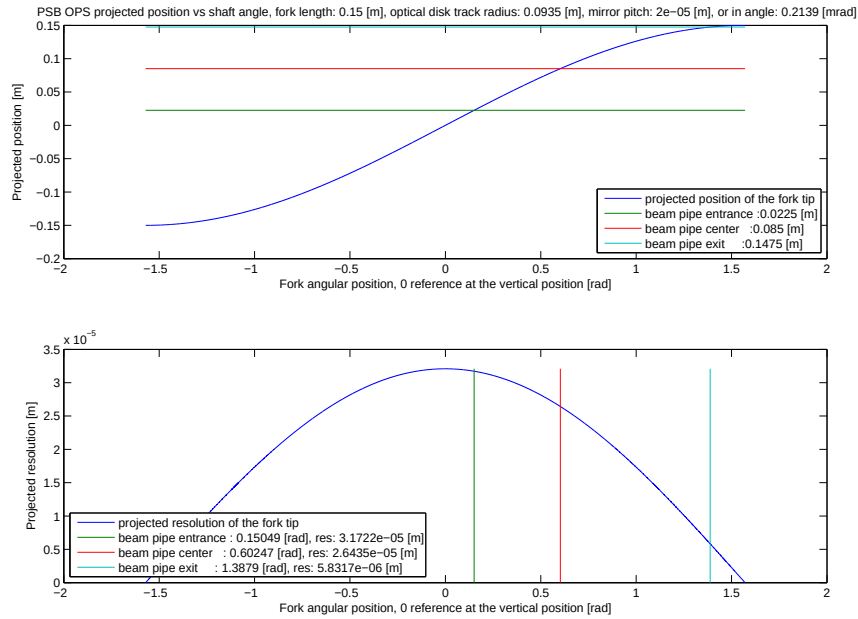


Figure 8.4: PSB Scanner resolution

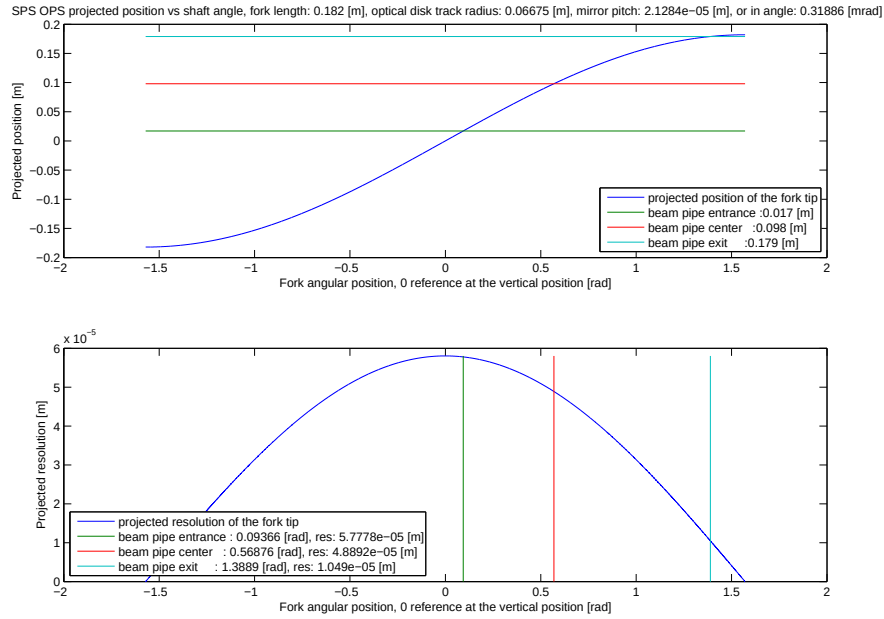


Figure 8.5: SPS Scanner resolution

Table 8.1 summarises the characteristics of two existing IOPS. The available recorded data in this thesis were measured with the SPS version.

Type	Track diam. [mm]	Track length [mm]	Mirror pitch [μm]	Number of mirrors	Angular resolution [μrad]	Angular resolution [mdeg]	Fork length [mm]	Resolution at fork tip [μm]	Projected res. at interaction [μm]
PSB	187	587.48	20	29374	214	12.26	150	32.1	26.4
SPS	133.5	419.40	21.28	19705	318.8	18.28	182	58.0	48.9

Table 8.1: Performance of the BWS with the IOPS for the Proton Synchrotron Booster (PSB) and SPS

In order to clearly define the terms precision, accuracy and resolution, Fig.8.6 combines these terms visually. Based on these definition, a position recovery system can be defined by three parameters: accuracy, repeatability and resolution. The accuracy is the difference between the real location and the recovered location. Repeatability is the variation in the recovered location between measures and the resolution is the minimum distance that the measurement system can detect.

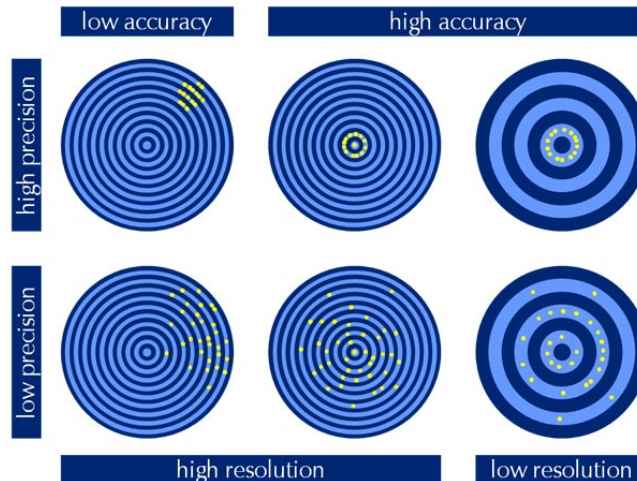


Figure 8.6: accuracy-precision-resolution [24]

8.1.1 Incremental and absolute encoding technique

The limitations of this sensor is that it does not produce the direction of the rotation nor the absolute position. The trick used to get a pseudo absolute information while the scanner is performing a measurement cycle is to use a larger mirror at a predefined position. With this reference, the mirrors counter can be reset and the position is known with an absolute reproducibility between defined positions.

To arrive to an absolute encoder, the usual approach is to create a disk drawing having multiple tracks. Each one encodes for a different piece of information at a different scale. By considering each track as a bit of information, it is possible to create a binary value which represent the absolute position, since there is only one combination of value by angular position. The drawback is that the number of track can be very large if the required resolution is high. To have an absolute resolution in the range of the absolute encoder precision, we would need sixteen different tracks 2^{16} to achieve 65000 position.

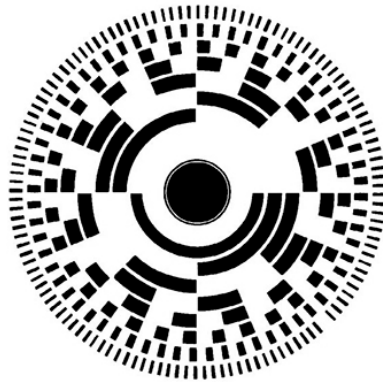


Figure 8.7: Exemple of absolut angular position disk

It is not easy to place multiple optical fibres on the current design, as the windows from where the laser light is sent to the vacuum is too small compared to the number of tracks that would require this high resolution encoder. In addition, having sixteen fibres per system would make it uneasy to operate.

A possible solution to overcome the existing limitation is to propose an absolute angular encoding embedded into a single disk track as described in the next section.

8.2 Optical disk pattern optimisation

The failure mode analysis in Chap.3 has reveal a situation where the loss of the resolver would cause a loss of its control, when the system feedback is controlling the position without movement. To prevent this situation, a redundant encoder needs to be used.

The limitation with the IOPS is that this system is unable to understand in which direction the scanner is moving. This limitation is not a problem for the primary use of this system, since the high precision is only required when the mechanism is moving. To fully take advantage of this encoder as a resolver backup, one way needs to be found to complete this system and to use it as an absolute encoder.

The proposal is to work on the disk pattern, and to embed the direction into the mirrors track. The fundamental repetition period of $20\mu\text{m}$ need to be kept for the increment, while the track needs some sort of direction encoding. One way is to play with the mirror sizes to encode the direction, which modifies the signal width and height, but without modifying the centroid of each mirror giving the increment information.

The direction will be determined by processing the reflected signal from the track and understanding if the progression of mirrors size is ascending or descending, which gives the desired information. Fig. 8.8 illustrate this ascending of descending mirror size variation (bottom) compared with a constant mirror size track (top)

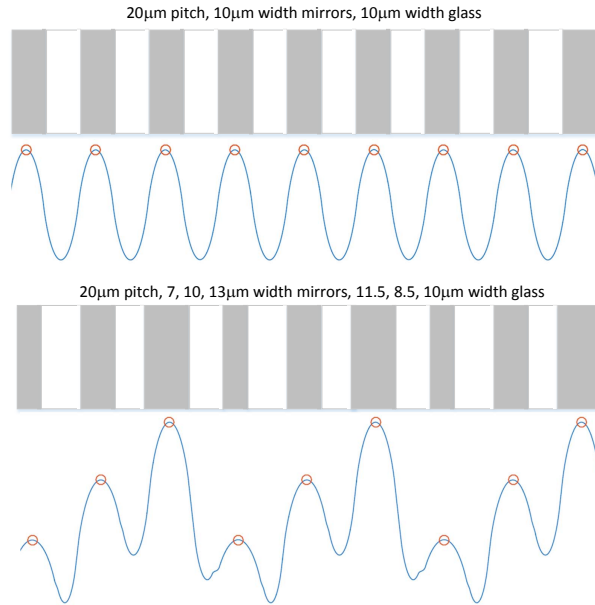


Figure 8.8: Illustration of a variable mirror sizes track having constant centroid distance between them (bottom) compared with a constant mirror width track (top)

8.2.1 Simulation of the optical process

This simulation is built by approximating the physical process of the laser beam passing on top of a track. The parts of the laser beam reflected by the 1 or 2 mirrors in front of this beam will be integrated to build the final signal. Fig.8.9 illustrates this physical process with the laser beam at different positions (top of the figure) and the resulting signal dots (bottom).

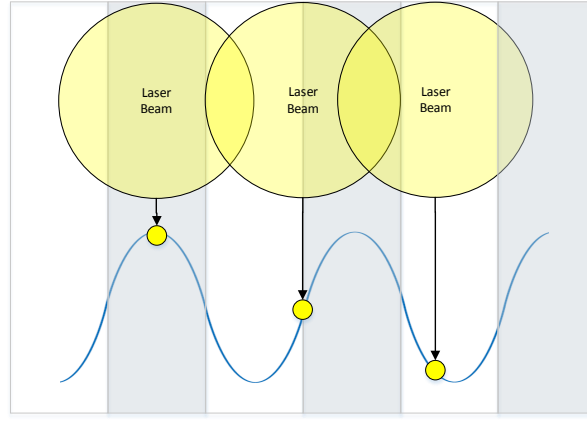


Figure 8.9: Physical process of theIOPS with the laser beam at different positions (top) and the resulting signal (bottom)

For the simulation, the laser beam is modelled by a perfect circle, which is sliced with slits size equal to the resolution defined in the simulation. The integration of the area of each slice becomes the weights of the convolution core representing the laser. Fig.8.10 shows this slicing process with a low resolution of $1\text{ }\mu\text{m}$ while Fig.8.11 shows this process for a resolution of $0.01\text{ }\mu\text{m}$. This last configuration will be used later.

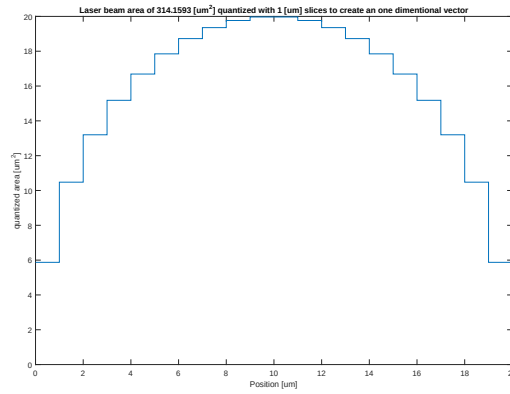


Figure 8.10: Laser beam area approximated to create one dimensional vector with $1\text{ }\mu\text{m}$ resolution

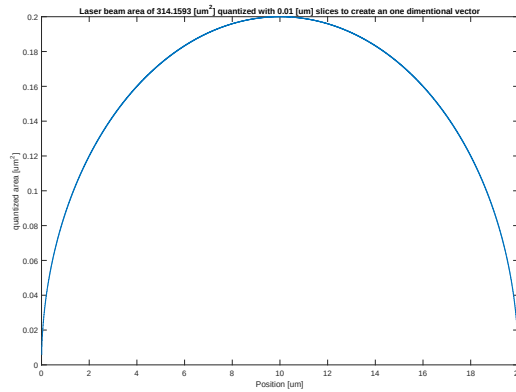


Figure 8.11: Laser beam area approximated to create one dimensional vector with $0.01\text{ }\mu\text{m}$ resolution

The track is modelled as binary list of values (1 represent 100% reflection) with the resolution equal to the one defined for the laser. The convolution products are directly the resulting signal intensity. To approach the

real signal produced by the sensor, noise and low frequency modulation is added to the convolution output. Fig.8.12 shows the mirrors track (top), the result of the convolution with the laser beam vector (center) and final sensor simulation with added noise and low frequency modulation (bottom).

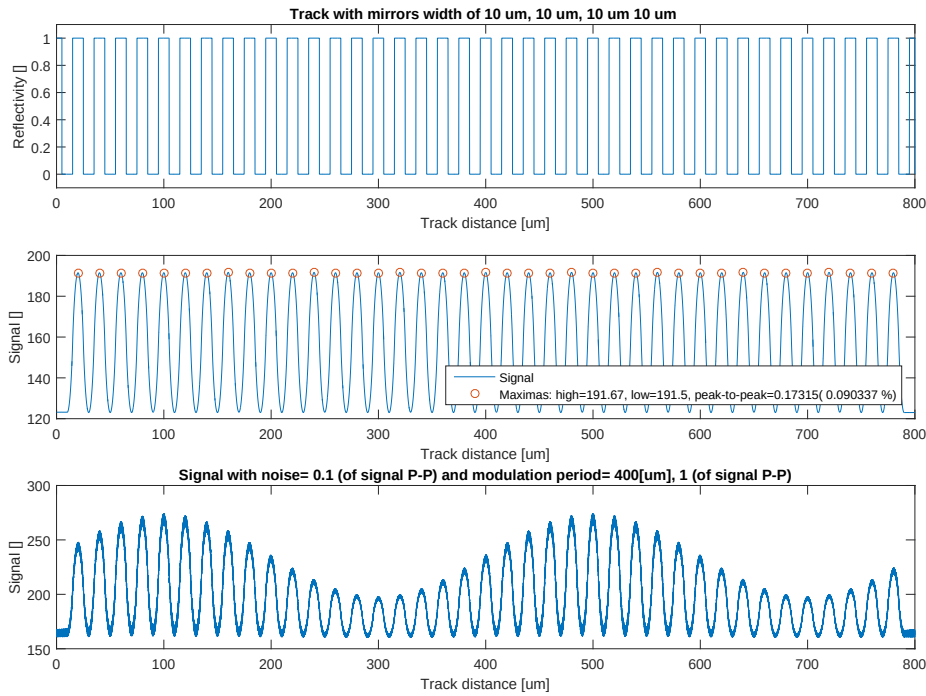


Figure 8.12: Simulation of the Incremental Optical Position Sensor (IOPS) with only one mirror size, the mirrors track (top) is convoluted with the laser beam vector giving a quasi sinusoid (center) and the final sensor simulation with added noise and low frequency modulation (bottom)

Fig.8.13 presents the diagram of this simulation where the input parameters (left) are used by the processing (centre) to create the file outputs (right).

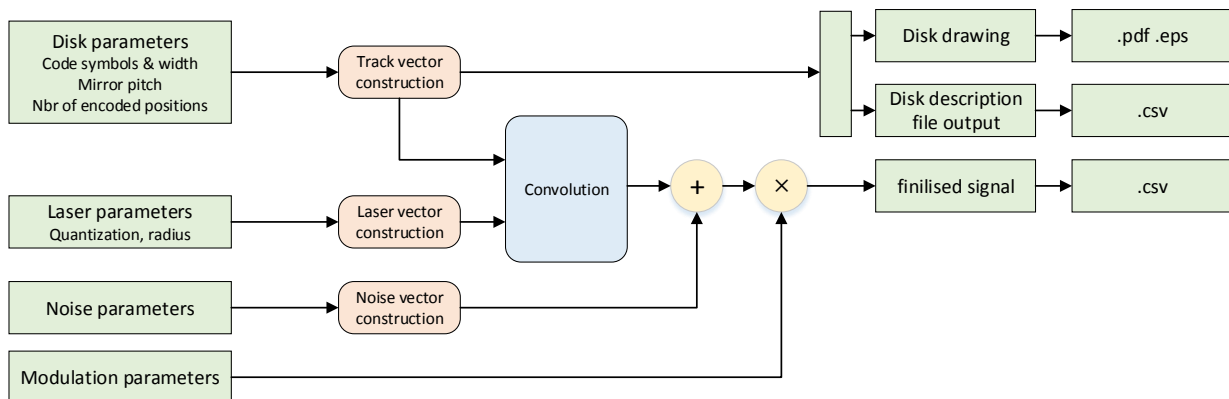


Figure 8.13: Overview of the Simulation process

8.2.2 Patterns optimisation

The parameters that can be tuned to produce the desired disk pattern are the number of symbols (S_n) and the width of each of them (S_v). The resulting value to be optimized in this process is the final signal which will be used to determine the symbol currently lit by the laser. The variation of the mirror width induce a change in the width of the signal, but also a difference in its maximal amplitude. Since this last characteristic is simpler to measure for the future signal processing system, the optimization will focus on improving the difference between the maximal amplitude of each symbol. A second parameter need to be verified during this optimisation. The minima of the signal period is determined by part of two adjacent mirrors, meaning that the interaction of mirrors of different sizes need to be verified.

Fig.8.14 illustrate the variation of the minima and maxima of the signal when the track has three different mirror sizes while having a constant pitch. A version with four mirror sizes is illustrated in Fig.8.15.

The optimisation process used here is the simulation of multiple cases, a sort of trial and error. An analytical optimum would surely exist, but its implementation would require much more efforts going against other parts of this thesis.

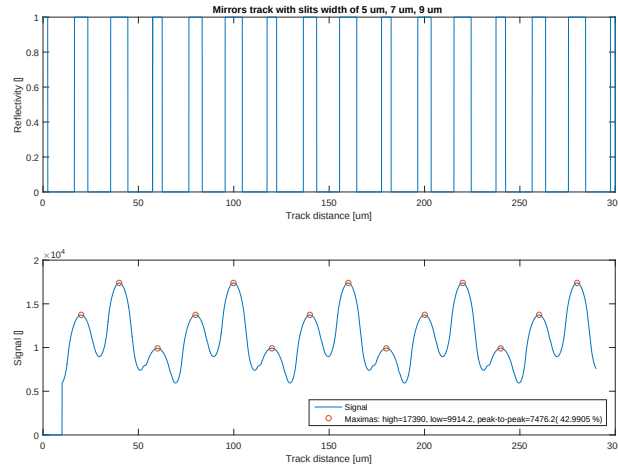


Figure 8.14: Simulation of the IOPS with three mirror sizes to determin the rotation direction

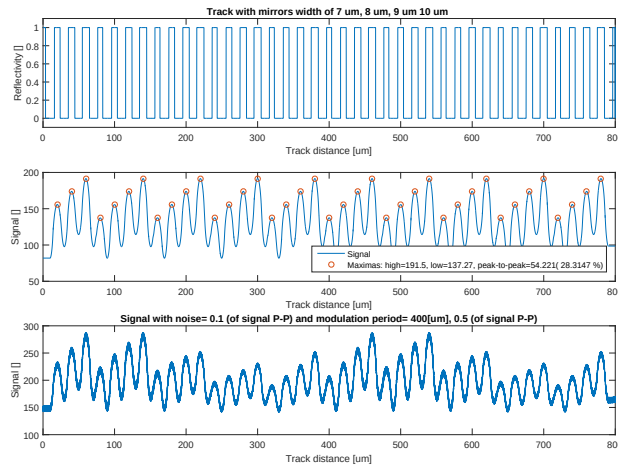


Figure 8.15: Simulation of the incremental optical position system with four mirror sizes

8.2.3 Absolute angular position encoding into the track

To complete the concept, it is even possible to imagine an encoding of the absolute angular position into the track. This encoding need to be independent of the rotation direction while not perturbing too much the direction information itself.

The current implementation of the disk has 29374 mirrors (see Table.8.1) for one full revolution. To provide at least an absolute angular information each degree, the number of mirrors available would be $29374/360 = 81$.

The simplest proposal is to encode each decimal digit rank with a predefined mirror size. When the absolute position would have to be written into the track, the usual mirrors sequence would be modified with the value of the position.

The following examples are taking into account mirrors direction patterns having three size, 7, 9 and 11 μm . This pattern sequence encoding for the rotation direction can be written **-7-9-11-**. In case of encoding the absolute angular position, the size 7 encodes the least significant digit, 9 the tens digit and the 11 the hundreds.

When no absolute position is encoded, the sequence is repeated continuously, **-7-9-11-7-9-11-7-9-11-**. When an absolute position needs to be inserted in the track at a defined position, the next sequence will be modified as follow. The first symbol **-7-** will be repeated as many times as the value of the digit + 1 (to encode the 0), then the next symbol **-9-** and finally the last one **-11-**. If digit to encode are 0, the symbol is only repeated ones.

This method insure that there is no more that 11 times the same symbol in the sequence, and thus determines the longest angular position without rotation direction. this means:

$$11 * 214[\mu\text{rad}] = 2,354[\text{mrad}] = 0.1349[\text{deg}] \quad (8.1)$$

Table 8.2 provides some example of decimal values encoded with the proposed scheme of this thesis. The largest number possible to encode is the '999', which will produce a sequence of thirty-three symbols, the longest with this scheme.

Table 8.2: Absolute position encoding examples

Decimal value to encode	symbol sequence
001	-7-7-7-9-9-11-11-
010	-7-7-9-9-9-11-11-
100	-7-7-9-9-11-11-11-
111	-7-7-7-9-9-9-11-11-11-
444	-7-7-7-7-7-7-9-9-9-9-9-11-11-11-11-11-11-
999	-7-7-7-7-7-7-7-7-7-9-9-9-9-9-9-9-9-9-9-11-11-11-11-11-11-11-11-11-11-

To create a track with this particular encoding, a script was created which produce the direction sequences and insert angular position encoding at predefined location in the sequence. Fig.8.16 illustrate the track created. It shows the reflection coefficient of the track in function of the position (in the unit defined by the simulation resolution). The lower left figure is the encoded number '42' with the mirror sequence of -7-7-7-7-9-9-9-9-9-11-11- while the lower right figure is '296' with -7-7-7-7-7-7-7-7-9-9-9-9-9-9-9-9-9-2-2-2-2.

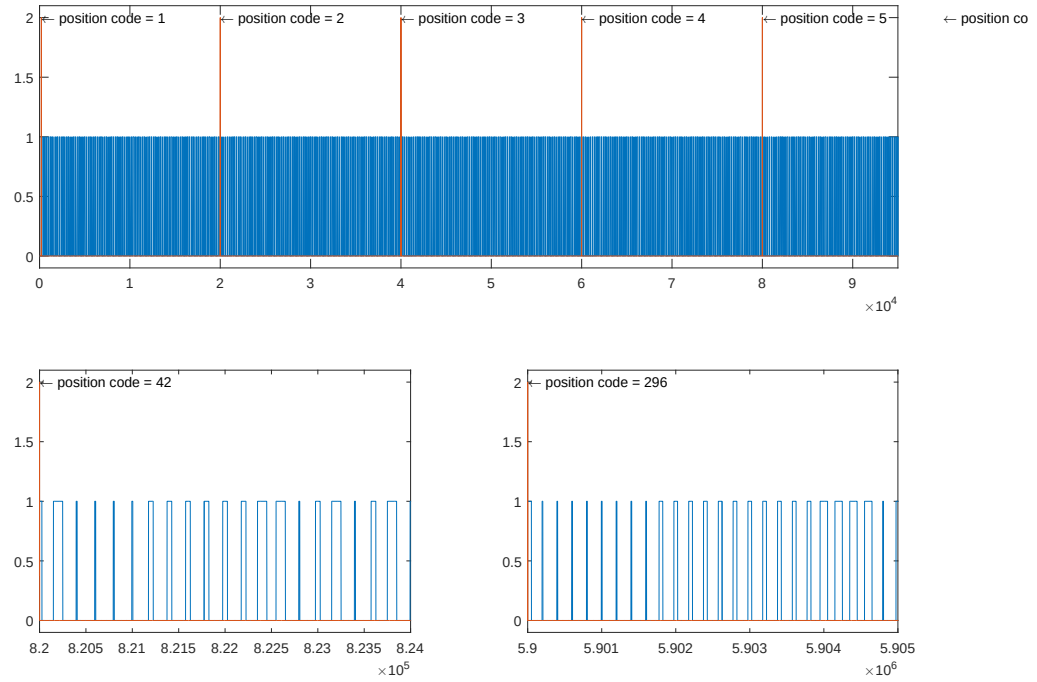
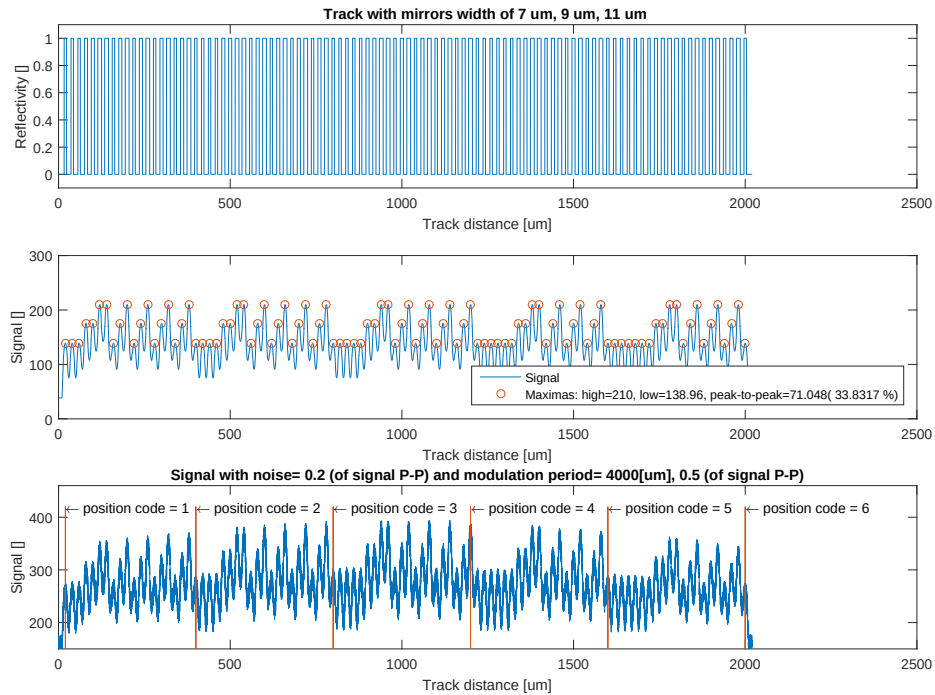


Figure 8.16: Illustration of a disk track having the position embedded in its mirror width

The created track is the used as input to the simulation of the physical process of the IOPS sensor. Fig.8.17 shows the track disk (top) used as input to the simulator giving an optimal output (centre) and one perturbed by noise and low frequency amplitude modulation (bottom).


 Figure 8.17: Simulation of the track with the position encoded with 7, 8 and 11 μm mirrors

8.2.4 Finalisation of the simulator for the detection algorithm development

There is one last physical process which had to be integrated in the simulator to be very close to the original signal coming out of the IOPS. This is the effect of the disk motion during the rotation of the scanner. The reference profiles applied on the real system is visible in Fig.8.19 with the position (top) and speed (second from the top).

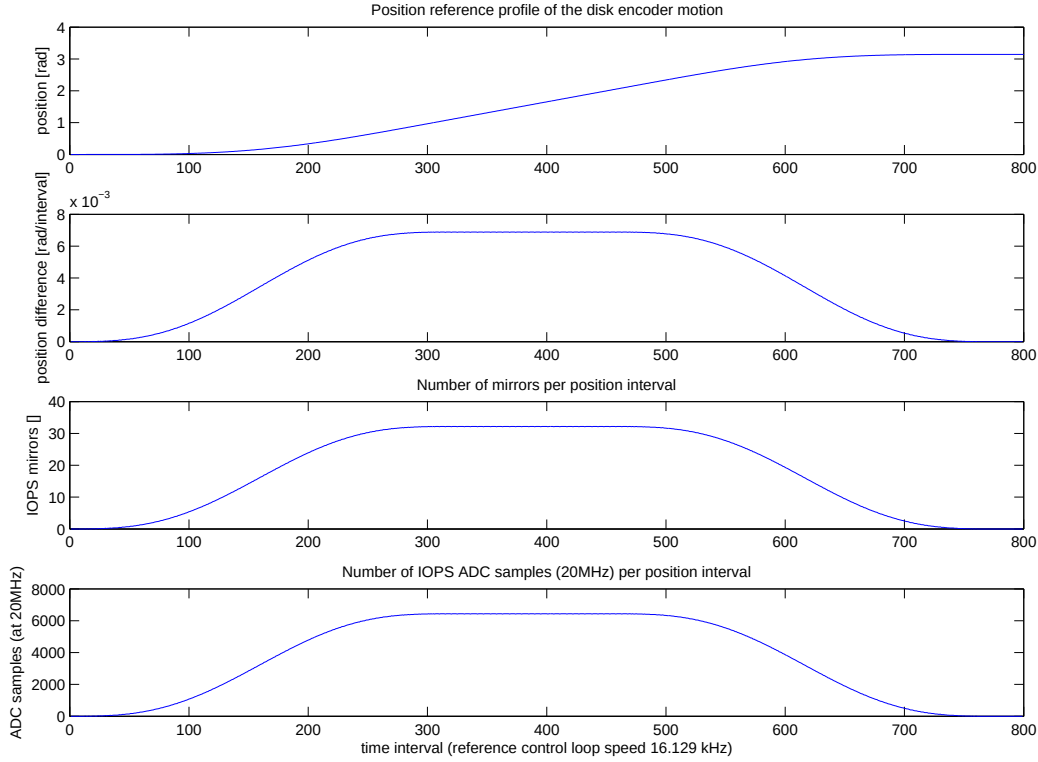


Figure 8.18: Motion profile of the IOPS disk used for calculating the final simulation of the 20MHz digitalisation of the track

Until now the simulator was providing the output signal in function of the position of the track, as visible on the top of Fig.8.18, while the real signal is given in function of the time with a given sampling period. To make this, the speed profile was used to convert the position axis into time. The result is visible in the middle plot of Fig.8.18. The final step was to quantify the signal to provide a vector with pseudo ADC reading visible on the bottom of Fig.8.18.

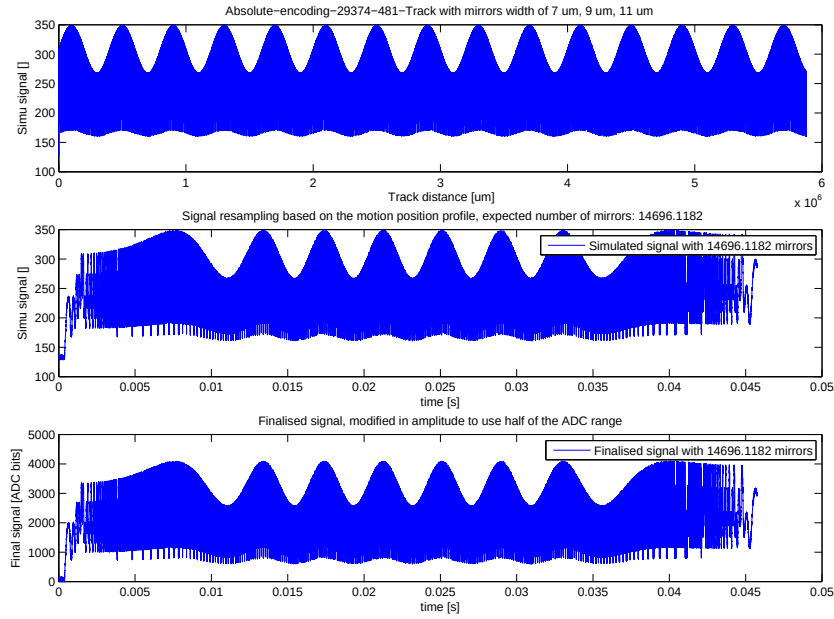


Figure 8.19: Finalised signal with a given position profile (central plot) and amplitude adaptation as the IOPS ADC input (bottom plot)

Fig.8.20 shows the signal resulting from the disk embedding the absolute position analysed by the script later used for the real field data sets in the subsection 8.3.

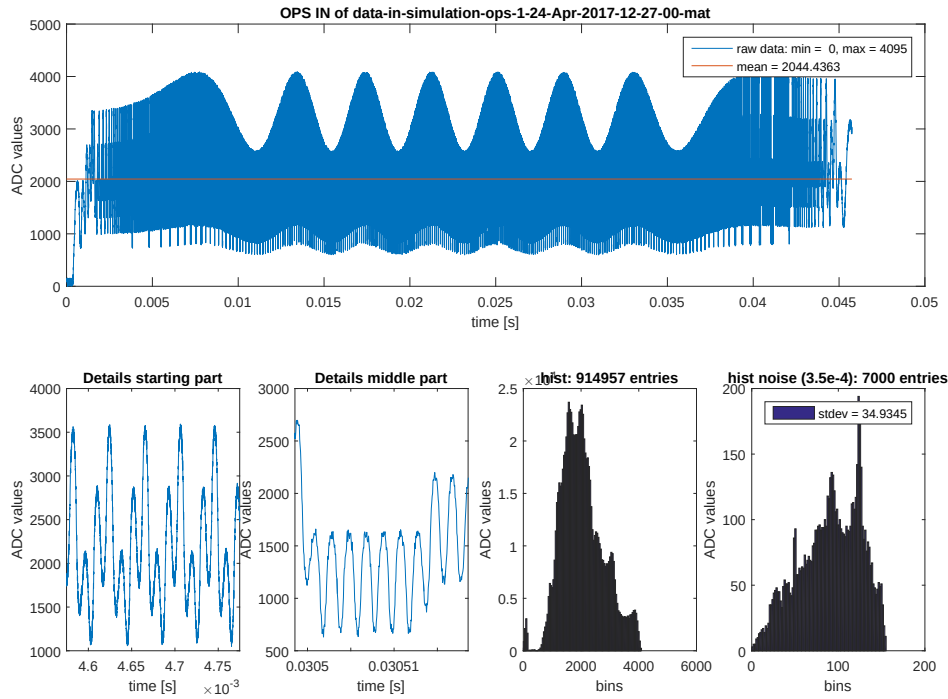


Figure 8.20: Analysis of the signal resulting from the absolute disk encoding simulation

8.2.5 Disk mask generation for the photolithography process

The last step needed prior to the manufacturing of this disk is to build this particular track pattern onto a disk. For this purpose, a disk automated mask drawer has been added to the script to output a figure with all the mirrors. A test file is also generated with comma separated value representing the coordinate of each mirror, in a vectorised fashion.

The approximation selected for drawing the mirrors is represented in Fig.8.21. Each mirror is created with a Trapezoid using simple geometric formula.

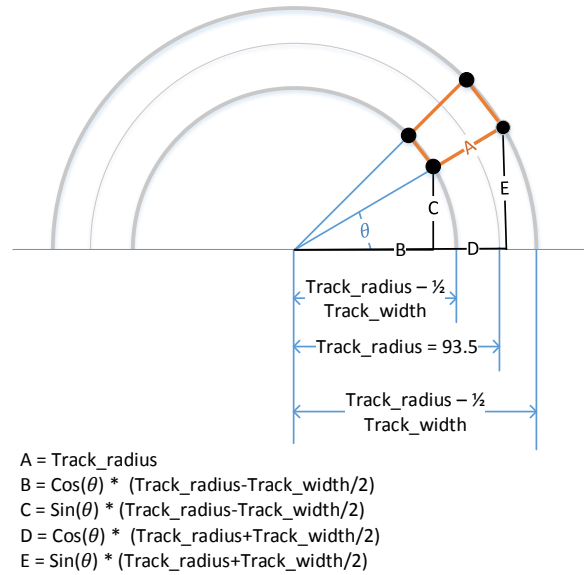


Figure 8.21: Mirror geometry approximation

Fig.8.22 shows the resulting disk map almost ready to be manufactured. The mirrors are not visible as they are very small compared to the disk diameter. Fig.8.23 gives an idea of the type of mirrors generated by the script.

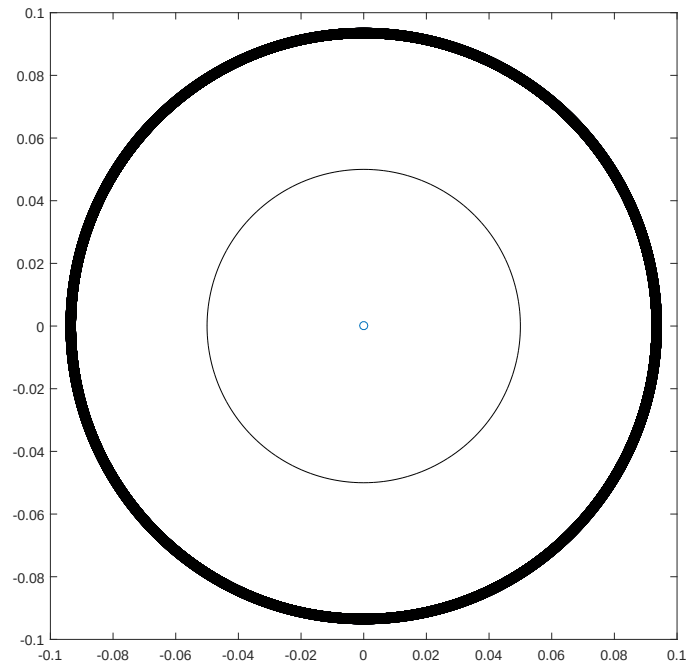


Figure 8.22: Disk mask for track photolithography

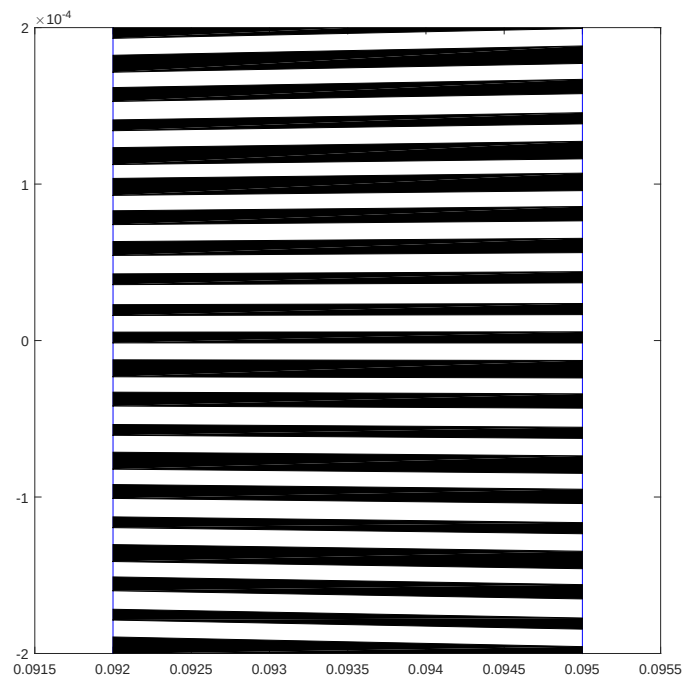


Figure 8.23: Zoom on the disk mask for track photolithography

8.3 field data characterisation

The first step for building the data processing id to develop a mechanism to look at the data sets and to characterise the available measurements. This sensor provides two independent channels looking at the same disk, but at different position. Fig.8.24 shows these two independent channels, channel one on top and two at the bottom, for an "IN" movement, representing a rotation of π rad. The signal is a periodic, quasi sinusoidal which is modulated with a low frequency. this is a perturbation due to various instabilities such as disk vibration, mechanical play or even laser interferences due to the reflected light coupled back to the same emission fibre.

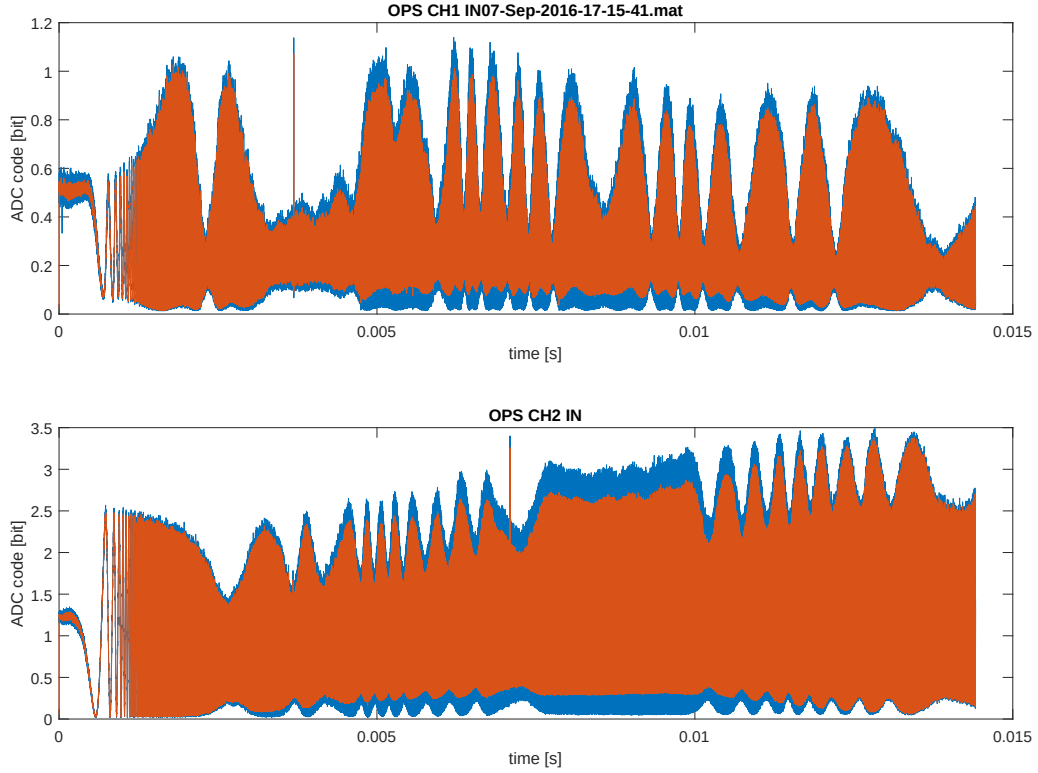


Figure 8.24: IOPS signals for a displacement π rad on the 7th September 2016 at 17h15m41s

An analysis script was written to check the measurements data sets and to extract some characteristics. Fig.8.25 shows the IOPS channel 1 data set (top) and its characteristics such as minimum, maximum and mean value as well as plotting the histogram (bottom third plot) of the complete data set. The noise estimation is performed by plotting the histogram of the data set recorded before the movement start, at the very beginning (bottom right) giving a standard deviation of about 95 [bit] for this particular example. Fig.8.26 shows similar data for channel two. One figure per record has been generated to get an idea on this sensor output.

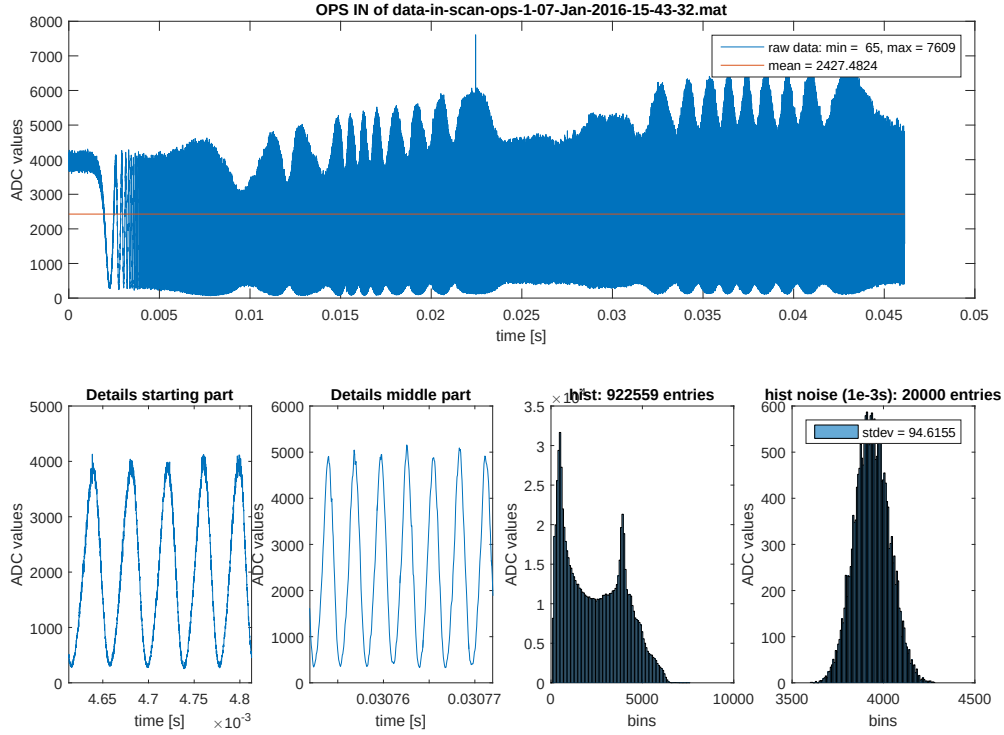


Figure 8.25: Characterisation of the IOPS channel 1 signal on the 7th January 2016 at 15h43m32s

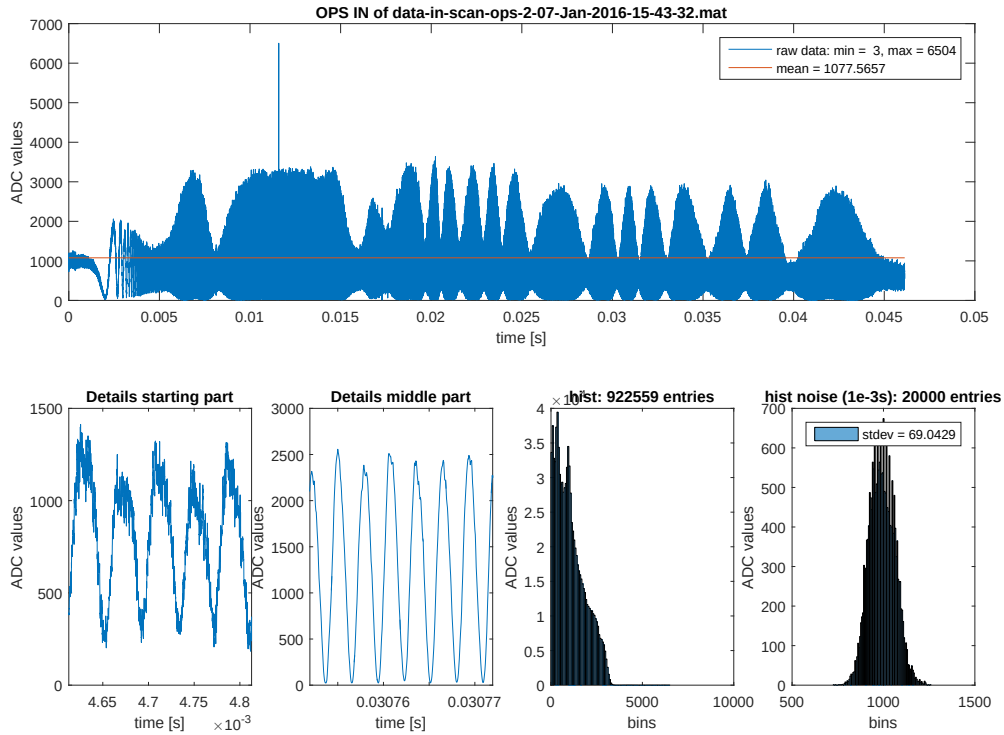


Figure 8.26: Characterisation of the IOPS channel 2 signal on the 7th January 2016 at 15h43m32s

8.4 Processing algorithm prototyping

the aim of the prototyping phase was to find an efficient and elegant way to extract the position and speed from the IOPS signals. The position information is given by the integrated number of mirrors crossed by the laser beam when the shaft rotates while the speed information is the number of mirror detected for a defined time. the speed can also be found by extracting the time between the detection of two mirrors.

The key challenge of this processing is to detect correctly the period of the signal having a very large range of frequency, from around 1[kHz] to 400[kHz] with an unstable amplitude and offset. Several prototypes where realised using different ways to detect these mirrors periods:

- Unique threshold based decision
- Adaptive threshold based decision
- Derivative based detection
- Pattern based recognition
- Determination based on Adaptive Frequency Tracking

Each of these methods have a dedicated subsection where the algorithm is explained with an example of processed data to illustrate it. Subsection [8.4.5](#) summaries the results of each method applied the multiple data sets and a short analysis conclude this part by selecting one method to be implemented in VHDL.

8.4.1 fixed threshold based decision

The simplest method possible to detect a periodic signal is to determine the beginning of a period by comparing its value against a predetermined threshold. With Matlab this operation is very easily done as we have in hand the complete data set, which doesn't apply for the real-time processing foreseen. The mean value of the whole set is used to define two levels around it. Fig.8.27 presents the result for one data set having a stable signal amplitude (top). The position (bottom left) and the speed (bottom right) is nicely resolved. Fig.8.28 shows the distribution of the mirrors period expressed in frequency. We can see that almost 6000 mirrors are seen by the system at a frequency higher 300[kHz].

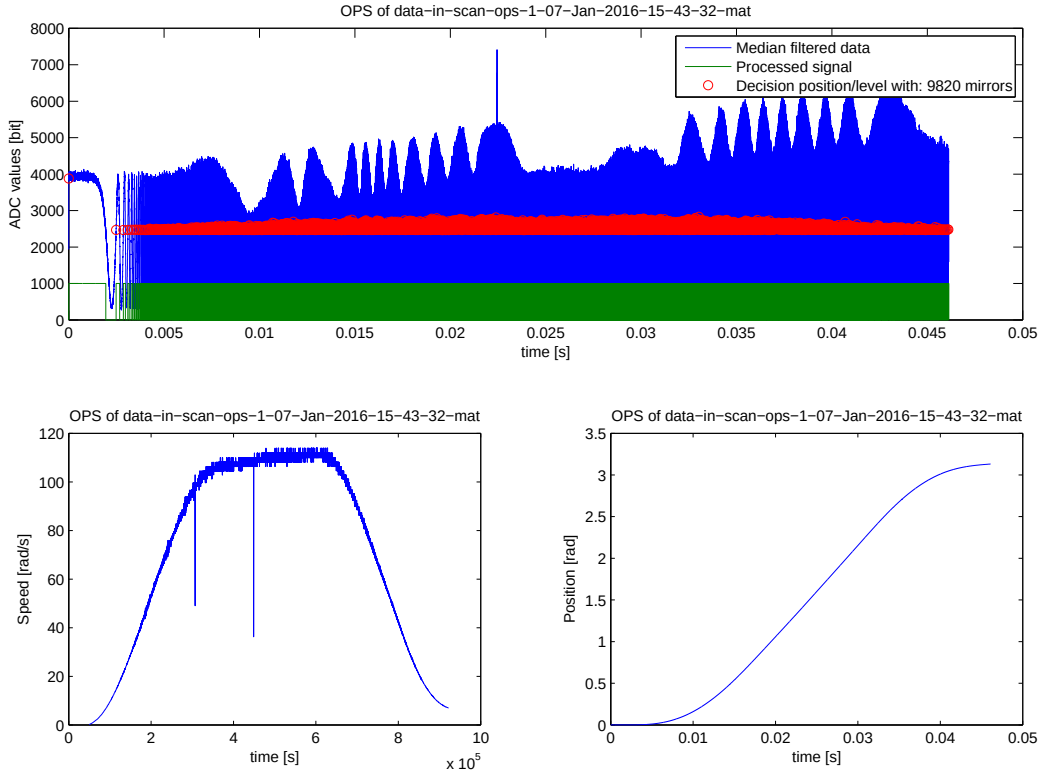


Figure 8.27: IOPS signal processed with a fixed threshold based decision on an optimal data set recorded on the 7th January 2016 at 15h43m32s

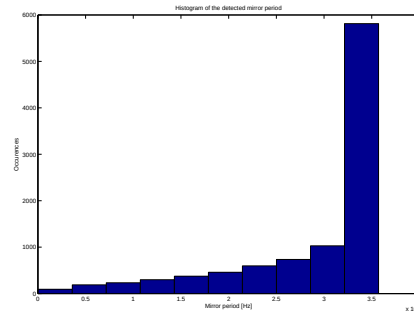


Figure 8.28: Distribution of the mirrors period found using the fixed threshold based decision algorithm on the data set recorded on the 7th January 2016 at 15h43m32s

The use of a fixed threshold technique is not without drawbacks. The determination of the threshold prior to the measure for the real-time system is a challenge. Issues arise when dealing with data sets having large amplitude variation as the example of Fig.8.29 .

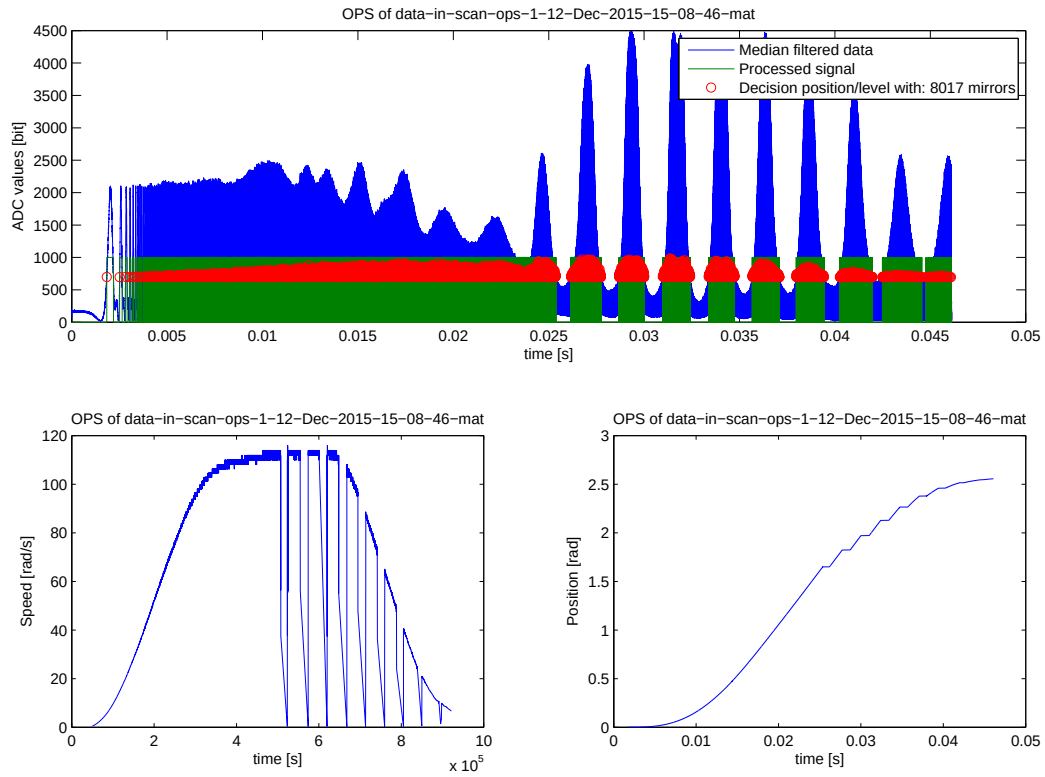


Figure 8.29: Exemple on a none optimal data set recorded on the 12th December 2015 at 15m08m46s processed with the fixed threshold based decision algorithm

8.4.2 adaptive threshold based decision

An evolution of the fixed threshold algorithm is the adaptive version where the level of the threshold is adjusted in real time depending on previous mirror periods. The mean value of the current period is filtered with the two previous period to provide the threshold level for the next period. This filter extracts the median of the three in order to avoid eventual large variation of this period due to the reference mark lasting longer than normal mirror or disk scratches.

The result is a threshold which evolves with the incoming data making it suitable for real time processing. Fig.8.30 and Fig.8.31 presents one data set processed with this algorithm. The decisions levels are varying while the time passes to fit the evolution of the signal amplitude and offset.

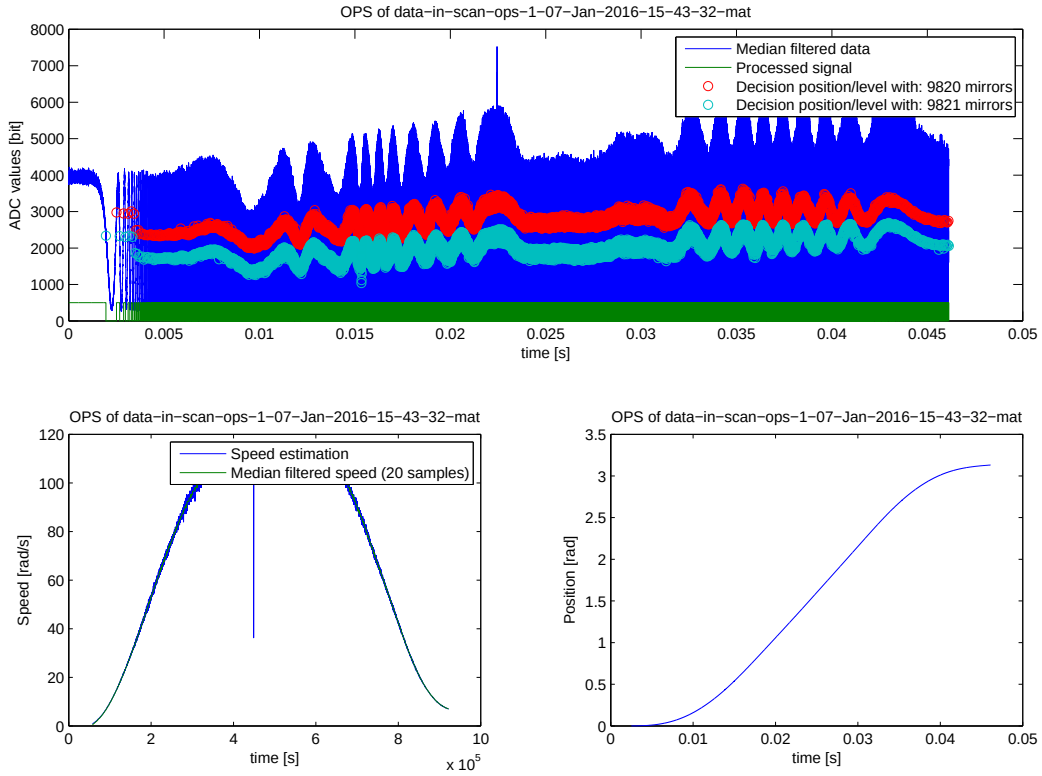


Figure 8.30: IOPS signal detection using adaptive threshold on the channel 1 of the data set recorded on the 7th January 2016 at 15h43m32s

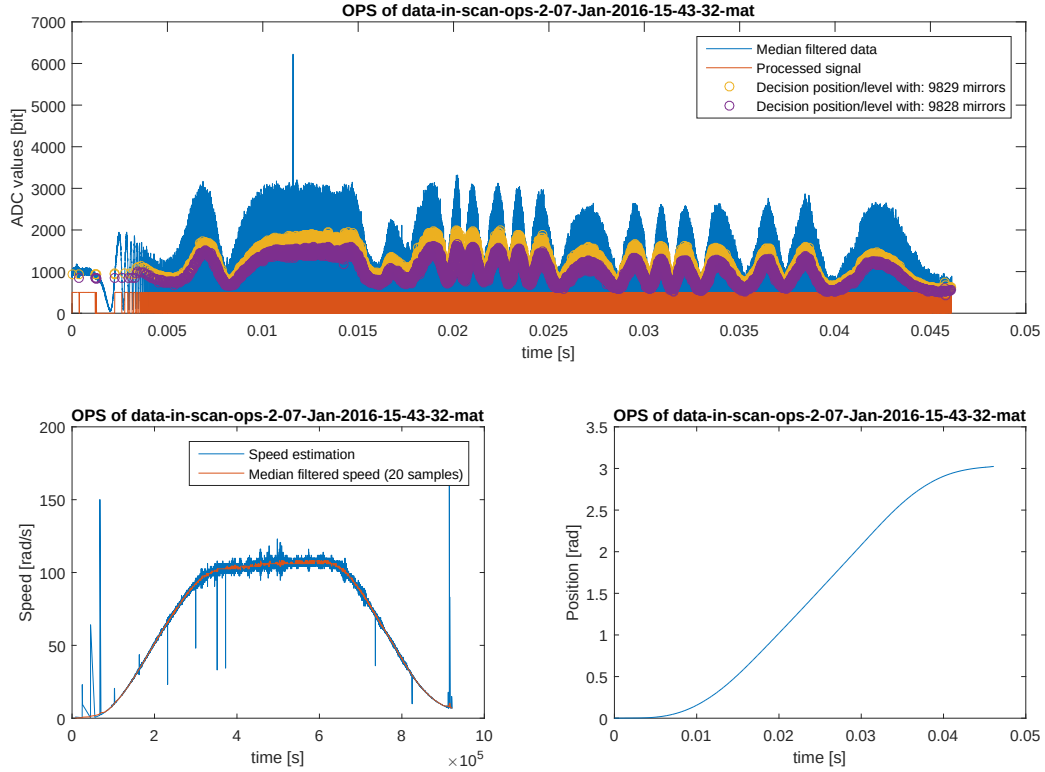


Figure 8.31: IOPS signal detection using adaptive threshold on channel 2 of the data set recorded on the 7th January 2016 at 15h43m32s

8.4.3 Adaptive detection of the maxima

Another technique was investigated to detect local minima or maxima of the signal without a threshold. the principle is that this algorithm is trying to find a predetermined curve having a defined height.

For this, the algorithm is looking at the data using a variable window. At each incoming sample the window data is shifted by one sample and the oldest is removed. At this moment, if the first and last value within the window are close enough and the central sample is lower or higher than the edges by a determined value, the algorithm define that the central point is a minima or a maxima. When more than two maxima are found, the window is adapted to the period of the signal. Fig.8.32 and Fig.8.32 are showing two test data set processed with this algorithm. The minima and maxima recorded are shown on the top plot, the window size evolution is shown on the middle plot while the resulting speed is at the bottom.

This processing scheme is more complex than the previous one using a simple threshold, fixed or adaptive. It uses two fixed thresholds to be manually tune plus one adaptive window, making it more likely to fail on unexpected data. A major evolution is proposed in the next subsection which try to overcome these limitations.

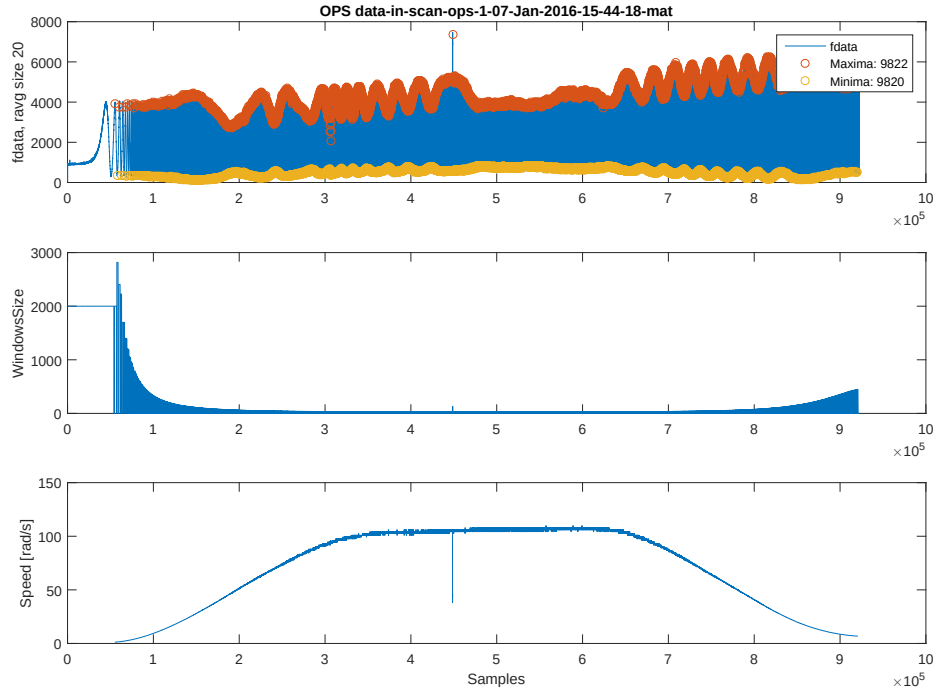


Figure 8.32: IOPS signal detection using minima and maxima pattern detection and adaptative windowing

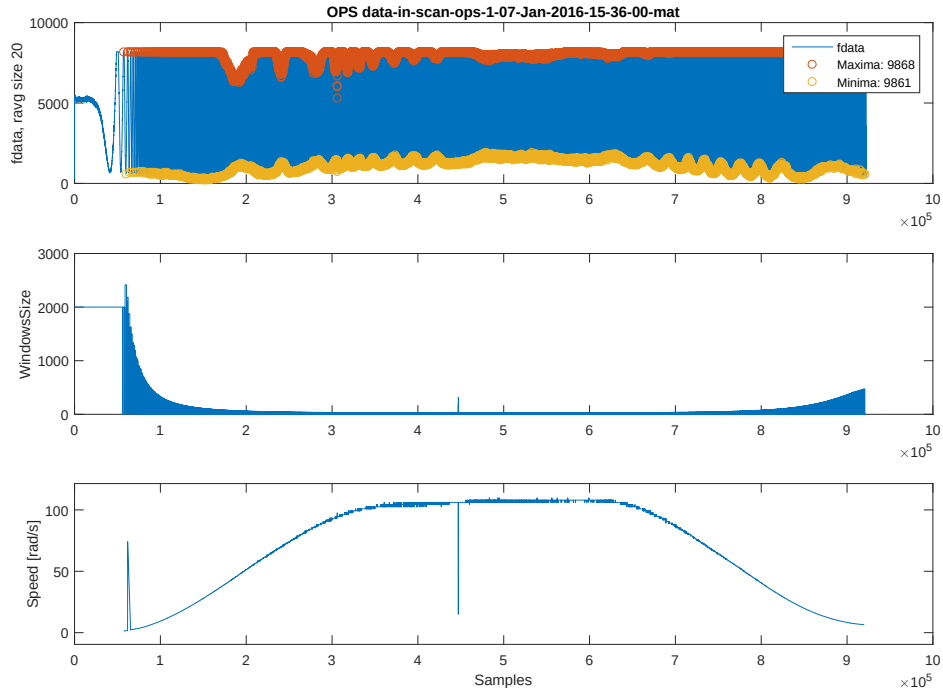


Figure 8.33: IOPS signal detection using minima and maxima pattern detection and adaptative windowing

8.4.4 Derivative based variable period detection

The last algorithm that was prototyped is based on the idea to look at local minima and maxima rather than looking at the value itself.

This technique is potentially able to react much better to unexpected variation of the signal amplitude, as it looks at the slope of the signal.

The first step in the algorithm is to calculate the derivative of the incoming data. This is done by using a convolution filter to perform a Savitzky Golay filter [23] as previously used from the resolver in section 7.4. By selecting the appropriate coefficients, it can calculate the first derivative of the signal and smoothing it already. The result is also a periodic quasi sinusoidal signal which is now centred around zero and its amplitude reveal the signal slope. The higher the signal is, the higher the slope of the original signal is.

The next step is to clean this derivative, such process is picking any noise easily. A median filter is applied to remove large diverging values. Then signal compared again 0 to create a binary vector. If the signal is below zero, the value is 0 and if above it becomes one.

The final step is to detect the rising edge of the binary vector in order to record the beginning of a period.

Fig.8.34 presents various view of an data set processed with this algorithm. The original data set (top) is marked with the detected period using this algorithm (in red) and using the Adaptative Threshold algorithm (in yellow and violet). The second line is showing various part of this set. the third line is the derivative and binary signal. The last line is showing various part where we can better visualise the derivative and the binary signal together.

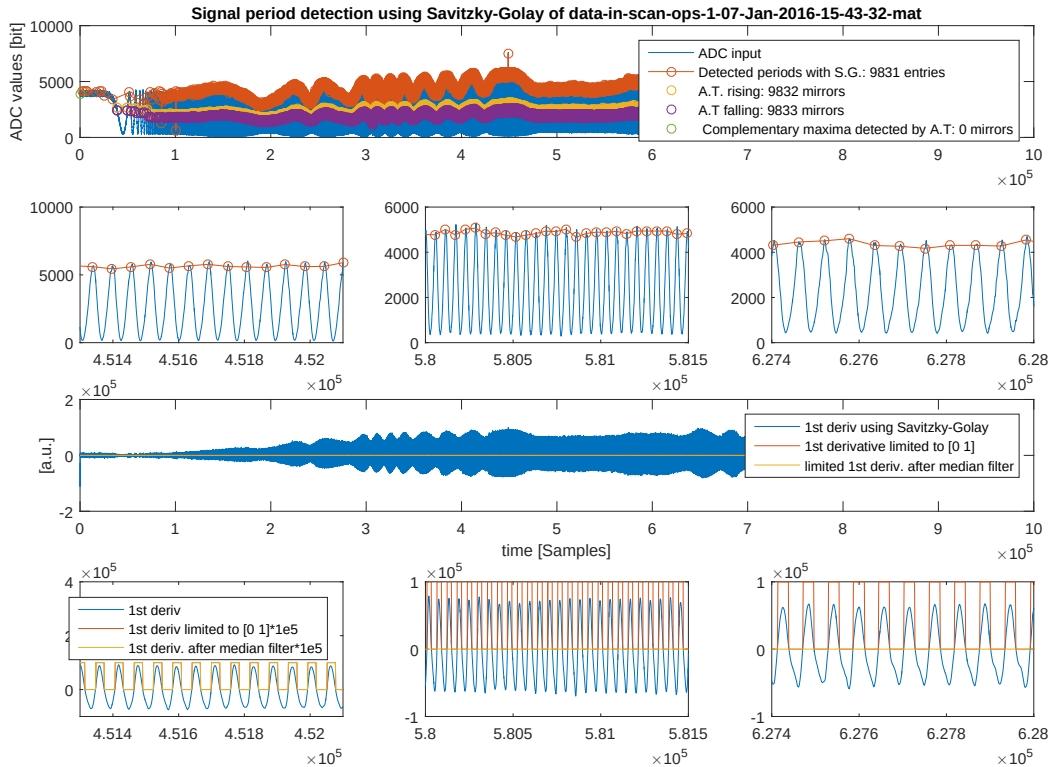


Figure 8.34: Mirror detection using the deivative algorithm on the IOPS channel 1 data set recorded on the 7th January 2016 at 15h43m32s

Fig.8.35 shows the recovered position and speed using this algorithm on channel 1 of the data set recorded on the 7th January 2016 at 15h43m32s.

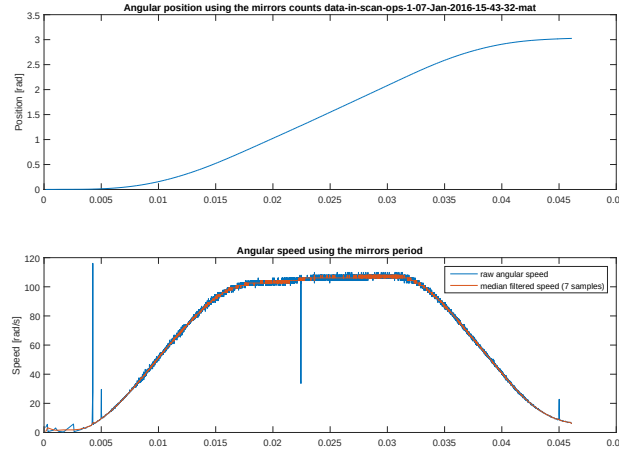


Figure 8.35: Position and speed recovered by the deivative algorithm on the IOPS channel 1 data set recorded on the 7th January 2016 at 15h43m32s

Fig.8.36 and Fig.8.37 are showing the same processing with a different data set. In this second example, the derivative and the adaptive threshold algorithms are still able to recover most of the mirrors and are giving similar results.

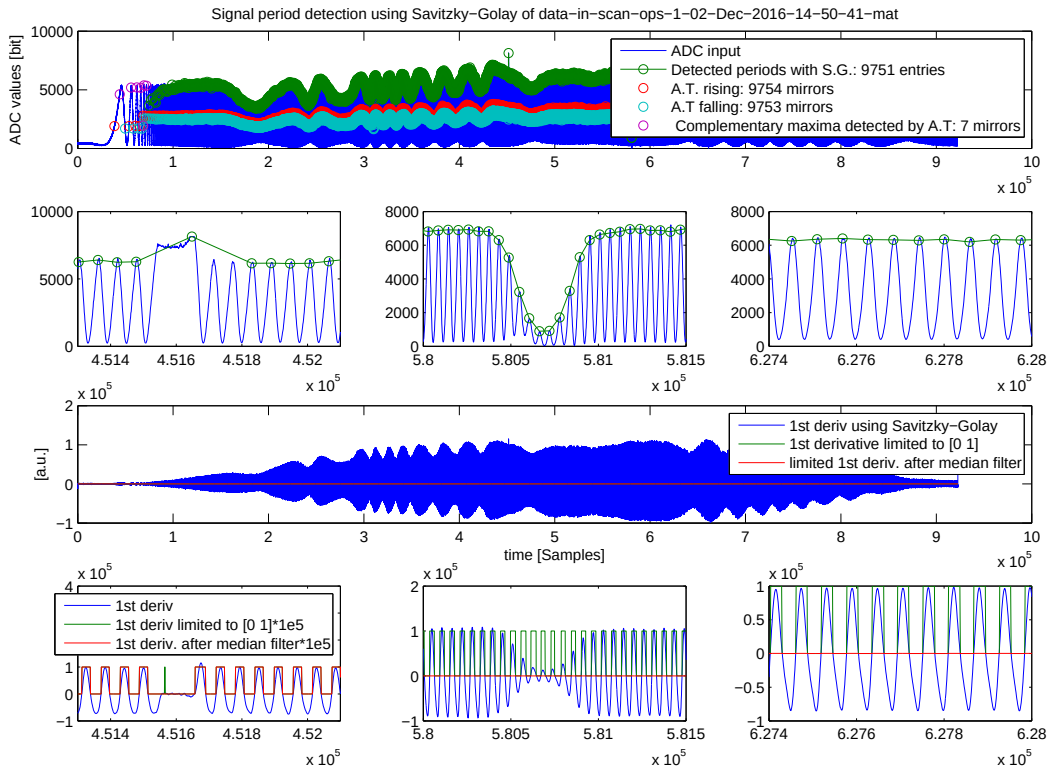


Figure 8.36: Mirror detection using the deivative algorithm on the IOPS channel 1 data set recorded on the 2nd December 2016 at 14h50m41s

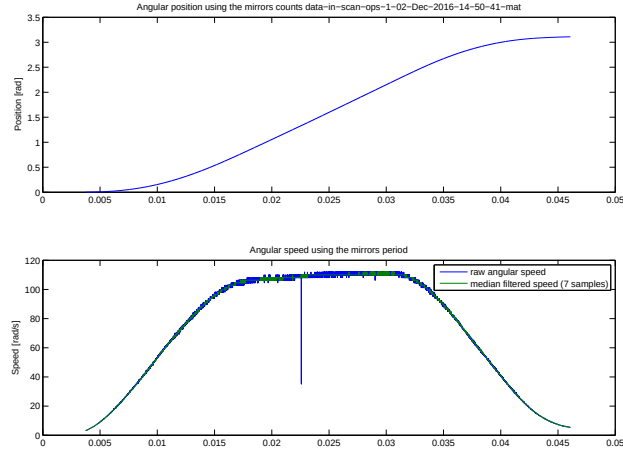


Figure 8.37: Position and speed recovered by the derivative algorithm on the IOPS channel 1 data set recorded on the 2nd December 2016 at 14h50m41s

The last example shown on Fig.8.39 and Fig.8.39 reveals the power of this algorithm. It is the only one which is able to adapt itself to very fast amplitude variations, due to the position and direction encoded onto the mirrors track. The data set given by the IOPS absolute disk simulation developed in section 8.2 has a total number of mirrors of 14696, due to the increased radius of the PSB disk compared to the SPS. The adaptive threshold can't follow this amplitude variation, on a period per period bases, and loses 27% of the mirrors. The derivative is not affected by this effect, as it looks at zero crossing of the slope (maxima), and loses only 0.005% of the mirrors.

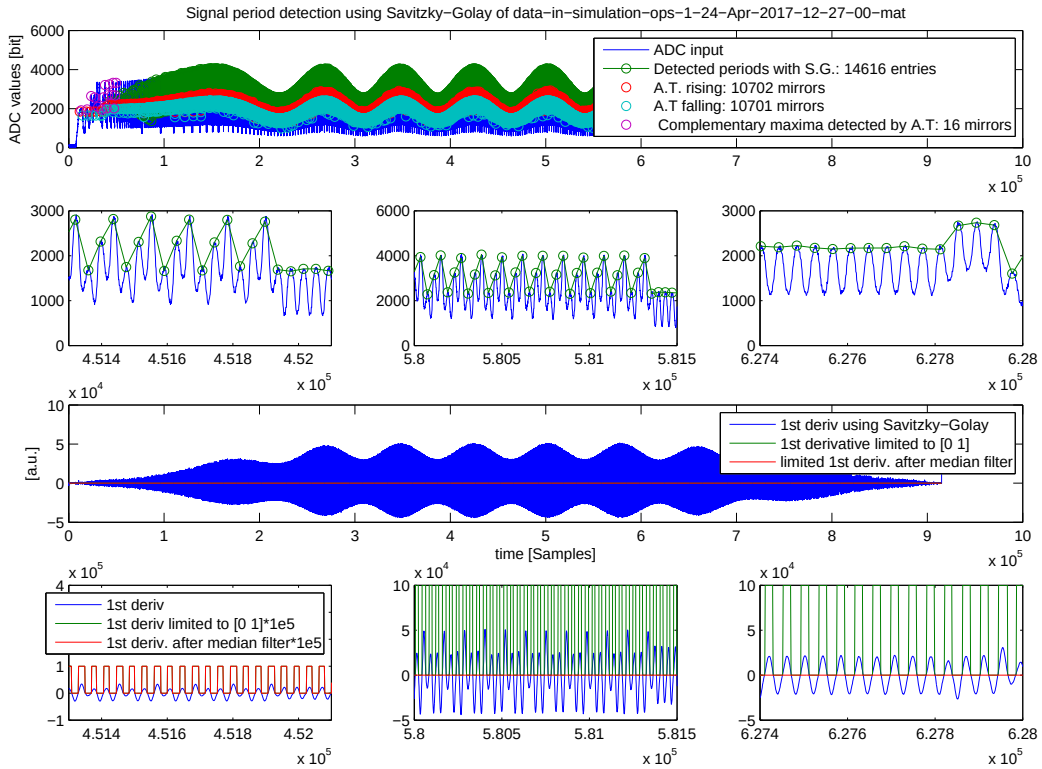


Figure 8.38: SG derivative method applied to the simulation of absolute track developed in this thesis. Expected nbr of mirrors: 14696

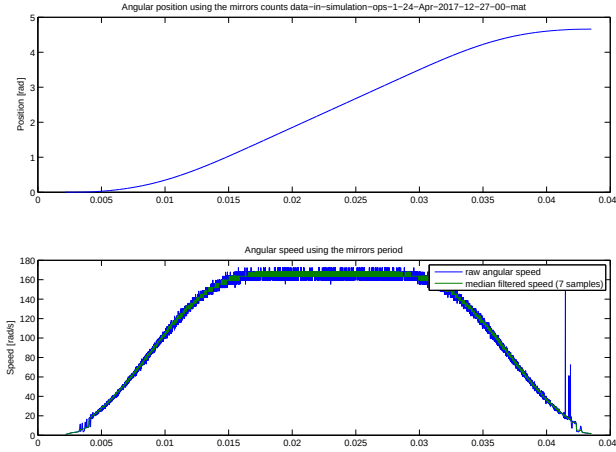


Figure 8.39: Motion detection using the absolute track developed in this thesis

8.4.5 Summary of the algorithms development

The four algorithm prototyped for processing the IOPS have been fed with the same data sets for comparing their results. Table 8.3 summarises the number of detected mirrors for ten data sets with the four algorithms. The data sets coming from the same measure, but made with a different channel are ideal to compare as the number of mirror is expected to be very similar. Additional lines are made to calculate this difference in mirror and percent. The lower is this value the better the algorithm should be able to resolve the position. For each line, the mean of the two central values between the four is calculated under the "cval" and the worst case deviation from this value is calculated under the "mdev". This last column provides the information on the ability of the algorithms to find the same number of mirrors, meaning the data set is simple to process.

Table 8.3: Results of the IOPS fields measurements processed with the algorithms developed earlier

Data set date	CH	Dir	Threshold	Adaptive th.	Adapt. Maxima	S.G.	cval*	mdev*
02-Dec-2016-14-50-41	1	in	9754	9751	9759	9751	9753.8	5.3
02-Dec-2016-14-50-41	2	in	5212	6790	153	5385	4385.0	4232.0
Difference abs(ch1 - ch2)			4542 (46%)	2961 (30%)	9606 (98%)	4366 (44%)		
02-Dec-2016-14-50-42	1	out	9627	9624	9633	9640	9631.0	9.0
07-Jan-2016-15-43-32	1	in	9820	9820	9820	9831	9822.8	8.3
07-Jan-2016-15-43-32	2	in	8920	9829	6762	9703	8803.5	2041.5
Difference abs(ch1 - ch2)			900 (9%)	9 (0%)	3058 (31%)	128 (1%)		
07-Jan-2016-15-43-32	1	out	9667	9669	9669	9708	9678.3	29.8
12-Dec-2015-15-08-46	1	in	8017	9951	17	10911	7224.0	7207.0
12-Dec-2015-15-08-46	2	in	9607	9940	972	9860	7594.8	6622.8
Difference abs(ch1 - ch2)			1590 (19%)	11 (0%)	955 (5617%)	1051 (9%)		
12-Dec-2015-15-08-46	1	out	7612	8757	348	8681	6349.5	6001.5
24-Apr-2017-12-27-00	1	in	8615	9041	642	14616	8228.5	7586.5

*cval: mean of the two central values (by excluding the min and max value), *mdev: maximum deviation from cval,

**Simulated signal for absolute track encoding, PSB disk, nbr mirrors: 14696

The adaptive threshold algorithms presents the best results for the observed datasets, when excluding the special case of the disk with embedded direction and position.

The derivative based variable period detection algorithm is a serious candidate for the processing of the next generation of Incremental Optical Position Sensor (IOPS) disk embedding the direction and position. The increase in complexity is compensated by its unique ability to resolve complex situations.

In the scope of this thesis, the adaptive threshold algorithm will be implemented in VHDL.

8.5 VHDL implementation

The processing of the IOPS is based on the adaptive threshold described in subsection 8.4.2. It has been written as a self contained entity as defined in Chap.6. Fig.8.40 show the entity with its inputs and outputs

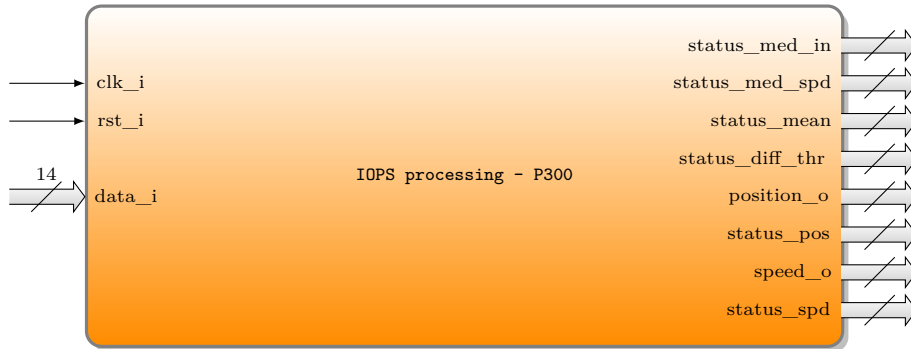


Figure 8.40: The IOPS processing component

The conversion of the prototype algorithm written in MatLab to the VHDL description was performed manually according to the defined procedure in 4.2. the following paragraph describing the implementation comes from the data processing overview chapter in 6

The IOPS data flow is presented on Fig.8.41. The incoming data is filtered using a median filter and the difference between input and output is monitored. Then the processing is extracting the position information from the data. This is done by comparing the value against a calculated threshold to determine the length of the signal period, and the system angular position (P340). The mean of the signal and the system speed is calculated by the module P350. The thresholds are calculated at every new period by the module P360, using multiple signal periods to obtain smooth values. The calculated speed is filtered to exclude eventual peaks. All the resulting quantities are monitored against predefined thresholds and the status are send further.

The following components are described in details in Chap.12: the median filter (P320 and P370), the statistics and monitoring (P310), the

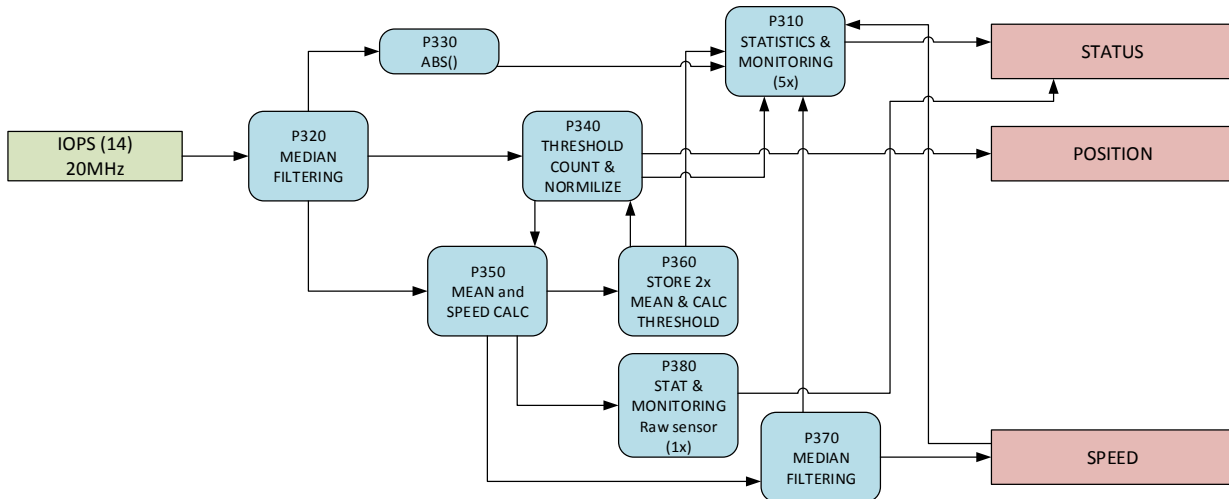


Figure 8.41: IOPS data processing detailed view

One of the question during the implementation was if the representation of the sfixed already used in the design was sufficient for representing the angle between each mirror. This very small value is needed in the calculation of the position and speed of the system. The value for the SPS disk is as low as 0.00031886. Fig.8.42 represents the error on this value introduced by the limited number of decimal digit after the dot. For a

resolution of 16 [bit] after the dot, as it is implemented in the whole design, the error produced is about 0.5[%]. This error is acceptable as this system is attended to be used for redundancy purpose and not for the accurate evaluation of the angular position.

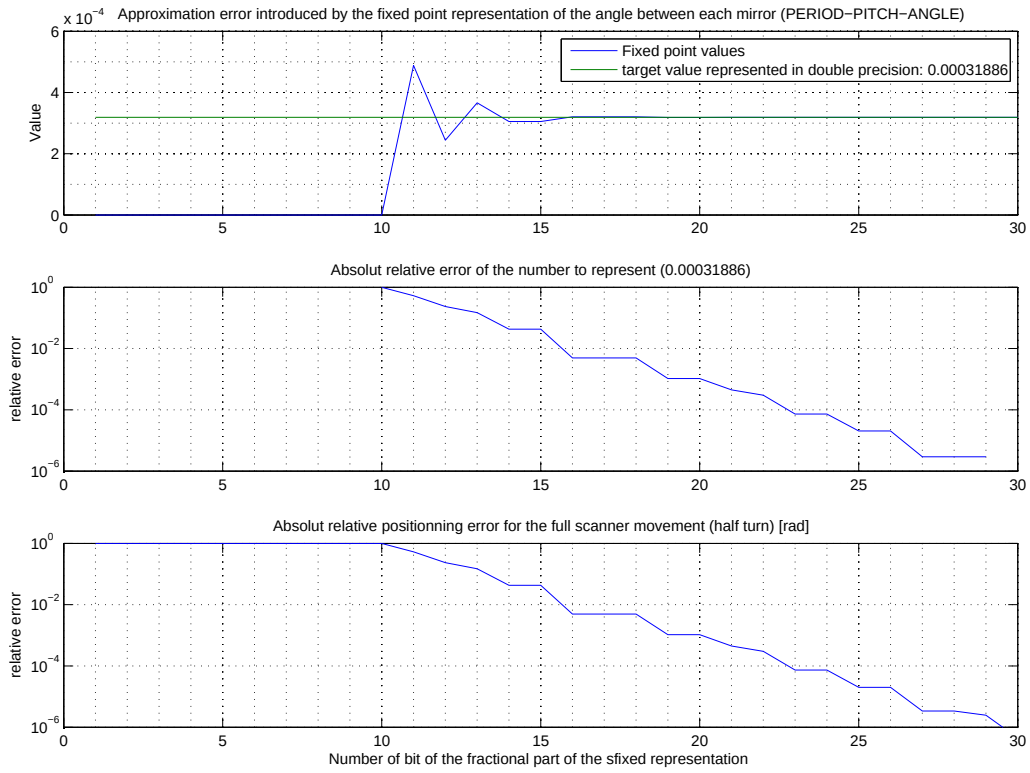


Figure 8.42: IOPS representation error of the angle between each mirror in sfixed of various precision after the dot

8.6 Synthesis

The module 'iops-processing-P300' is build with the following components that have been individually written and verified.

1. median-filter-raw-data
2. abs-sensor-processed-data
3. monitoring-raw-sensor-data
4. monitoring-processed-sensor-data
5. threshold-compare-raw-sensor-data
6. mean-and-period-calc-lpm-divide
7. store-means-calc-thresholds
8. median-filter-processed-data

the challenge of the synthesis was in the division used to calculate the angular speed based on the IOPS period.

The first version of the entity would not synthesise for the target since the division to obtain the speed (normalisation-to-speed/counter-value-one-period) was using fixed point number (32 bits, 16 bits decimal part) and would result by an implementation with numerator of more that 64 bits. Which is not supported by the semiconductor Intellectual Property core (IP) called LPM-DIVIDE, that Quartus would infer when using the division "/".

The first mitigation was to lower the resolution of the decimal part of the numerator to 8 bits, since this representation is enough (constant = 6377.2). The denominator is an integer for the lowest possible decimal part is -1. this fixed the compilation issue.

But the maximum clock speed resulting from this implementation was around 8[MHz]. By changing the type of the division to integer and loosing precision, the speed would increase a little to 15[MHz]. Another modification of a variable to a signal, inserting a pipeline increase the maximum frequency to 16[MHz]. The critical path would still be the division.

Analysis & Synthesis Resource Utilization by Entity								
<<Filter>>								
	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Block Memory Bits	DSP Blocks	Pins	Virtual Pins	
1	└─ iops_processing_P300	3486 (1)	2898 (0)	0	2	241	0	iops_processing_P300
1	└─ abs_sensor_processed_data:duv2	35 (35)	19 (19)	0	0	0	0	iops_processing_P300
2	└─ mean_and_period_calc:duv6	2498 (354)	117 (117)	0	0	0	0	iops_processing_P300
1	└─ lpm_divide:Div0	869 (0)	0 (0)	0	0	0	0	iops_processing_P300
1	└─ lpm_divide_76m:auto_generated	869 (0)	0 (0)	0	0	0	0	iops_processing_P300
1	└─ sign_div_unsign_bnh:divider	869 (0)	0 (0)	0	0	0	0	iops_processing_P300
1	└─ alt_u_div_ure:divider	869 (869)	0 (0)	0	0	0	0	iops_processing_P300
2	└─ lpm_divide:Div1	1275 (0)	0 (0)	0	0	0	0	iops_processing_P300
1	└─ lpm_divide_3ko:auto_generated	1275 (0)	0 (0)	0	0	0	0	iops_processing_P300
1	└─ abs_divider_adg:divider	1275 (0)	0 (0)	0	0	0	0	iops_processing_P300
1	└─ alt_u_div_6se:divider	1275 (1275)	0 (0)	0	0	0	0	iops_processing_P300
3	median_filter_processed_data:duv9	355 (355)	1688 (1688)	0	0	0	0	iops_processing_P300
4	median_filter_raw_data:duv1	145 (145)	735 (735)	0	0	0	0	iops_processing_P300
5	monitoring_processed_sensor_data:duv10	33 (33)	23 (23)	0	0	0	0	iops_processing_P300
6	monitoring_processed_sensor_data:duv11	25 (25)	18 (18)	0	0	0	0	iops_processing_P300
7	monitoring_processed_sensor_data:duv3	32 (32)	23 (23)	0	0	0	0	iops_processing_P300
8	monitoring_processed_sensor_data:duv5	36 (36)	23 (23)	0	0	0	0	iops_processing_P300
9	monitoring_processed_sensor_data:duv8	28 (28)	19 (19)	0	0	0	0	iops_processing_P300
10	store_means_calc_thresholds:duv7	179 (179)	184 (184)	0	0	0	0	iops_processing_P300
11	threshold_compare_raw_sensor_data:du4	119 (119)	49 (49)	0	2	0	0	iops_processing_P300

Figure 8.43: IOPS P300 entity first version of the synthesis with fixed point division, maximum speed 8MHz

Analysis & Synthesis Resource Utilization by Entity									
Compilation Hierarchy Node		LC Combinationals	LC Registers	Block Memory Bits	DSP Blocks	Pins	Virtual Pins	Full Hierarchy Name	
1	└─ [iops_processing_P300]	1920 (1)	2660 (0)	0	2	300	0	iops_processing_P300	
1	└─ [abs_sensor_processed_data:duv2]	31 (31)	23 (23)	0	0	0	0	iops_processing_P300 abs_sensor_processed_data:duv2	
2	└─ [mean_and_period_calc_lpm_divide:duv6]	1064 (91)	709 (91)	0	0	0	0	iops_processing_P300 mean_and_period_calc_lpm_divide:duv6	
1	└─ [lpm_divide:duv1]	735 (0)	390 (0)	0	0	0	0	iops_processing_P300 mean_and_period_calc_lpm_divide:duv6 lpm_divide:duv1	
1	└─ [lpm_divide_5ut:auto_generated]	735 (0)	390 (0)	0	0	0	0	iops_processing_P300 mean_and_period_calc_lpm_divide:duv6 lpm_divide_5ut:auto_generated	
1	└─ [sign_div_unsign_boi:divider]	735 (0)	390 (0)	0	0	0	0	iops_processing_P300 mean_and_period_calc_lpm_divide:duv6 lpm_divide_5ut:auto_generated sign_div_unsign_boi:divider	
1	└─ [alt_u_div_usf:divider]	735 (735)	390 (390)	0	0	0	0	iops_processing_P300 mean_and_period_calc_lpm_divide:duv6 lpm_divide_5ut:auto_generated sign_div_unsign_boi:divider alt_u_div_usf:divider	
2	└─ [lpm_divide:duv2]	238 (0)	228 (0)	0	0	0	0	iops_processing_P300 mean_and_period_calc_lpm_divide:duv6 lpm_divide_5ut:auto_generated sign_div_unsign_boi:divider alt_u_div_usf:divider lpm_divide:duv2	
1	└─ [lpm_divide_7ut:auto_generated]	238 (0)	228 (0)	0	0	0	0	iops_processing_P300 mean_and_period_calc_lpm_divide:duv6 lpm_divide_5ut:auto_generated sign_div_unsign_boi:divider alt_u_div_usf:divider lpm_divide:duv2 lpm_divide_7ut:auto_generated	
1	└─ [sign_div_unsign_boi:divider]	238 (0)	228 (0)	0	0	0	0	iops_processing_P300 mean_and_period_calc_lpm_divide:duv6 lpm_divide_5ut:auto_generated sign_div_unsign_boi:divider alt_u_div_usf:divider lpm_divide:duv2 lpm_divide_7ut:auto_generated sign_div_unsign_boi:divider	
1	└─ [alt_u_div_2tf:divider]	238 (238)	228 (228)	0	0	0	0	iops_processing_P300 mean_and_period_calc_lpm_divide:duv6 lpm_divide_5ut:auto_generated sign_div_unsign_boi:divider alt_u_div_usf:divider lpm_divide:duv2 lpm_divide_7ut:auto_generated sign_div_unsign_boi:divider alt_u_div_2tf:divider	
3	[median_filter_processed_data:duv10]	193 (193)	792 (792)	0	0	0	0	iops_processing_P300 median_filter_processed_data:duv10	
4	[median_filter_raw_data:duv1]	145 (145)	771 (771)	0	0	0	0	iops_processing_P300 median_filter_processed_data:duv10 median_filter_raw_data:duv1	
5	[monitoring_processed_sensor_data:duv11]	28 (28)	19 (19)	0	0	0	0	iops_processing_P300 median_filter_processed_data:duv10 median_filter_raw_data:duv1 monitoring_processed_sensor_data:duv11	
6	[monitoring_processed_sensor_data:duv12]	25 (25)	18 (18)	0	0	0	0	iops_processing_P300 median_filter_processed_data:duv10 median_filter_raw_data:duv1 monitoring_processed_sensor_data:duv11 monitoring_processed_sensor_data:duv12	
7	[monitoring_processed_sensor_data:duv3]	28 (28)	27 (27)	0	0	0	0	iops_processing_P300 median_filter_processed_data:duv10 median_filter_raw_data:duv1 monitoring_processed_sensor_data:duv11 monitoring_processed_sensor_data:duv12 monitoring_processed_sensor_data:duv3	
8	[monitoring_processed_sensor_data:duv5]	32 (32)	27 (27)	0	0	0	0	iops_processing_P300 median_filter_processed_data:duv10 median_filter_raw_data:duv1 monitoring_processed_sensor_data:duv11 monitoring_processed_sensor_data:duv12 monitoring_processed_sensor_data:duv3 monitoring_processed_sensor_data:duv5	
9	[monitoring_processed_sensor_data:duv9]	28 (28)	19 (19)	0	0	0	0	iops_processing_P300 median_filter_processed_data:duv10 median_filter_raw_data:duv1 monitoring_processed_sensor_data:duv11 monitoring_processed_sensor_data:duv12 monitoring_processed_sensor_data:duv3 monitoring_processed_sensor_data:duv5 monitoring_processed_sensor_data:duv9	
10	[monitoring_raw_sensor_data:duv7]	27 (27)	18 (18)	0	0	0	0	iops_processing_P300 median_filter_processed_data:duv10 median_filter_raw_data:duv1 monitoring_processed_sensor_data:duv11 monitoring_processed_sensor_data:duv12 monitoring_processed_sensor_data:duv3 monitoring_processed_sensor_data:duv5 monitoring_processed_sensor_data:duv9 monitoring_raw_sensor_data:duv7	
11	[store_means_calc_thresholds:duv8]	177 (177)	184 (184)	0	0	0	0	iops_processing_P300 median_filter_processed_data:duv10 median_filter_raw_data:duv1 monitoring_processed_sensor_data:duv11 monitoring_processed_sensor_data:duv12 monitoring_processed_sensor_data:duv3 monitoring_processed_sensor_data:duv5 monitoring_processed_sensor_data:duv9 monitoring_raw_sensor_data:duv7 store_means_calc_thresholds:duv8	
12	[threshold_compare_raw_sensor_data:du4]	141 (141)	53 (53)	0	2	0	0	iops_processing_P300 median_filter_processed_data:duv10 median_filter_raw_data:duv1 monitoring_processed_sensor_data:duv11 monitoring_processed_sensor_data:duv12 monitoring_processed_sensor_data:duv3 monitoring_processed_sensor_data:duv5 monitoring_processed_sensor_data:duv9 monitoring_raw_sensor_data:duv7 store_means_calc_thresholds:duv8 threshold_compare_raw_sensor_data:du4	

Figure 8.44: IOPS P300 entity final version of the synthesis with explicit use of the IP core LPM-DIVIDE, maximum speed 79[MHz]

The root of the problem is the IP core LPM-DIVIDE again. This core is not pipelined when inferred by the synthesis tool. It is a pure combinational implementation of a division, it has no output enable, which indicate that the division is completed.

The solution to solve this was to instantiate this IOPS LPM-DIVIDE component explicitly inside code. At this point it is possible to specify the pipeline level. Some additional logic was placed to preform the delay for the use of the division output.

```

component LPM_DIVIDE
  generic (
    LPM_WIDTHN : natural;
    LPM_WIDTHD : natural;
    LPM_NREPRESENTATION : string := "UNSIGNED";
    LPM_DREPRESENTATION : string := "UNSIGNED";
    LPM_PIPELINE : natural := 0;
    LPM_TYPE : string := L_DIVIDE;
    LPM_HINT : string := "UNUSED");
  port (
    NUMER : in std_logic_vector(LPM_WIDTHN-1 downto 0);
    DENOM : in std_logic_vector(LPM_WIDTHD-1 downto 0);
    ACLR : in std_logic := '0';
    CLOCK : in std_logic := '0';
    CLKEN : in std_logic := '1';
    QUOTIENT : out std_logic_vector(LPM_WIDTHN-1 downto 0);
    REMAIN : out std_logic_vector(LPM_WIDTHD-1 downto 0);
  );
end component;

```

The option LPM-PIPELINE is then used to define the number of pipeline levels for this operation. The synthesis was tested for a level of pipeline equal to 5 and the maximum frequency went to 62[MHz]. In the final version, the pipeline is equal to 10 and the resulting maximum frequency jumped to 79[MHz].

8.6.1 Verification

The verification of the complete IOPS processing was performed using stimuli generation from real data sets. The output was then compared with the prototyped algorithm in Matlab.

To do this, the top entity 'iops-processing-P300' was connected to the field data generator and the outputs were saved into individual text file. The test bench entity description below shows that many generics parameters are available. This is to be able to simulate the design for multiple set of input data without having to recompile it. The 'OUTFOLDER' and 'FILE-PREFIX' generics are allowing to save multiple results set into separated folders.

```
entity iops_processing_P300_tb is
    generic(
        OUTFOLDER      : STRING;
        FILE_PREFIX     : STRING;
        IOPS_1_DATA     : STRING := "data_in_scan_ops_1_07_Jan_2016_15_43_32.hex";
        IOPS_2_DATA     : STRING := "data_in_scan_ops_2_07_Jan_2016_15_43_32.hex";
        IOPS_DEPTH      : integer;
        RESOLVER_DATA   : STRING;
        POSITION_DATA    : STRING;
        POS_DEPTH       : integer;
        CURR_U_DATA     : STRING;
        CURR_V_DATA     : STRING;
        CURR_W_DATA     : STRING;
        CURR_DEPTH      : integer;
        FIXPT_SIZE_HIGH : integer := 16;
        FIXPT_SIZE_LOW  : integer := -16;
        DIFF_MEAN_TO_THRESHOLD : integer := 300; -- in bit, to calculate high and low threshold
        INITIAL_MEAN_VALUE : integer := 1000 -- default value at reset
    );
end iops_processing_P300_tb;
```

Fig.8.45 shows a simulation of the top processing entity of the IOPS being stimulated with the channel 1 of the S4 dataset. The simulation output files were then used in MatLab to plot the resulting position and speed compared to the resolver information presented in Fig.8.46. As expected, the two curves are very close to each other. During the first verification iteration, by comparing these two outputs, the constant representing the angle between each mirror was corrected, as the two curves would largely diverge.

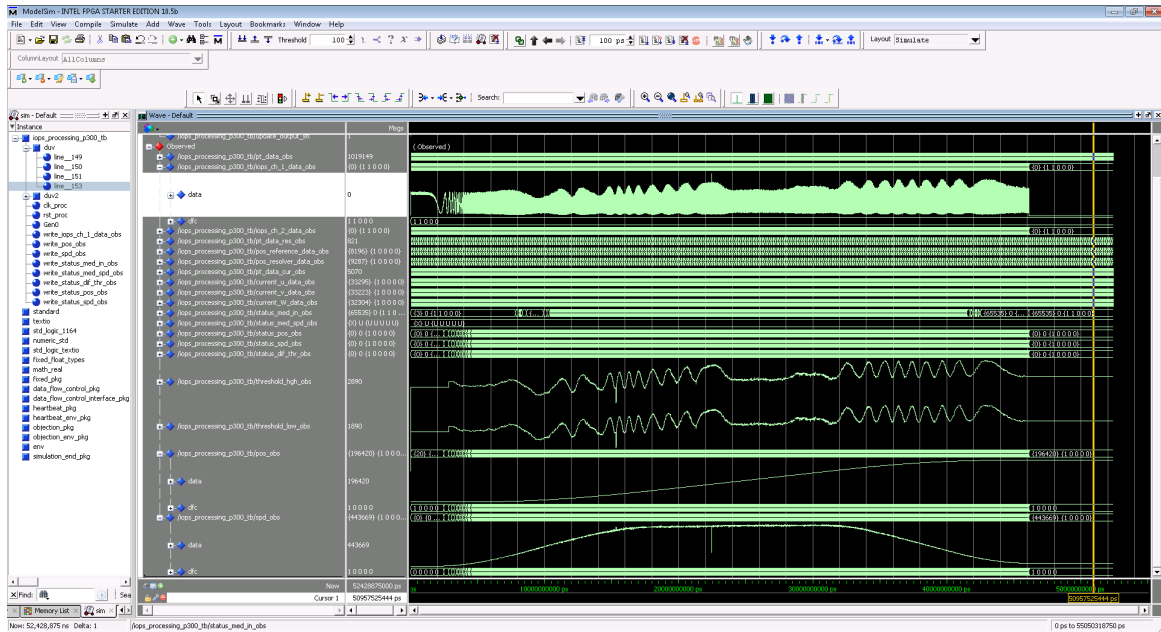


Figure 8.45: IOPS top processing simulation result using the S4 dataset IOPS CH1: 5.1

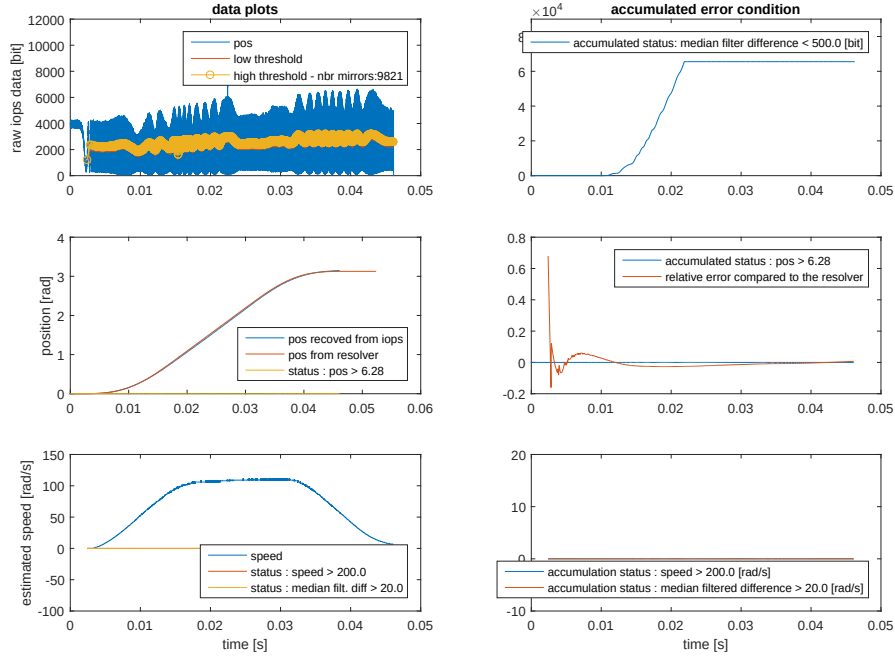


Figure 8.46: IOPS top processing simulation result using the S4 dataset IOPS CH1 5.1, with integer division for the speed calculation

Fig.8.47 shows a simulation of the top processing entity of the IOPS being stimulated for the failure case FM3 and Fig.8.48 for the case FM4. As expected, the output is affected by the missing signal and present a corrupted output compared to the resolver.

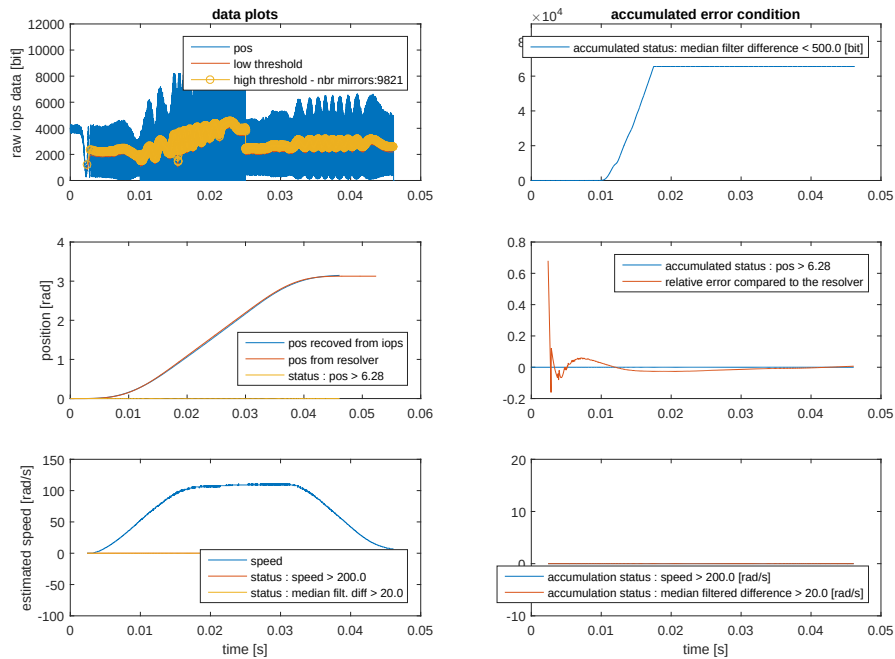


Figure 8.47: IOPS top processing simulation result using the S4 dataset IOPS CH1 modified for FM3: 5.3, with integer division for the speed calculation

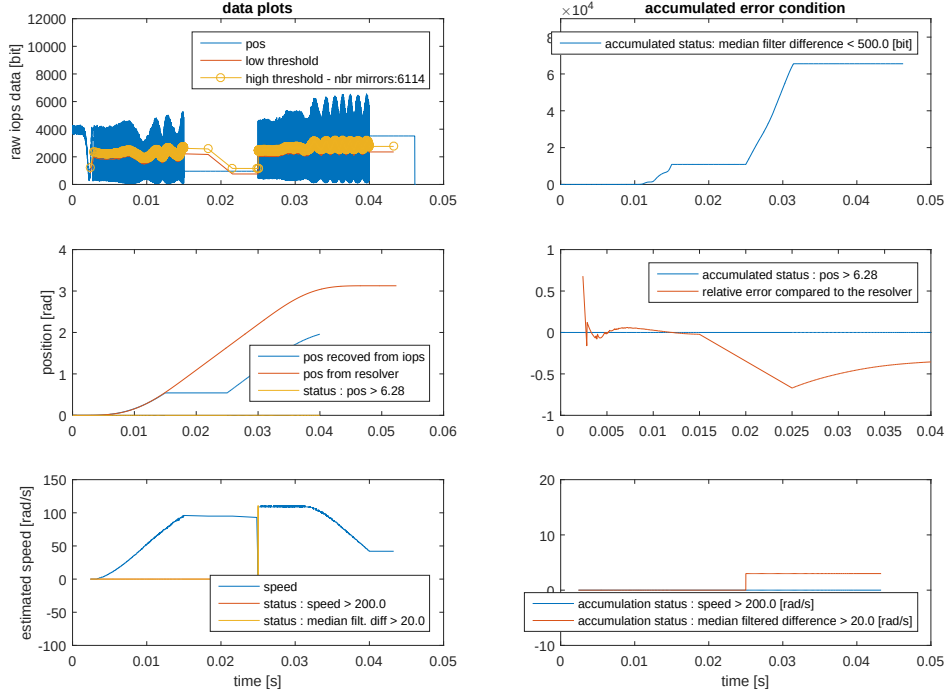


Figure 8.48: IOPS top processing simulation result using the S4 dataset IOPS CH1 modified for FM4: 5.3, with integer division for the speed calculation

8.7 Algorithm implementations comparison

The output files generated by the test bench have different forms depending on the output types, if it is a structured type or not. For each of these output type, one MatLab function was written to load them into the workspace. Then the script compares these results processed with the VHDL algorithms against the ones generated by the prototypes written with MatLab.

Fig.8.49 presents the result of this comparison for the reference channel 1 of the data set S4, see Table.8.3.

The first line shows the IOPS input data and the thrshold level for the MatLab prototype algorithm (left) and the VHDL version (right). The position plot (centre left) and the difference between the two implementation (bottom left) shows that the difference is as low as 0.5[%] which is directly comparable with the precision of the angle between each mirror. The speed plot (centre right) and the comparison (bottom right) shows a maximum error of about 3[%]. We can also distinguish the quantification of the speed which is calculated with an integer in the VHDL version of the algorithm.

Fig.8.50 presents the result for the reference channel 2 of the data set S4, see Table.8.3. this data set represent a signal of a sensor having abnormal behaviour. The amplitude modulation is supposed to be lower on the final system. Nerveless, the two algorithm are behaviours in a similar manner for the position. On the speed comparison, we can see that the speed differs between the algorithms during discontinuous events. The trend is similar, but the size of the deeps are different. This edge case is a very good test case for pushing further the comparison of the two implementations. In the frame of this work, this result provides enough confidence consider this entity as validated.

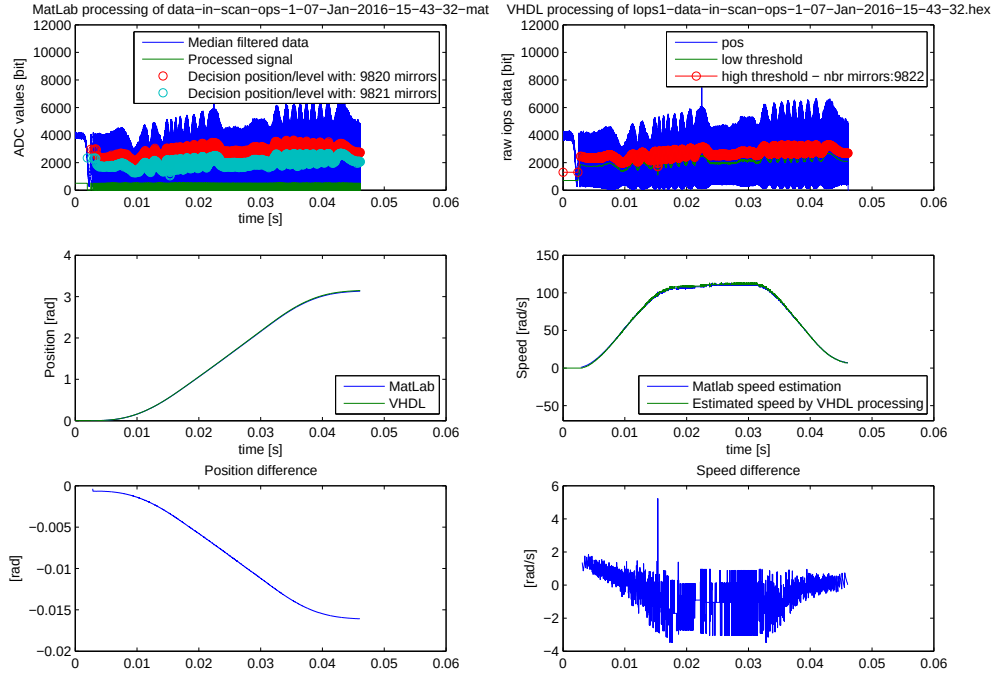


Figure 8.49: Comparison of IOPS processing implementations using S4 dataset IOPS CH1: 5.1

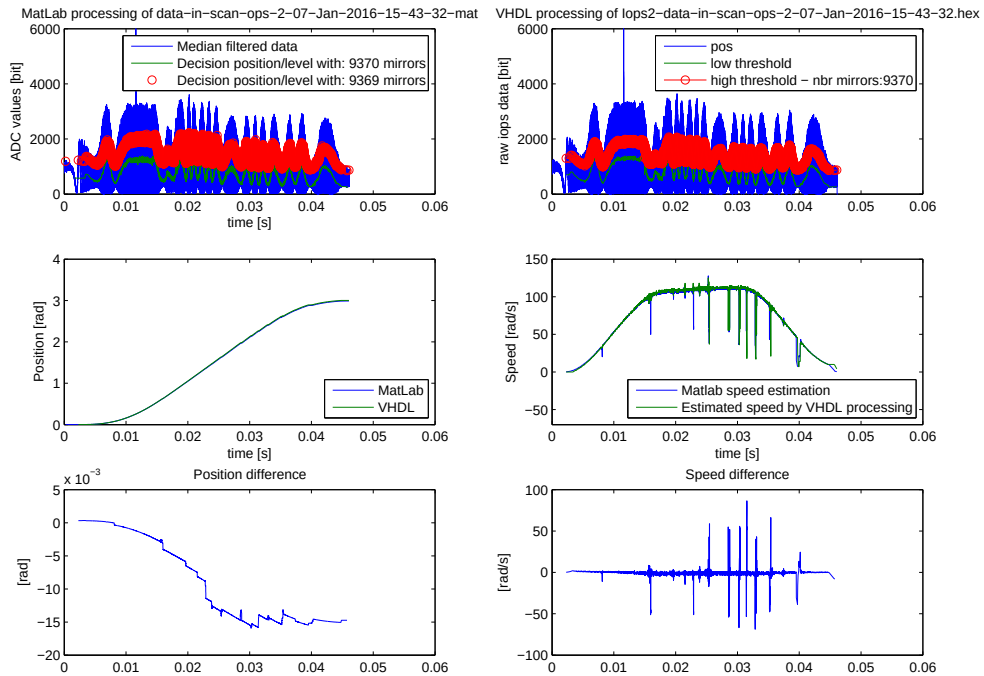


Figure 8.50: Comparison of IOPS processing implementations using S4 dataset IOPS CH2: 5.1

Chapter 9

Hall effect based current transducers

The knowledge of the current flowing inside the motor phases is of crucial importance. In fact the current flowing in the motor coils is producing the mechanical torque which defines the acceleration that the system will experience. An accurate control of this current allows an accurate control of the electro-mechanical system.

The new generation of Beam Wire Scanner (BWS) uses a Permanent Magnet Synchronous Motor (PMSM) with three phases connected in a star topology. The currents have to be accurately controlled to produce the highest torque without losses. This current is produced by an inverter which is a power supply controlled by Pulse Width Modulation (PWM). The current feedback is directly acting on this signal after measuring the present current using the transducer.

The current feedback usually needs only two out of the three currents as the third one is only the sum of the two others due to the star topology. A loss of one of the two would signify a complete loss of the motor control.

The Beam Wire Scanner (BWS) is equipped with one current sensor for each of its three phases. This redundancy can be used to monitor and detect failure of the sensor acquisition chain. More complex scheme can even be imagined where one of the three sensor is isolated and its measure would be simulated by the two others.

9.1 Technology and accuracy

This sensor is measuring the current flowing inside a conductor. This conductor is physically inserted through its measuring ring visible on Fig.9.1. This transducer produces a current on its secondary circuit which is proportional to the current flowing inside the conductor. The principle uses an active circuit sensing the primary current based on hall effect. A compensation circuit then tries to compensate the secondary current produced, which makes it an efficient close loop system. The output is equal to the compensation current.

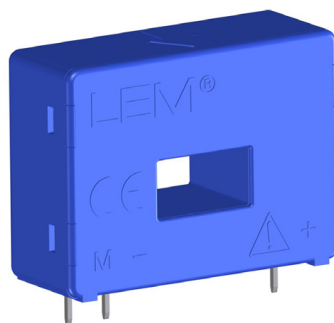


Figure 9.1: Current transducer LA-100-P used in the BWS inverter

This sensor is given for a measuring range up to $\pm 150[\text{A}]$, but the electronics is built to focus on a smaller range of $\pm 50[\text{A}]$. The linearity error is in the order of $0.15[\%]$ and the accuracy is given at $\pm 0.45[\%]$. The overall accuracy is then in the order of $1[\%]$.

The digitalisation of this signal take place with an 16 [bit] ADC from TI number ADC161S626. The range of the measure defined by the datasheet and the signal conditioning circuit is $\pm 2.5[\text{V}]$ for $\pm 50[\text{A}]$ and occupies the full scale of the Analog-to-Digital Converter (ADC). this makes the last significant bit to be equal to

$$\frac{100[\text{A}]}{2^{16} - 1} = 1.5[\text{mA}] \quad (9.1)$$

Due to various sources of uncertainty and noise, this value is far from the reality. Fig.9.2 shows the current measured during a complete system displacement in the two direction. The last line is the histogram of the residues between current measure and a linear fit for a defined range. The standard deviation is in the range of 0.43 to $0.47 [\text{A}]$ which means that the noise to take into account in this acquisition chain is in the order of $\pm 0.5[\text{A}]$ or $\pm 1[\%]$.

This uncertainty of the sensor accuracy need to be added to the uncertainty related to the stochastic noise. The global uncertainty becomes $2[\%]$ of the measurement range.

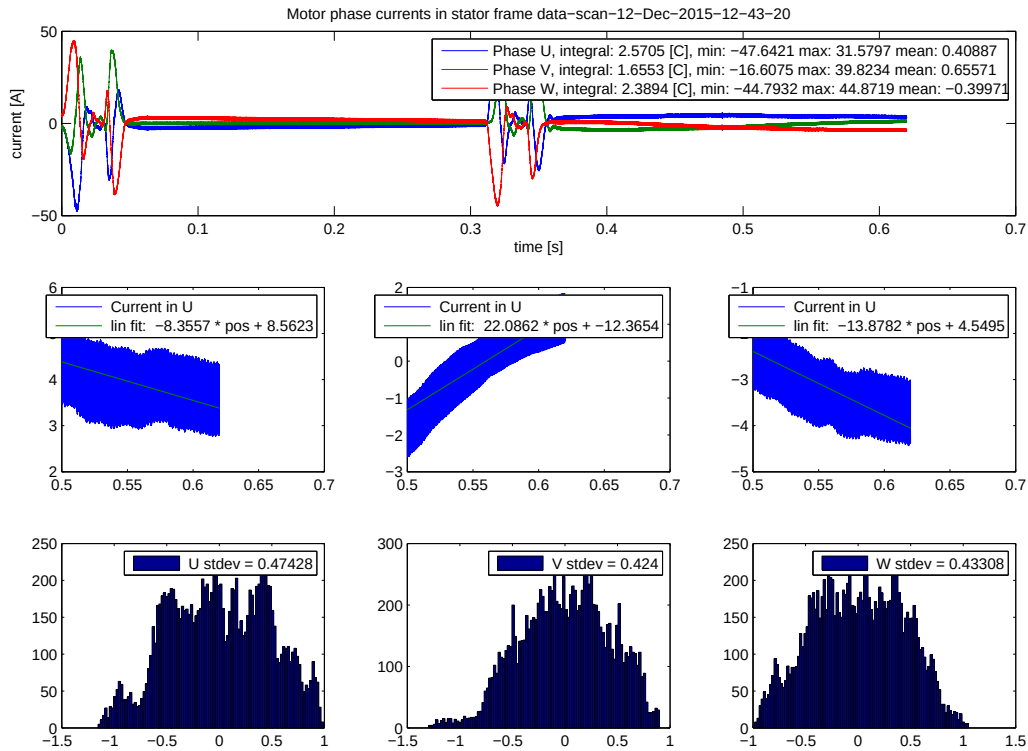


Figure 9.2: Statistics of the currents U, V, W in the stator frame

Finally, if currents from of the three phases are sum, the errors of each terms will be sum as well and will become as high as $6[\%]$ of the measuring range. the sum of the three current may be of by a maximum of $\pm 3[\text{A}]$

9.2 field data characterisation

The first step of the data processing was to develop a mechanism to look at the data sets and to characterise these measurements. Fig.9.2 shows an example of the analysis script output for one measure cycle. the three currents a plot together (top). The second line shows for the three currents on particular range of data which is used for evaluating the measure noise. The last line is the histogram of the residues between the current measures and the linear fit for the defined range.

9.3 Processing algorithm prototyping

There is one current transducers, one for each phases. According to the schematics of the motor connection, there is only need of two measures, since the last branch is the sum of the two others. The processing can take advantage of this redundancy. The currents node equation for the phases connected with a star topology is

$$I_u + I_v + I_w = 0$$

This result of this sum can be monitored using a threshold with predefined value. this processing unit could then discover if a problem appears. this could be a current transducer acquisition chain failure, or a motor phase issue. In both cases, the system needs to be verified.

Some other type of failure could be detected by building more complex system. By reconstructing the currents from the applied voltages and the system electrical characteristics, these could be compared to the measured current to detect which phase failed. Another type of processing is possible based on observers which potentially could reconstruct the position from the currents measurements and be compared with other sensors. Such techniques are outside of the thesis scope as their complexity would requires a complete work for themselves.

Fig.9.3 present a prototyping exercise where one current phase is compared to the minus of the two othe phase summed. We can see that this assumption is verified as the two curves are on top of each other for all the three phases.

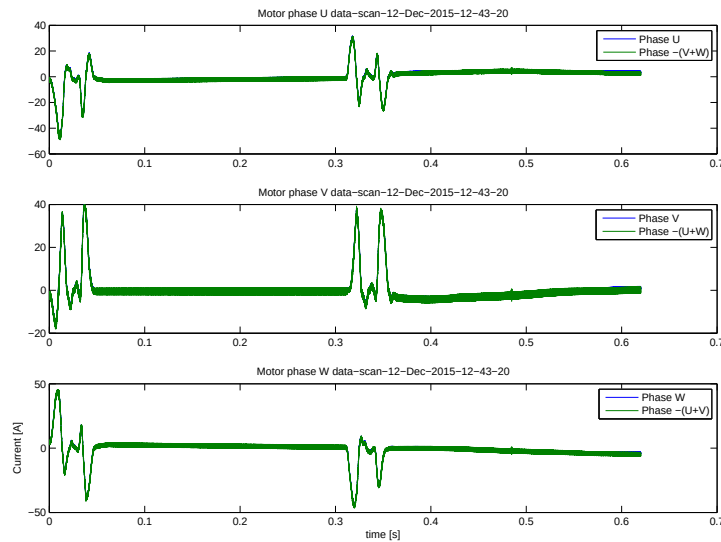


Figure 9.3: Third current calculation based on the two others

In addition to the median filtering of the three input, as already done for the other sensors, the prototyping algorithm integrate the conversion from raw data to currents in ampere. Then the three currents are sum and this result is monitored by a predefined threshold.

9.4 VHDL implementation

The processing algorithm for the currents has been written as a self contained entity as defined in Chap.6. Fig.9.4 show the entity with its inputs and outputs.

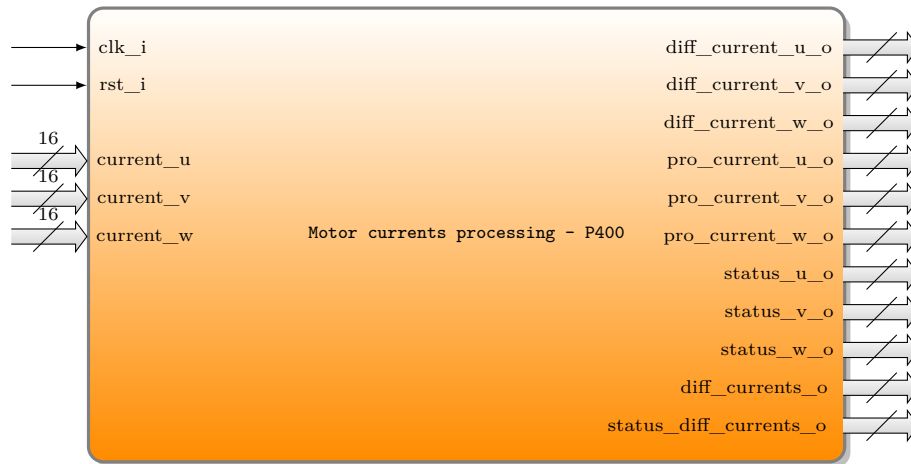


Figure 9.4: The current processing component

The conversion of the prototype algorithm written in MatLab to a VHSIC Hardware Description Language (VHDL) description was performed manually according to the defined procedure in 4.2. The following paragraph describing the implementation comes from the data processing overview chapter in 6

The data flow of the currents is presented on Fig.9.5. The blue part as been implemented while the white part is foreseen for future work. Each of the three incoming data (u, v, w) is firstly filtered using a median filter and the difference between input and output is monitored. Then the values are converted to correctly represent the physical quantity, the currents in Ampere. To detect failure, the three current are sum (P440) and the result is monitored using a predefined threshold (P410). the absolute value of the currents are monitored as well against predefined thresholds and the status are send further.

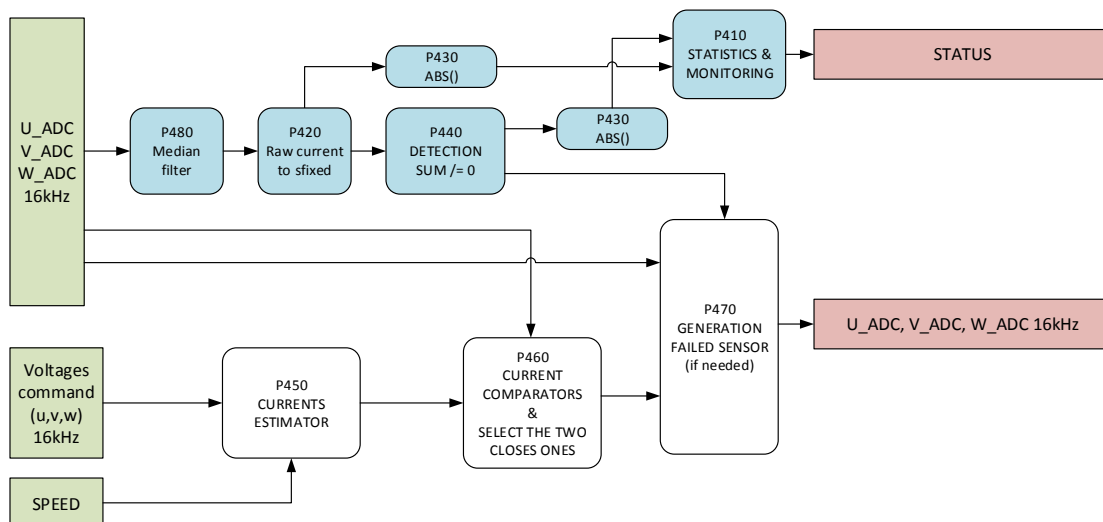


Figure 9.5: Motor current data processing

9.5 Synthesis

A first synthesis of this module was performed without archiving the minimal clocking speed required. This first version was quoted for 25 [MHz]. After analysis, this was again due to long combinational logic between the output of the median filter and the conversion from raw current representation to sfixed in Ampere. Fig.9.6 is a screenshot of the the resources utilisation performed by the synthesis tool on the currents processing entity for the given target.

Analysis & Synthesis Resource Utilization by Entity								
	Compilation Hierarchy Node	LC Combinationals	LC Registers	Block Memory Bits	DSP Blocks	Pins	Virtual Pins	Full Hierarchy Name
1	└─ currents_processing_P400	2535 (0)	3066 (0)	0	6	482	0	currents_processing_P400
2	└─ labs_sensor_processed_data:duv14	49 (49)	24 (24)	0	0	0	0	currents_processing_P400 labs_sensor_processed_data:duv14
3	└─ labs_sensor_processed_data:duv7	47 (47)	22 (22)	0	0	0	0	currents_processing_P400 labs_sensor_processed_data:duv7
4	└─ labs_sensor_processed_data:duv8	47 (47)	22 (22)	0	0	0	0	currents_processing_P400 labs_sensor_processed_data:duv8
5	└─ labs_sensor_processed_data:duv9	47 (47)	22 (22)	0	0	0	0	currents_processing_P400 labs_sensor_processed_data:duv9
6	└─ median_filter_raw_currents:duv1	201 (201)	882 (882)	0	0	0	0	currents_processing_P400 median_filter_raw_currents:duv1
7	└─ median_filter_raw_currents:duv2	201 (201)	882 (882)	0	0	0	0	currents_processing_P400 median_filter_raw_currents:duv2
8	└─ median_filter_raw_currents:duv3	201 (201)	882 (882)	0	0	0	0	currents_processing_P400 median_filter_raw_currents:duv3
9	└─ monitoring_processed_sensor_data:duv10	28 (28)	27 (27)	0	0	0	0	currents_processing_P400 monitoring_processed_sensor_data:duv10
10	└─ monitoring_processed_sensor_data:duv11	28 (28)	27 (27)	0	0	0	0	currents_processing_P400 monitoring_processed_sensor_data:duv11
11	└─ monitoring_processed_sensor_data:duv12	28 (28)	27 (27)	0	0	0	0	currents_processing_P400 monitoring_processed_sensor_data:duv12
12	└─ monitoring_processed_sensor_data:duv15	28 (28)	27 (27)	0	0	0	0	currents_processing_P400 monitoring_processed_sensor_data:duv15
13	└─ raw_current_to_sfixed:duv4	483 (483)	35 (35)	0	2	0	0	currents_processing_P400 raw_current_to_sfixed:duv4
14	└─ raw_current_to_sfixed:duv5	483 (483)	35 (35)	0	2	0	0	currents_processing_P400 raw_current_to_sfixed:duv5
15	└─ raw_current_to_sfixed:duv6	483 (483)	35 (35)	0	2	0	0	currents_processing_P400 raw_current_to_sfixed:duv6
16	└─ sum_current_sensor_signals:duv13	181 (181)	117 (117)	0	0	0	0	currents_processing_P400 sum_current_sensor_signals:duv13

Figure 9.6: Currents processing entity synthesis results, maximum speed 25[MHz]

The problematic instruction is the following:

```
current_sfixed_o.data <= std_logic_vector( resize (
    input_conversion_factor_sfixed *
    to_sfixed(to_integer(unsigned(raw_current_i.data))-(2**16-1)/2, FIXPT_SIZE_HIGH-1, FIXPT_SIZE_LOW),
    FIXPT_SIZE_HIGH - 1, FIXPT_SIZE_LOW));
```

This long instruction was performing the conversion from unipolar binary representation to bipolar coded with fixed point in Ampere. The incoming data is multiplied by a conversion factor and then half the range of the ADC is subtracted to it.

To fix the issue, this instruction was pipelined as followed

```
constant half_adc_range_adc : integer := ((2**current_sensor_type.data'length)-1)/2;
signal intermediate_result : integer;

intermediate_result <= to_integer(unsigned(raw_current_i.data)) - half_adc_range_adc;

current_sfixed_o.data <= std_logic_vector( resize( input_conversion_factor_sfixed *
    to_sfixed(intermediate_result, FIXPT_SIZE_HIGH - 1, FIXPT_SIZE_LOW), FIXPT_SIZE_HIGH - 1, FIXPT_SIZE_LOW));
```

The resulting maximal clocking speed jumped to the comfortable value of 92 [MHz]. Fig.9.7 presents the resources utilisation of the final entity.

Analysis & Synthesis Resource Utilization by Entity								
	Compilation Hierarchy Node	LC Combinationals	LC Registers	Block Memory Bits	DSP Blocks	Pins	Virtual Pins	Full Hierarchy Name
1	└─ currents_processing_P400	1673 (0)	3160 (0)	0	6	482	0	currents_processing_P400
2	└─ labs_sensor_processed_data:duv14	50 (50)	25 (25)	0	0	0	0	currents_processing_P400 labs_sensor_processed_data:duv14
3	└─ labs_sensor_processed_data:duv7	50 (50)	25 (25)	0	0	0	0	currents_processing_P400 labs_sensor_processed_data:duv7
4	└─ labs_sensor_processed_data:duv8	50 (50)	25 (25)	0	0	0	0	currents_processing_P400 labs_sensor_processed_data:duv8
5	└─ labs_sensor_processed_data:duv9	50 (50)	25 (25)	0	0	0	0	currents_processing_P400 labs_sensor_processed_data:duv9
6	└─ median_filter_raw_currents:duv1	273 (273)	882 (882)	0	0	0	0	currents_processing_P400 median_filter_raw_currents:duv1
7	└─ median_filter_raw_currents:duv2	273 (273)	882 (882)	0	0	0	0	currents_processing_P400 median_filter_raw_currents:duv2
8	└─ median_filter_raw_currents:duv3	273 (273)	882 (882)	0	0	0	0	currents_processing_P400 median_filter_raw_currents:duv3
9	└─ monitoring_processed_sensor_data:duv10	29 (29)	27 (27)	0	0	0	0	currents_processing_P400 monitoring_processed_sensor_data:duv10
10	└─ monitoring_processed_sensor_data:duv11	29 (29)	27 (27)	0	0	0	0	currents_processing_P400 monitoring_processed_sensor_data:duv11
11	└─ monitoring_processed_sensor_data:duv12	29 (29)	27 (27)	0	0	0	0	currents_processing_P400 monitoring_processed_sensor_data:duv12
12	└─ monitoring_processed_sensor_data:duv15	29 (29)	27 (27)	0	0	0	0	currents_processing_P400 monitoring_processed_sensor_data:duv15
13	└─ raw_current_to_sfixed:duv4	119 (119)	63 (63)	0	2	0	0	currents_processing_P400 raw_current_to_sfixed:duv4
14	└─ raw_current_to_sfixed:duv5	119 (119)	63 (63)	0	2	0	0	currents_processing_P400 raw_current_to_sfixed:duv5
15	└─ raw_current_to_sfixed:duv6	119 (119)	63 (63)	0	2	0	0	currents_processing_P400 raw_current_to_sfixed:duv6
16	└─ sum_current_sensor_signals:duv13	181 (181)	117 (117)	0	0	0	0	currents_processing_P400 sum_current_sensor_signals:duv13

Figure 9.7: Final currents processing entity synthesis results, with a maximum clocking speed 92 [MHz]

The module 'currents-processing-P400' uses the following components:

1. median-filter-raw-currents
2. raw-current-to-sfixed
3. monitoring-processed-sensor-data
4. abs-sensor-processed-data
5. sum-current-sensor-signals

9.6 Verification

The verification of the complete currents processing module was performed using stimuli generation from the real data sets. The output was then compared with the prototyped algorithm in Matlab.

Fig.9.8 presents the Modelsim simulation of the currents processing core being stimulated with the dataset S4 5.1.

Fig.9.9 shows the simulation output files plot with MatLab. The raw input current is filtered with the median filter (top) then is converted into fixed point representation having the ampere as unit (middle). On this same plot the monitoring of the value is represented and we can see that the low limit which was set to 40 [A] is correctly monitored.

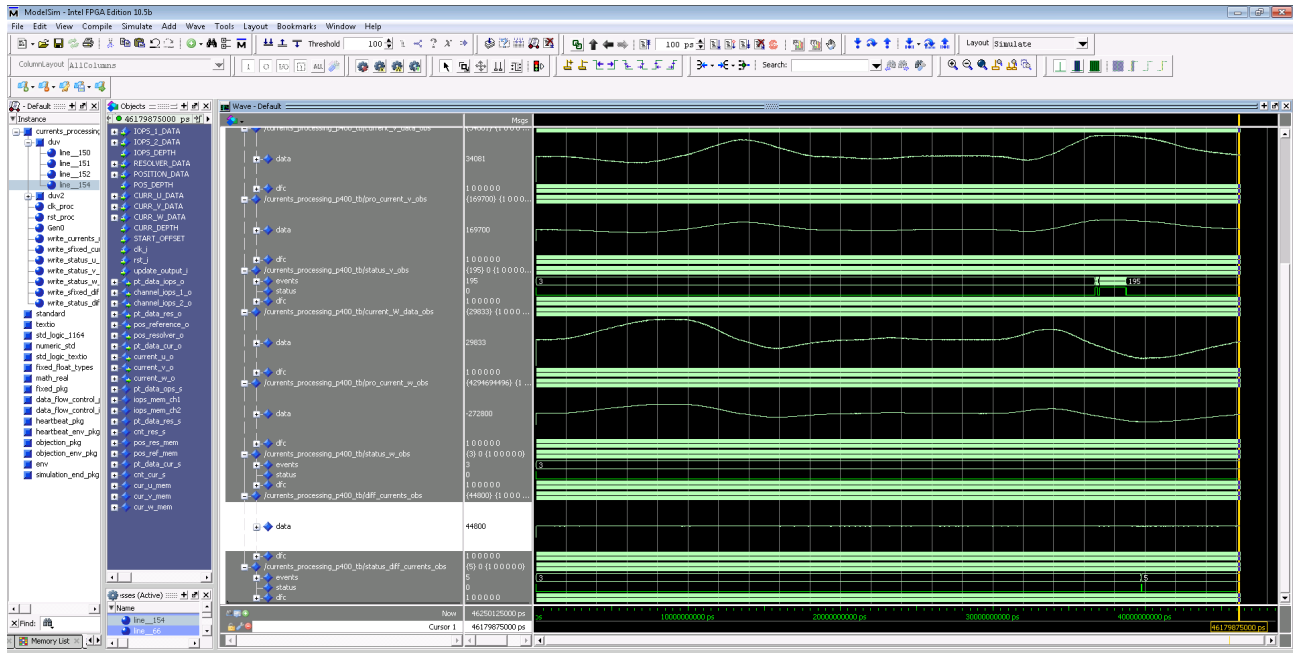


Figure 9.8: VHDL implementation of the currents processing P400 simulated with Modelsim using the S4 dataset: 5.1

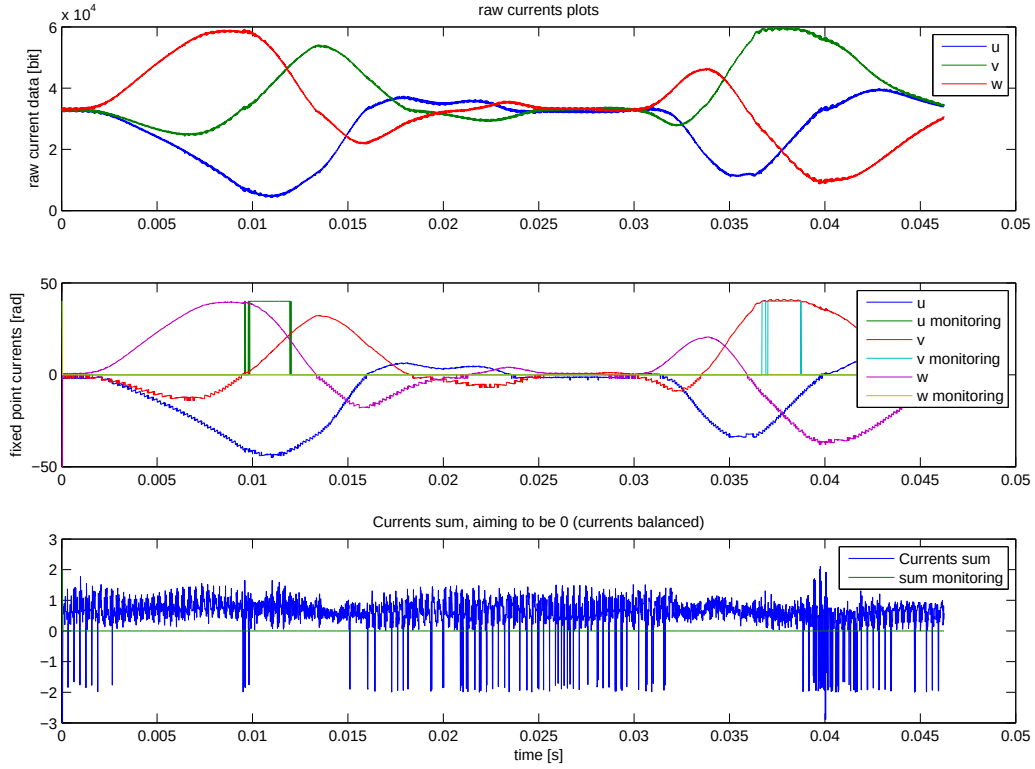


Figure 9.9: VHDL implementation of the currents processing P400 simulated with Modelsim using the S4 dataset: 5.1

9.7 Algorithm implementations comparison

The output files generated by the test bench have different forms depending on the output types, if it is a structured type or not. For each of these output type, one MatLab function was written to load them into the workspace. Then the script compares these results processed with the VHDL algorithms against the ones generated by the prototypes written with MatLab.

Fig.9.10 for the current U, Fig.9.11 for the current V and Fig.9.12 for the current W are showing the comparison of the prototype algorithm written with MatLab against the implementation in VHDL. The top plots is the currents and the monitoring status while the bottom plots is the difference between the two curves. The differences are relatively large with up to 4.5 [A]. There is also a structure visible on the curve of the VHDL algorithm which very much looks like quantisation steps. Further investigation will be needed to limit these differences by careful investigation.

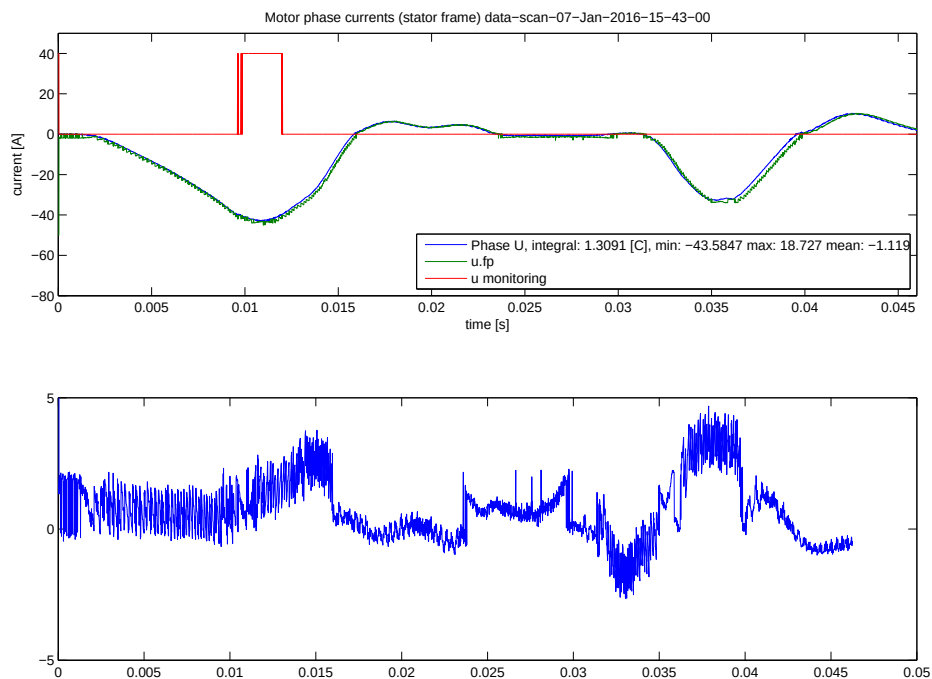


Figure 9.10: Comparison of the U current processing implementations using S4 dataset: [5.1](#)

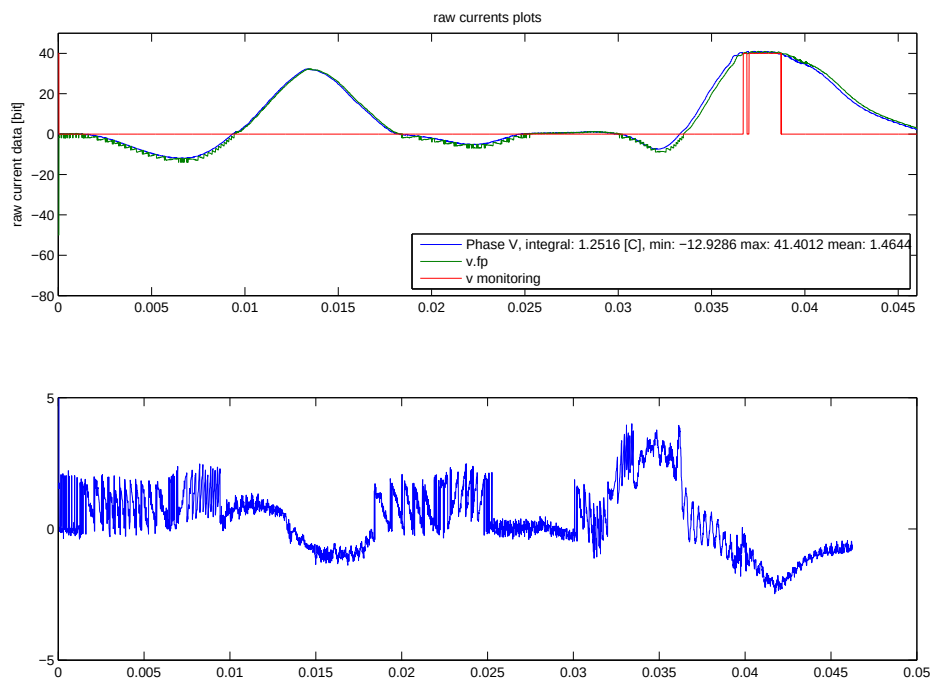


Figure 9.11: Comparison of the V current processing implementations using S4 dataset: [5.1](#)

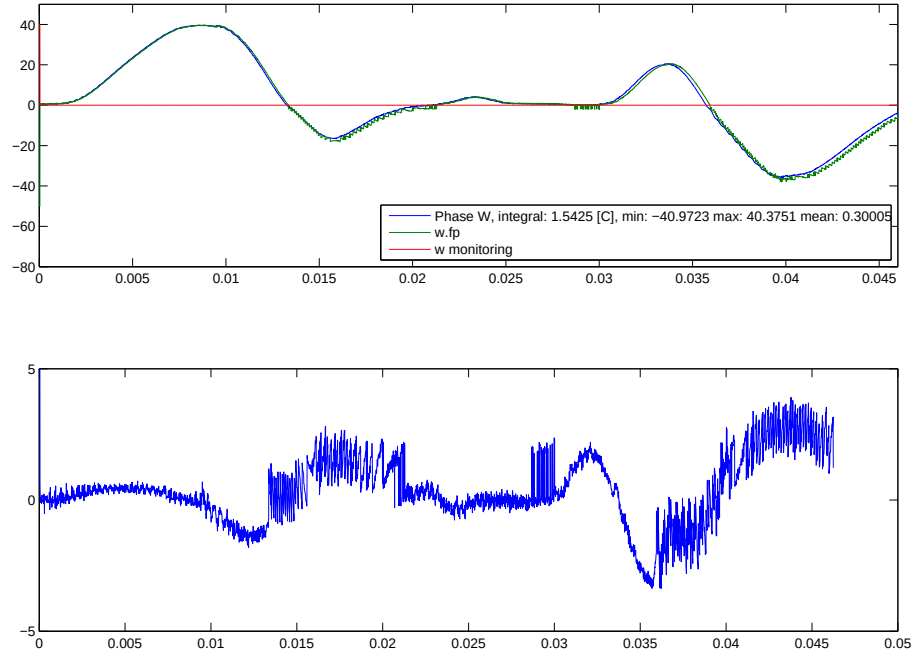


Figure 9.12: Comparison of the W current processing implementations using S4 dataset: 5.1

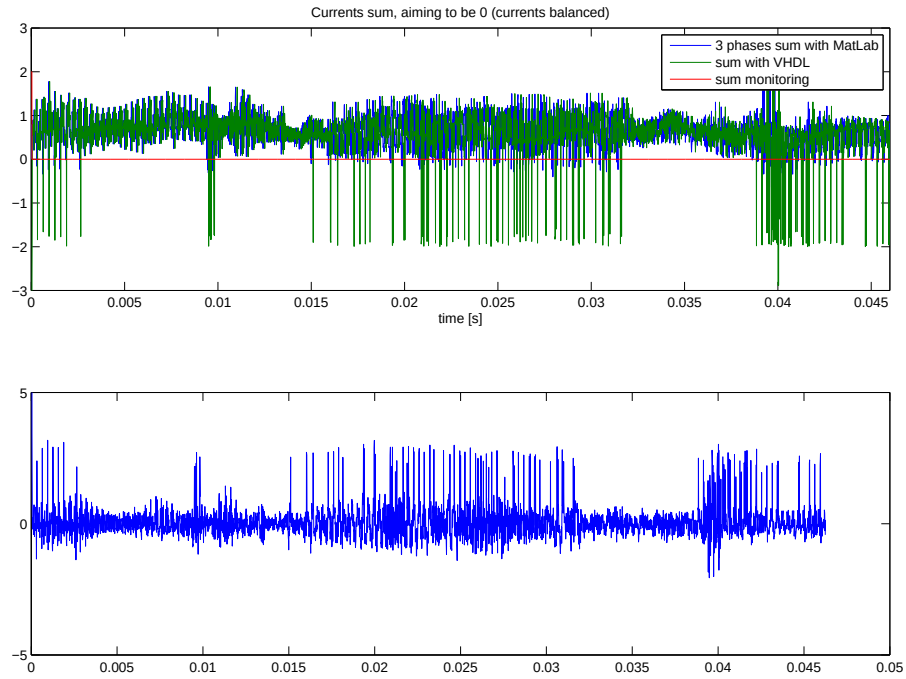


Figure 9.13: Comparison of the current summing processing implementations using S4 dataset: 5.1

Fig.9.13 shows that for both algorithm implementation, the resulting sum of the three currents is limited in the range of $\pm 2[A]$ (top plot), to the exception of on moment where the value goes below $3[A]$ and is catch by the monitoring system. The difference MatLab-VHDL shows values in the range of $\pm 2.5[A]$.

Chapter 10

Sensors data fusion

Now that all the three sensors type, resolver, IOPS and currents are pre-processed individually and the robustness of their delivered signal is improved, the last part of the processing can take place. It consist in using the data from multiple sensors to provide an improved output compared to single sensor solution.

The goal of the data combination also called fusion, is to increase the detectability of failures by comparing the sensor measures between them. Ones a failure is detected on one source, it is then possible to switch to a more reliable one. The outcome of the fusion can also be a mathematical operation between the sources aims at "obtaining information of superior quality" [25]

The wikipedia definition gives that "sensor fusion is combining of sensory data or data derived from disparate sources such that the resulting information has less uncertainty than would be possible when these sources were used individually" [26].

In the frame of this thesis, the sensors fusion is aiming at combining the information of position and speed coming from the four different sources available.

The reference position and speed are the look-up tables inside the FPGA which are used for setting the expected position and speed over time, during an measure cycle of the system. The data inside is very clean, i.e. without statistical noise, to the exception of the one introduced by the quantification. Even being not an physical measure of the system, this information could be used to 'trick' the feedback system. This information is still better than a completely wrong position that would induce abnormal movement leading to system damages. If this information is used instead of others, this completely disable the feedback part of the system leaving the feed-forward only in charge. The result would be a measure cycle perturbed and without system damages.

The resolver provides the absolute angular position at any time, which makes it the most suitable for the brushless motor control. Combined with an resolver to digital converter electronic chip, it provides robust and reliable measures most of the time.

The Incremental Optical Position Sensor (IOPS) provides an analog signal build with a variable frequency almost harmonic signal. Each period represent a defined and precise angular displacement that is sum up to create an pseudo-absolute position. In addition, the period of this signal provides the angular speed of the system in a simple manner without the need to operate an derivation of the position as in the case of the resolver. For this thesis, the IOPS signals are processed and are providing position and speed information to the sensors fusion part for each of the two channels.

A last source of position and speed could be extracted from the shape of the motor currents giving a rough idea of the rotor position. As the processing of the currents to obtain this information is a large subject on this own, this option has been left for further work. In addition, the four sources are already providing a very good level of confidence for the target application.

Fig.10.1 shows on the same plot the four positions (top plot) and the four speed (bottom plot) associated with the four sources of information described earlier

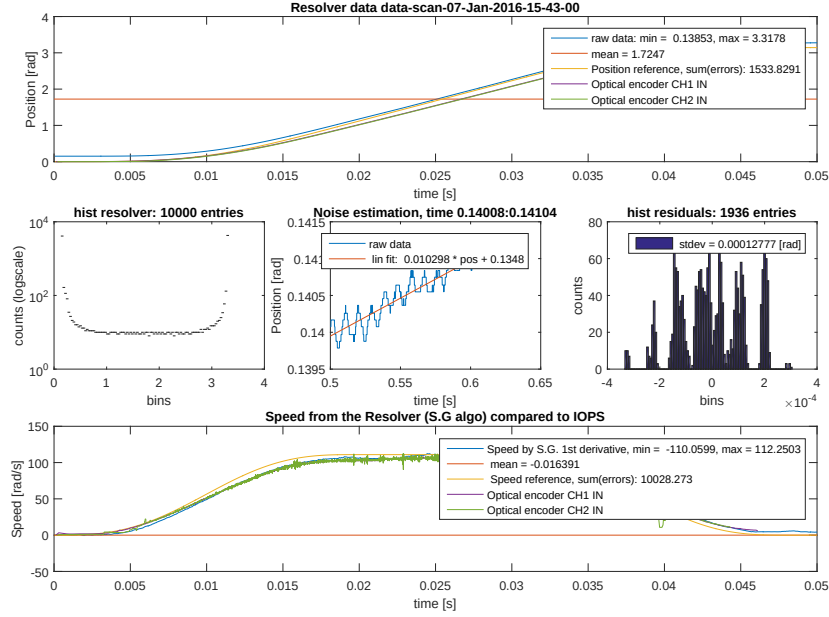


Figure 10.1: Positions (top plot) and speeds (bottom plot) provided by the reference look-up tables, the resolver and the two IOPS sensors that are used in the sensors fusion

10.1 Algorithm design

The algorithm for the sensors fusion is aiming to provide in a simple manner the best guess of a value based on four available values. The implemented approach to solve this challenge is a simple algorithm combining sorting and averaging. Every time an new value is made available at one of the inputs, the algorithm is sorting the four inputs in an ascending manner to exclude the two extreme values. The two remaining ones situated between the two extremes are used to create the output with will be the average between the two.. To monitor this process, the difference of the two remaining values is also computed and compared with predefined limits to generate a status. This status provides a sort of quality check of the generated output.

Fig.10.2 presents the principle of the algorithm selected for this work. This data driven approach proposes to perform exclusion of extremes values and averaging of the two remaining to create the output.

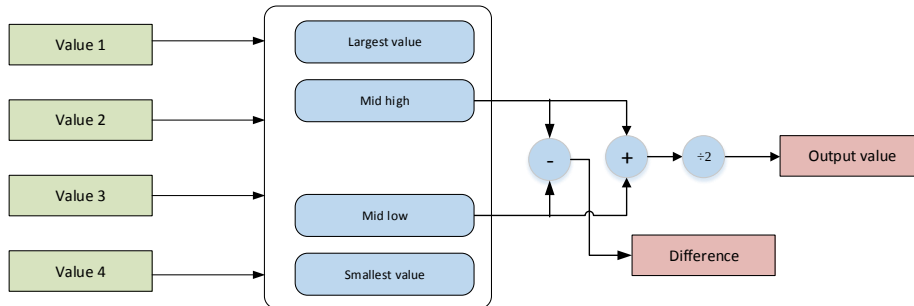


Figure 10.2: Data driven sensors fusion

10.2 Architecture

There are two types of information to fusion, the position and speed of the system. For each one, four sources of information are available. The two process are sharing the same processing bloc visible on the top of Fig.10.3 with the reference 'P510', only the input and outputs are different. In addition, an independent unit 'P530' is dedicated to the fusion of the status that are generated by the individual pre-processing treatment of each sensor. The blue part as been implemented while the white part is foreseen for future work.

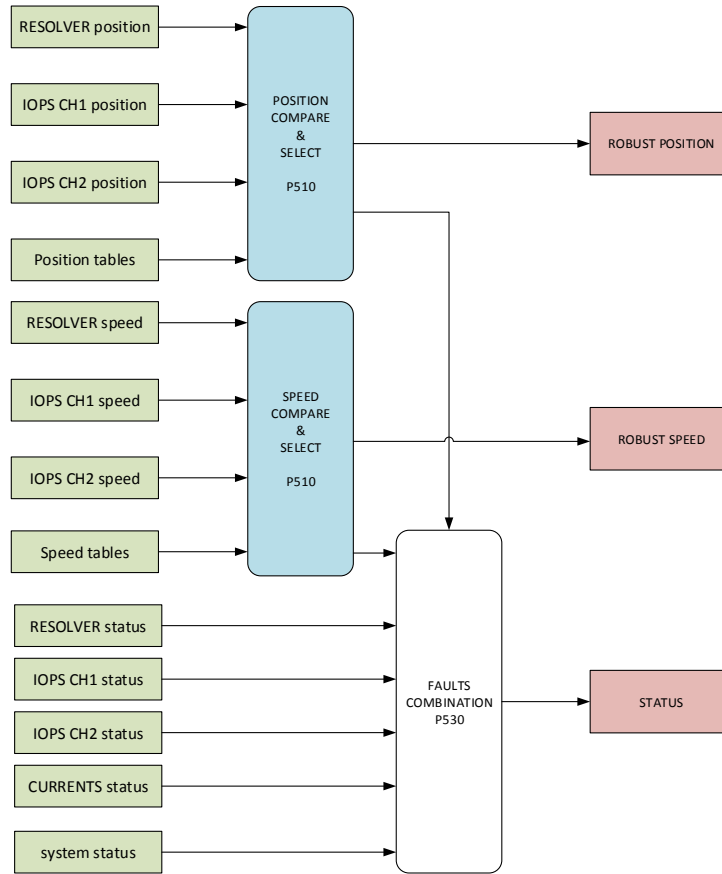


Figure 10.3: Sensors combination processing

Fig.10.4 details the processing bloc 'P510' responsible of the fusion of the four inputs. this schema is similar to the one on Fig.10.2 to the exception of the bloc added to monitor the difference of the two values used to calculate the output. This generated status should be used to judge on the relative quality of the output value, i.e. on how close the selected values are from each other.

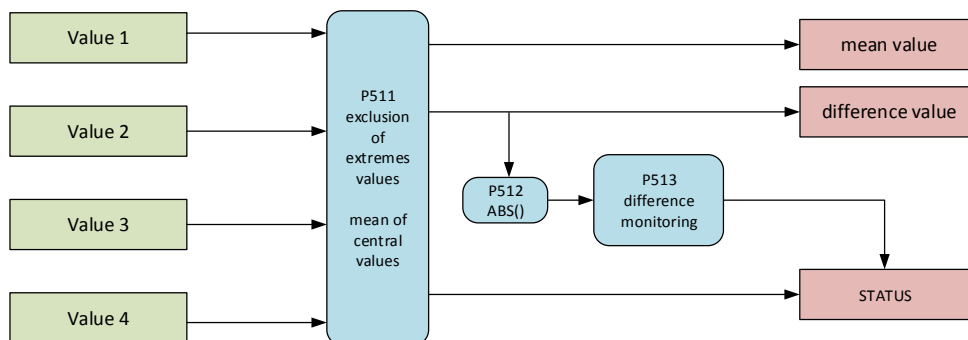


Figure 10.4: Sensors fusion details on values selection

10.3 VHDL Implementation

The implementation of this processing uses as input and output the interfaces developed during this thesis 'sensor-processed-type' which are simplifying a lot the interconnections between entities.. Fig.10.5 shows the top entity of the sensors fusion with these interfaces

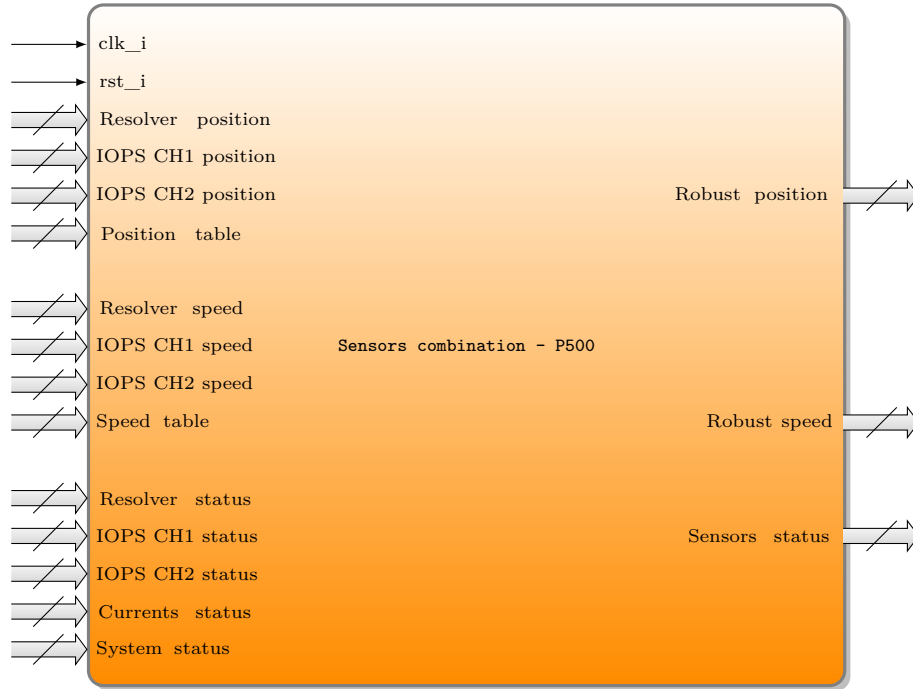


Figure 10.5: Top entity of the fusion sensor processing

The four sensors interfaces for each physical value (position, speed) are based on the fixed point representation, the algorithm sort them to extract the two middle values. To perform this operation efficiently inside the FPGA, a similar topology that the median filter developed earlier is used. Fig.10.6 present this pipelined architecture of sorting cells. From top to bottom, the four values are sorted from smallest (left) to highest (right) and the two middle values are then sum and subtracted, to calculate the mean and difference that will be output of the entity.

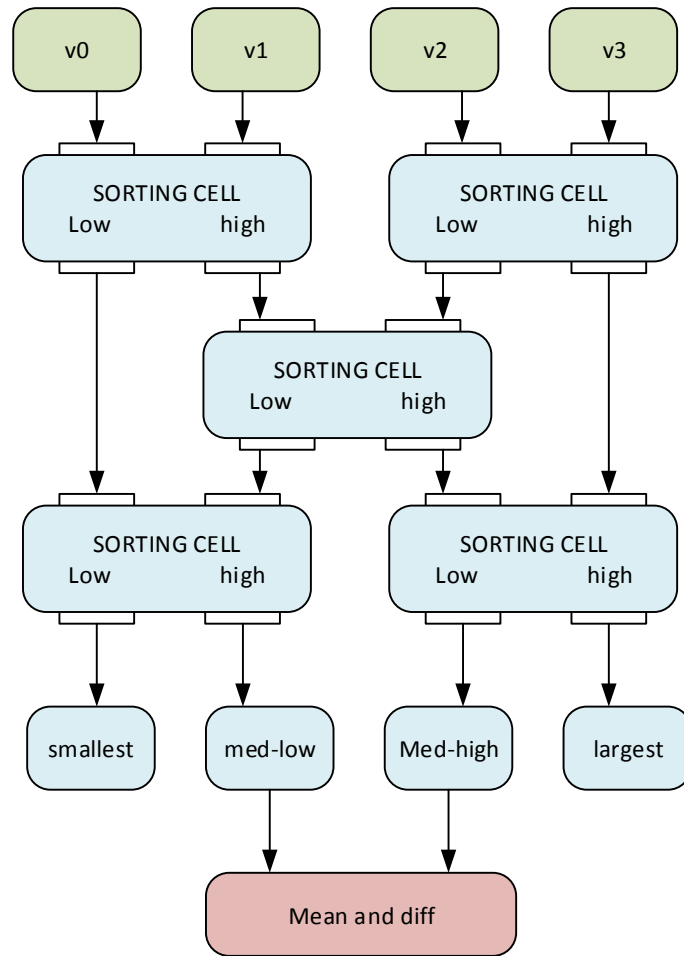


Figure 10.6: Sensors fusion details on the values sorting process

10.4 Verification

The verification of the fusion functionality has been performed directly with the top sensors processing in Chap.11. This approach allows to have directly on its inputs the exact stimulus. As the raw sensors data from the 'source-data-raw' are firstly processed by each individual sensor treatment algorithm, prior to their arrival on the fusion part.

Chapter 11

Top sensors processing

The complete processing module developed for this thesis contains all the necessary modules to pre-process each sensors and to combine these information. Th output provides monitored values, status and robust information of the position and speed of the system.

Fig.11.1 shows the final top level of the data processing for the three sensors. The input stream from the sensors are represented in green (left) while the outputs are in red (right). The processing itself is in blue (centre). This top hierarchical entity is a wrap-up of all previously developed components.

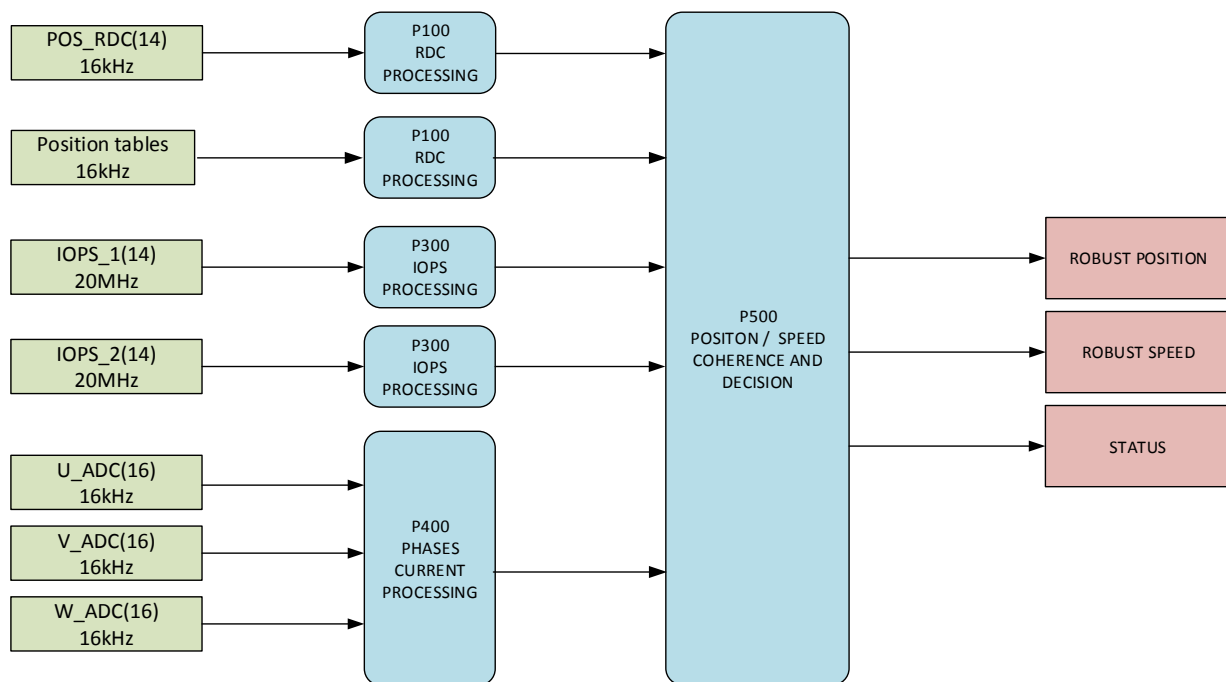


Figure 11.1: Overview of the top sensors processing internal components

The top hierarchical entity is a self contained component. Fig.11.2 shows its input and output ports to connect it with the sensors (left side) and to further processing and storage (right side).

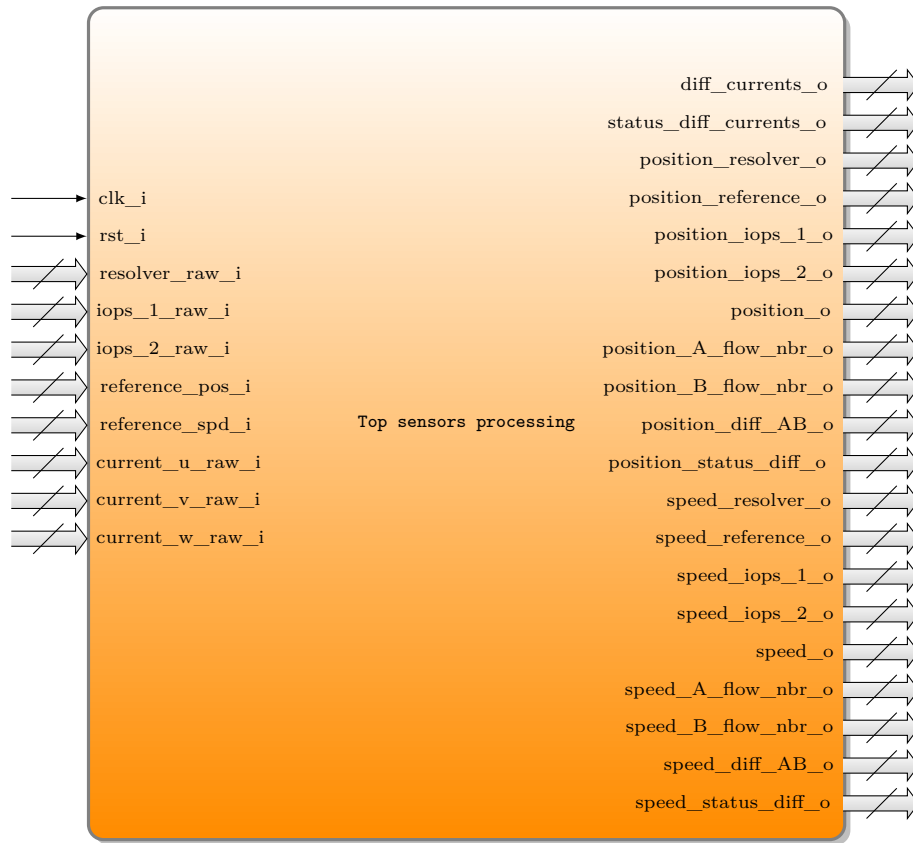


Figure 11.2: The top sensors processing component

11.1 Verification

The verification of the global sensors processing was performed using stimuli generation from real data sets. The outputs were compared to ones generated by the prototyped algorithm written with Matlab.

To realise this, the top entity 'top-sensors-processing' was connected to the field data generator 'source-data-raw' and the outputs were saved into individual text file. The test bench entity description below shows that many generics parameters are available. This is to be able to simulate the design for multiple set of input data without having to recompile it. The 'OUTFOLDER' and 'FILE-PREFIX' generics are allowing to save multiple results set into separated folders and with different names.

```
entity top_sensors_processing_tb is
  generic(
    OUTFOLDER      : STRING;
    FILE_PREFIX    : STRING;
    IOPS_1_DATA    : STRING := "data_in_scan_ops_1_07_Jan_2016_15_43_32.hex";
    IOPS_2_DATA    : STRING := "data_in_scan_ops_2_07_Jan_2016_15_43_32.hex";
    IOPS_DEPTH     : integer;
    RESOLVER_DATA  : STRING;
    POSITION_DATA   : STRING;
    POS_DEPTH      : integer;
    CURR_U_DATA    : STRING;
    CURR_V_DATA    : STRING;
    CURR_W_DATA    : STRING;
    CURR_DEPTH     : integer;
    FIXPT_SIZE_HIGH : integer := 16;
    FIXPT_SIZE_LOW  : integer := -16;
    DIFF_MEAN_TO_THRESHOLD : integer := 300;
    INITIAL_MEAN_VALUE : integer := 1000
  );
end top_sensors_processing_tb;
```

Fig.11.3 and Fig.11.4 are showing the VHDL simulation of this component using Modelsim. Each of the visible output represented in a pseudo analog way are automatically saved into files for later visualisation and comparison.

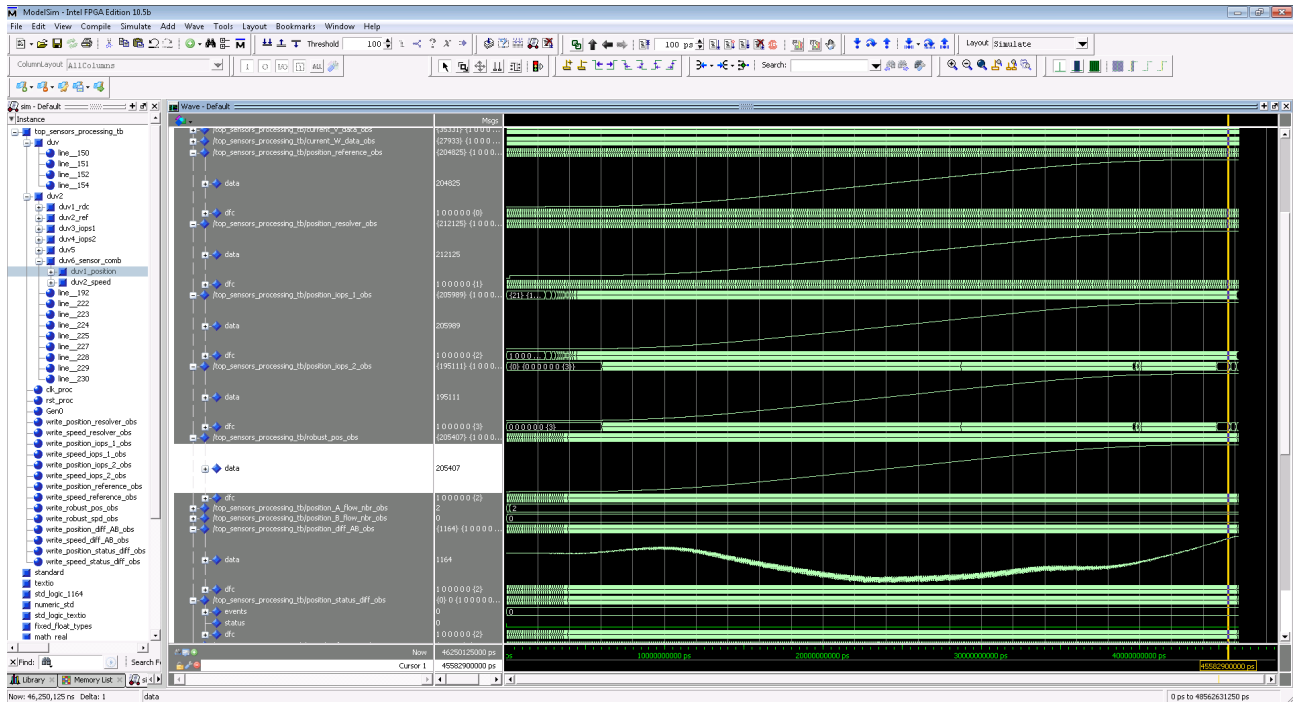


Figure 11.3: Modelsim top sensors processing - position - using S4 dataset: 5.1

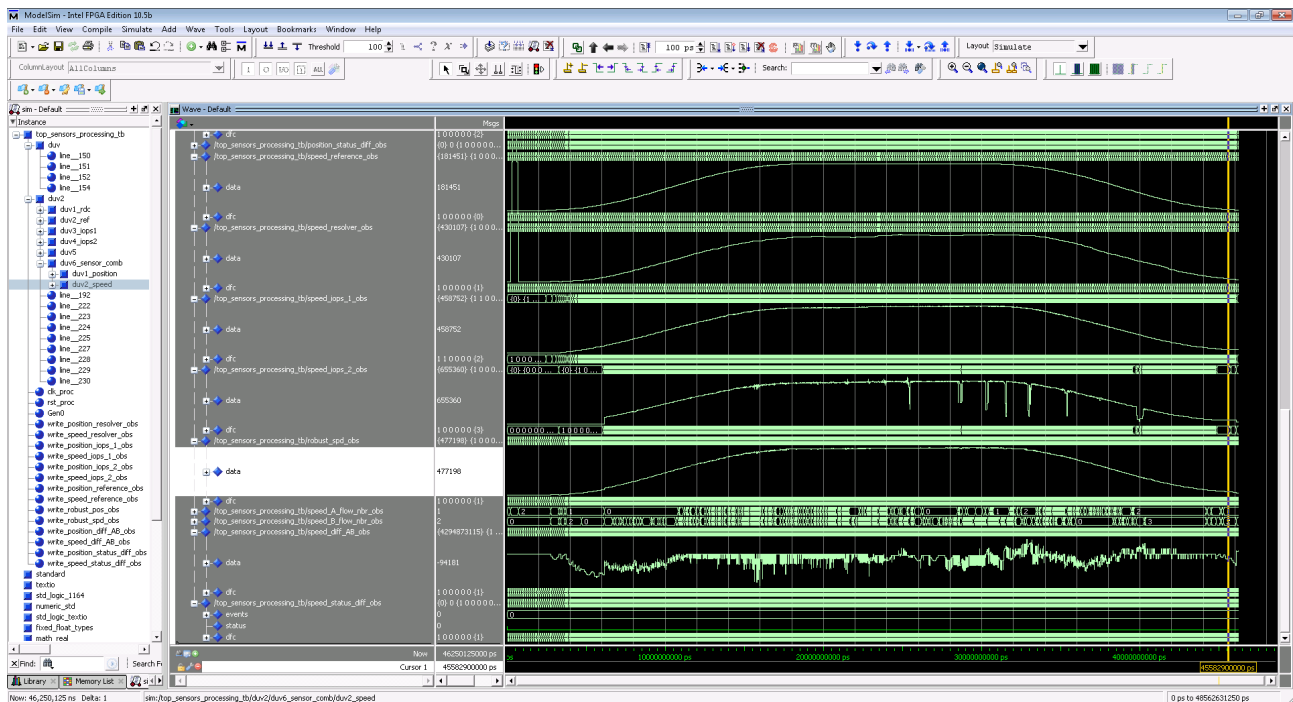


Figure 11.4: Modelsim top sensors processing - speed - using S4 dataset: 5.1

Fig.11.5 and Fig.11.6 are showing the position and speed generated with the Matlab and the VHDL algorithms. The four sources are plot on top and the robust output at the bottom of the figures. Without being exactly the same, the comparison presents remarkable similarities between the two versions.

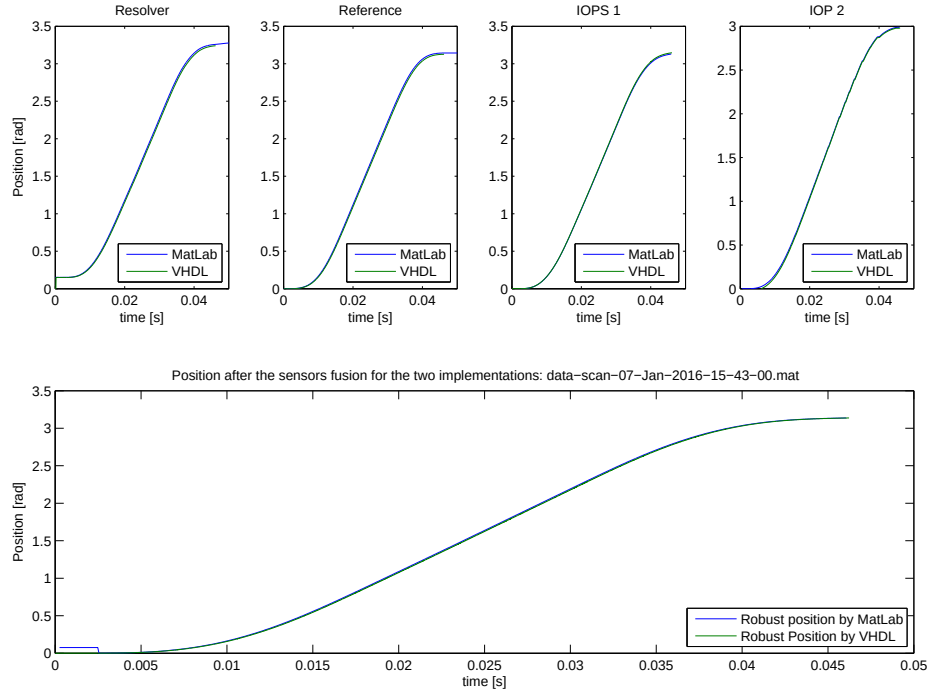


Figure 11.5: Comparison of the top sensors processing output position, for the two algorithm implementation (Matlab and VHDL)- using S4 dataset: 5.1

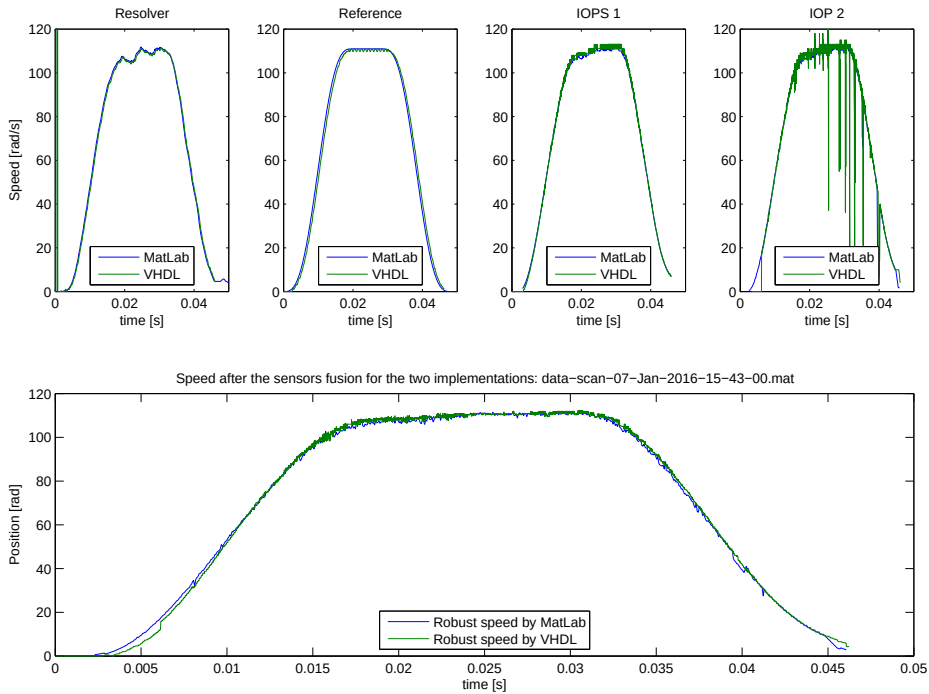


Figure 11.6: Comparison of the top sensors processing output speed, for the two algorithm implementation (Matlab and VHDL)- using S4 dataset: 5.1

11.1.1 Failure cases simulations

Additional comparisons were performed using simulated failures defined in Table 5.3. Fig.11.7 and Fig.11.8 are showing the comparison for inputs simulating the failure of the resolver defined as I01-FM4.

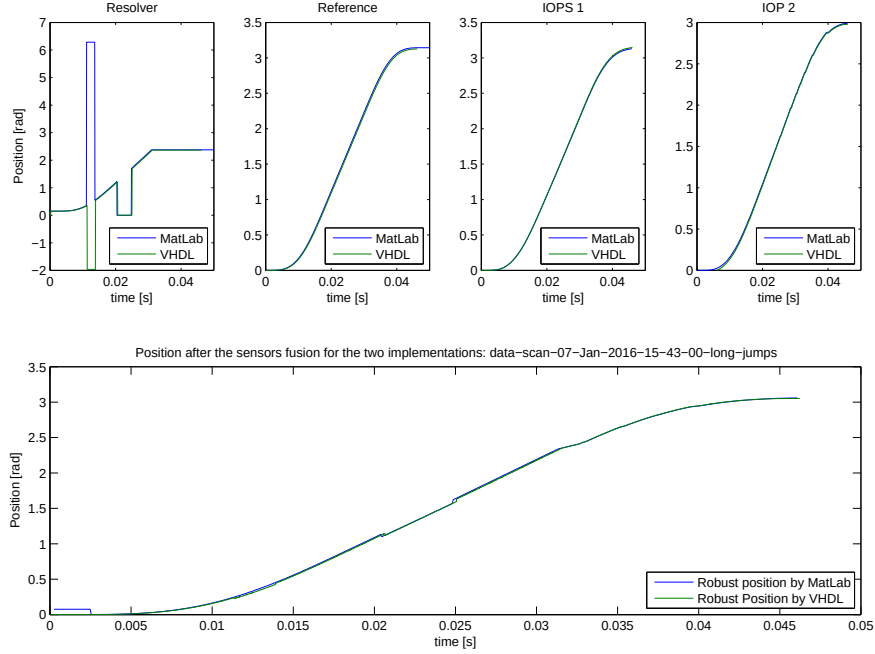


Figure 11.7: Comparison of the top sensors processing output position, for the two algorithm implementation (Matlab and VHDL) - using S4 dataset modified for the failure mode I01-FM4: 5.3

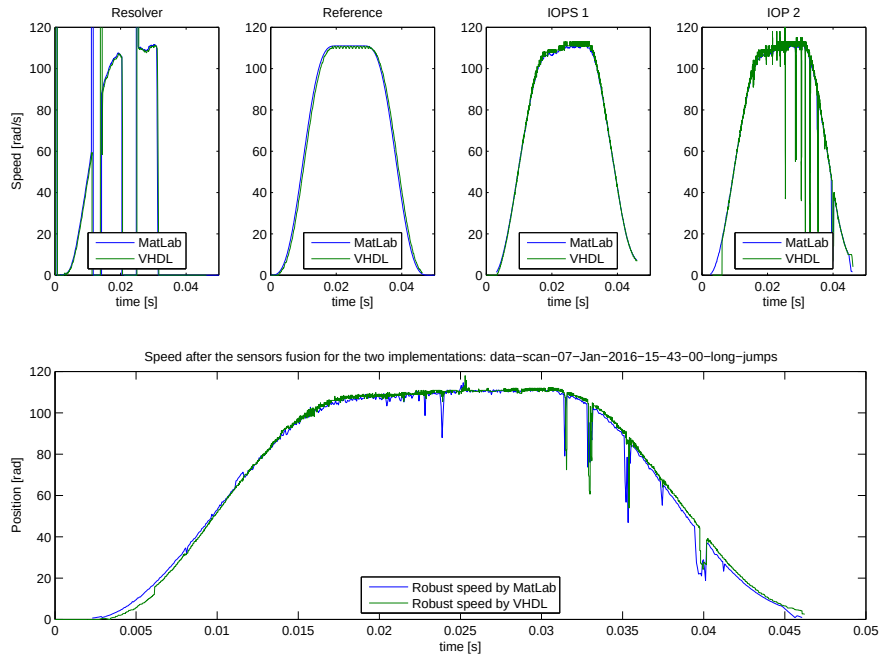


Figure 11.8: Comparison of the top sensors processing output speed, for the two algorithm implementation (Matlab and VHDL)- using S4 dataset modified for the failure mode I01-FM4: 5.3

Fig.11.9 and Fig.11.10 are showing the VHDL simulation for the IOPS failure modes I02-FM4 and I03-FM4.

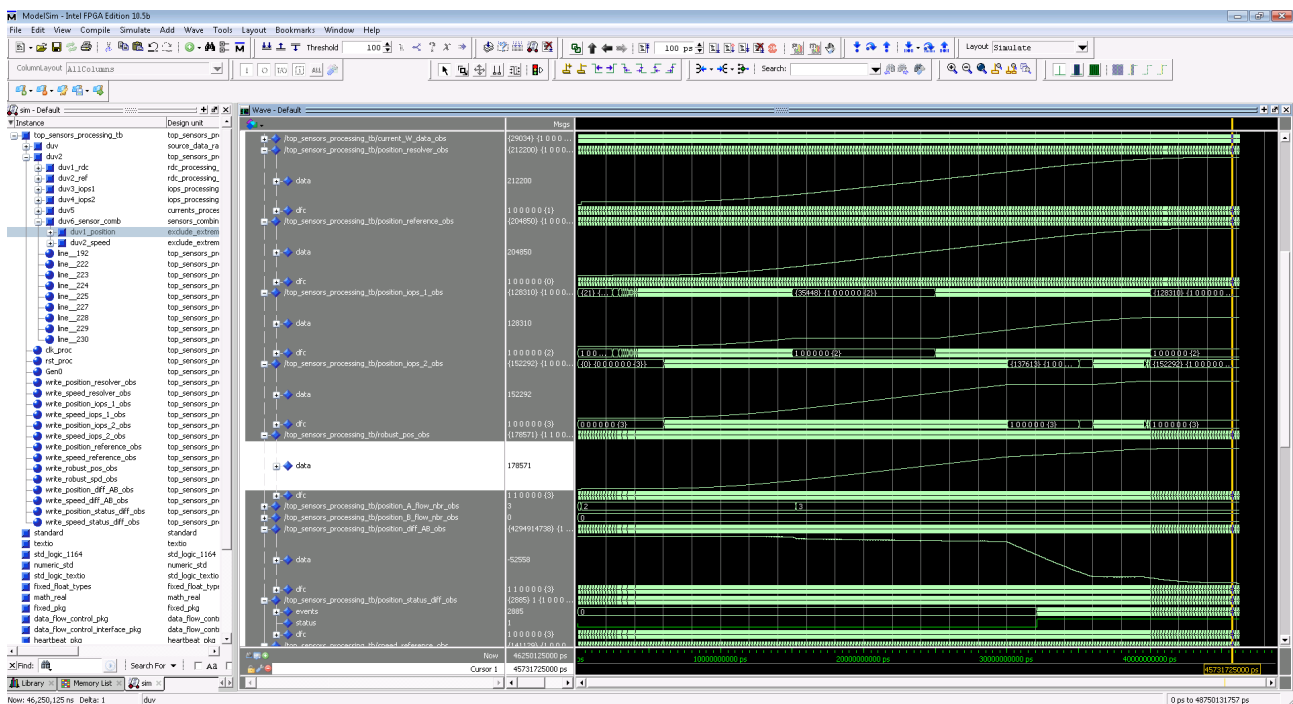


Figure 11.9: Modelsim top sensors processing - position - using S4 dataset modified (see bottom plot of 5.7 and 5.7) to test the failure mode I02-FM4 and I03-FM4: 5.3

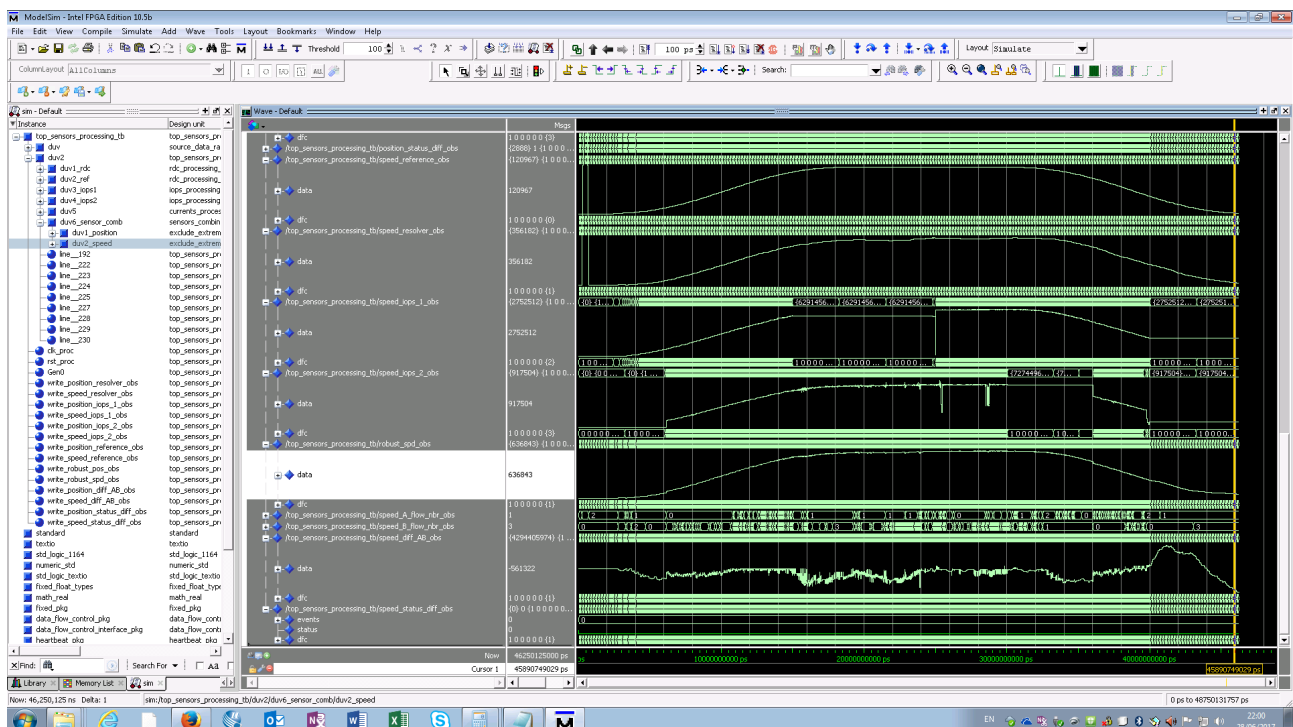


Figure 11.10: Modelsim top sensors processing - speed - using S4 dataset modified (see bottom plot of 5.7 and 5.7) to test the failure mode I02-FM4 and I03-FM4: 5.3

11.2 further work

The simulations are showing very similar results for the two algorithms implementations. Some tuning could allow them to fit perfectly. This conclude the primary objective of this thesis which was to give to the BWS, monitoring and detection capability for abnormal behaviour of its sensors.

For the second objective, the improvement of the fusion system compared to single sensor architecture is confirmed. Nevertheless, future project could be dedicated to improve the capability of this system against combined sensors failure. There are many other ways to fusion multi-sensor data, such as the voting, statistical approaches or even neural network based [27]. These have the potential to improve the quality and robustness of the system even further.

Chapter 12

Common components and implementation summary

This chapter combine the description of key components used in the VHDL implementation and summaries the results of the synthesis and performance of the complete design.

12.1 Common components

12.1.1 data flow control and standard interface between components

One of the challenges in VHDL is to manage the complexity of interfaces between numerous components when dealing with largely hierarchical design. In addition to the initial design, any changes to control signals oblige to modify multiple files into the hierarchy. One answer to this problem is the creation of bundle of signals through the use of VHDL 'record' structure. This type of structure allows to create a list of signals of different nature that are only declared in one location. In this work, a package called 'data-flow-control-pkg' has been created to store the list of control signals between components.

The 'dfc-type', for data flow control is defined as

```
type dfc_type is record
    flow_enabled      : std_logic;    -- The channel is open and can transport new data at anytime.
    value_updated     : std_logic;    -- A updated value is available, this acts as 'clock enable' for next register.
    sensor_overflow    : std_logic;    -- The analog source is saturated, for exemple out of range of the ADC.
    sfixed_overflow    : std_logic;    -- Overflow during a calculation somewhere in this channel with sfixed number.
    value_warning      : std_logic;    -- The data being transmitted is suspicious, could be partially corrupted.
    value_error        : std_logic;    -- The data being transmitted is known to be wrong.
    identifier         : std_logic_vector(3 downto 0); -- data flow number used to identify the source of the information.
end record;
```

A second package called 'data-flow-control-interface-pkg' is defined to centralise the complete interface, to include the data and the flow control together. For example, the interface holding raw data from the resolver or the iops sensor is:

```
type sensor_raw_type is record
    data : std_logic_vector(13 downto 0);
    dfc : dfc_type;
end record;
```

and the one holding the processed data having a physical unit, such a radians for the position, is represented in signed fixed-point (sfixed) and manipulated as std-logic-vector is

```
type sensor_processed_type is record
    data : std_logic_vector(31 downto 0);
    dfc : dfc_type;
end record;
```

Fig.12.1 illustrate these interface with an example which has an input of type 'sensor-raw-type' and an ouptput 'sensor-processed-type'. The signals inside the interfaces are listed in purpose with a name representing the way to access them in the structure (with '.').

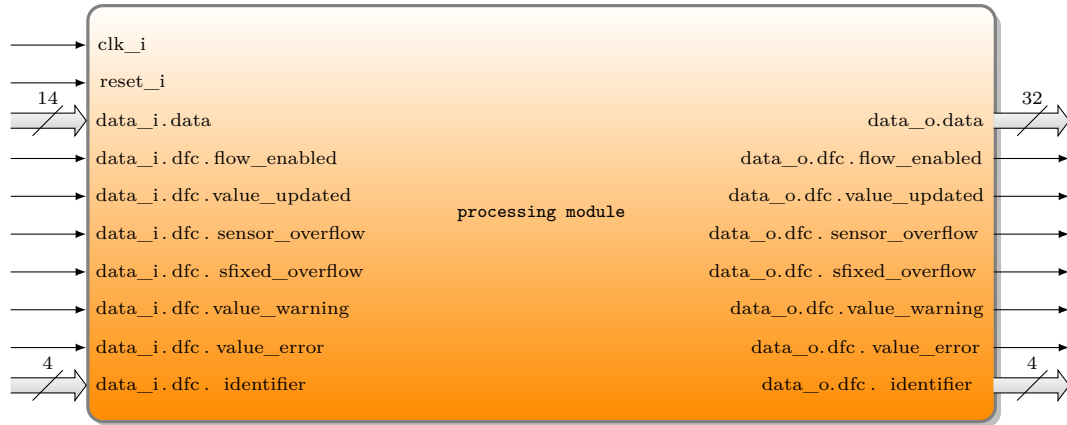


Figure 12.1: Processing module using the interface from the packages 'data-flow-control-pkg' and 'data-flow-control-interface-pkg'. The input is of type 'sensor-raw-type' and the ouptput 'sensor-processed-type'

12.1.2 Convolution core for filtering

The convolution is used multiple times in this development and to make it versatile to develop a generic convolution core. The challenge is to create code for a generic size of coefficients. Meaning that the depth of the FIFO holding the incoming data, the multiplication between the delayed data and the coefficient and the final sum is of variable size. To set the coefficients, since the tools can handle unconstrained generic array, the solution was to have the coefficient as an port array of integer that can have its size as generic.



Figure 12.2: Convolution module for FIR filter type processing

```
entity convolution_sfixed_data_generic_size is
  generic(
    FIXPT_SIZE_HIGH      : integer := 16;
    FIXPT_SIZE_LOW       : integer := -16;
    CONV_COEF_NUMBER     : integer := 9;          -- Number of coefficient , which is also the depth of the convolution
    CONV_COEF_NORMALISE : real := 60.0/16129.0    -- Convolution result will be multiplied by 1 / CONV_COEF_NORMALISE
  )
  port(
    clk_i      : in  std_logic;
    rst_i      : in  std_logic;
    conv_coef  : in  array_of_integer(0 to CONV_COEF_NUMBER-1);
    data_i     : in  sensor_processed_type;
    data_o     : out sensor_processed_type
  );
end convolution_sfixed_data_generic_size;
```

The defaults are predefined to implement the Savitzky–Golay filter configured to provide the 1st derivative of the signal and smoothing the data. conversion factor = 6.2832, coefficients vector = [4 3 2 1 0 -1 -2 -3 -4], normalisation = 60.0/16129.0,

https://en.wikipedia.org/wiki/Savitzky–Golay_filter

The discrete convolution product is defined as

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[n-m] \cdot g[m] \quad (12.1)$$

In our case, we have a finite length of the sequence between Z^{-c} until Z^0 , 'c' being the number of coefficients of the convolution.

$$(f * g)[n] = \sum_{m=-c}^0 f[n-m] \cdot g[m] \quad (12.2)$$

In fact, this filter is nothing else than a Finite Impulse Response (FIR) filter than can be written as

$$y[n] = \sum_{i=0}^N b_i \cdot x[n-i] \quad (12.3)$$

And its implementation in hardware is presented on Fig.12.3. It is this form which is implemented in the convolution core.

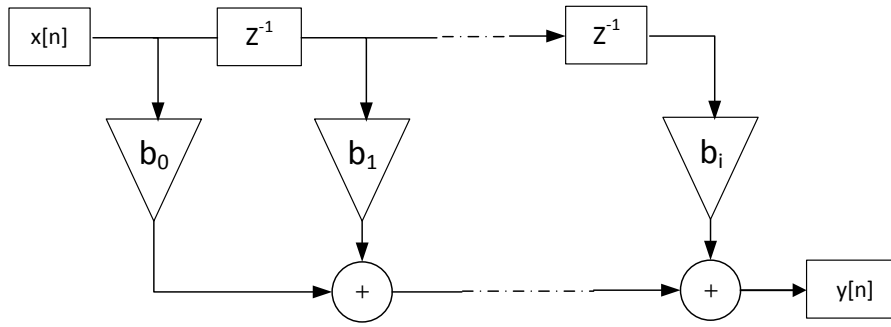


Figure 12.3: Topology of Direct form Finite Impulse Response (FIR) filter, used for discrete convolution of an input stream

For the synthesis, the first functional implementation was using large combinational logic as most of the calculations were stored in variables and thus synthesised as combinational logic. The result was a core which could not run at more than 30MHz having 9 parameters (creating 9 multiplication and sums in series). The second version is pipelined by using the signals instead of variables. The maximum frequency has been increased to more than double, up to 70MHz for 9 coefficient which is enough for the target design.

The verification of this function has been performed using a dedicated VHDL test bench called:

`convolution_sfixed_data_generic_size_tb.vhd`

The device under test (dvt) is stimulated with a variable input. The output is compared with one value calculated by the test bench and in case of difference, the log is showing an error message along with details. In addition, the input-output are stored in a text file for eventual later use. Fig.12.4 shows the chronographs of this verification where we can see the input to the dvt 'pos-sfixed-obs', which is a position in radians, and the output 'conv-processed-obs' which is the estimation of the speed in radians per second.

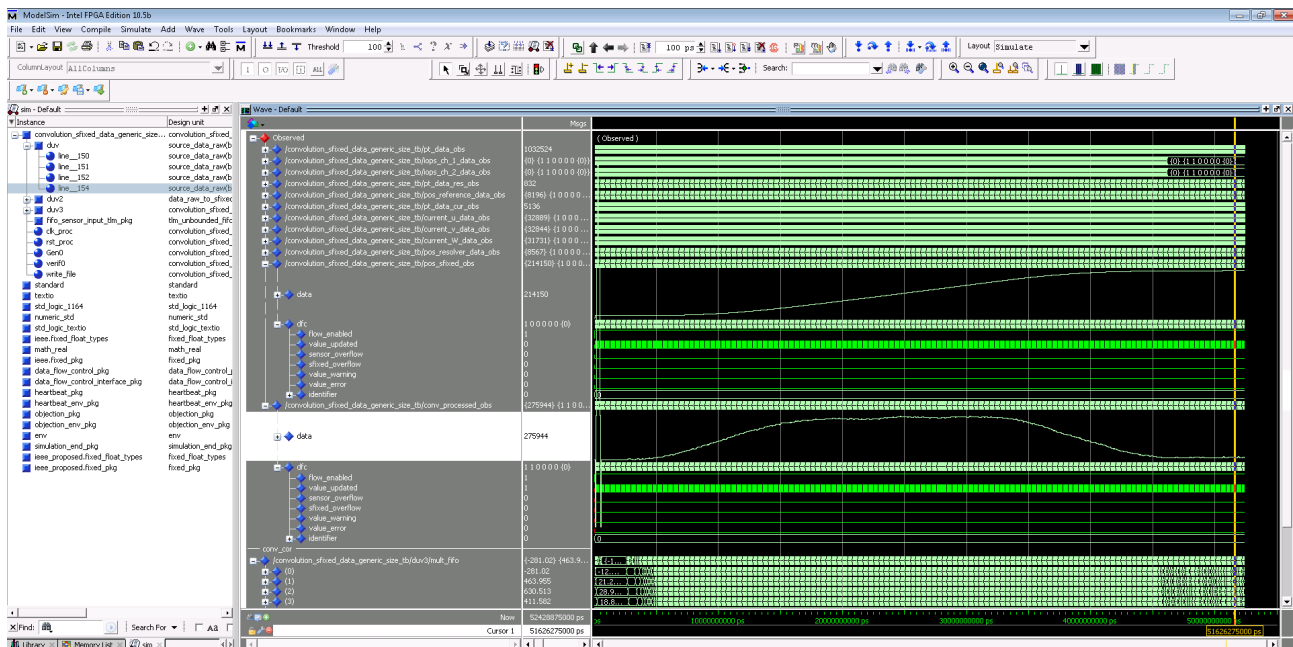


Figure 12.4: Convolution verification

12.1.3 median filter for raw and sfixed data

The median filter is a very powerful tool in signal and image processing. It allows the exclusion of extremes values of a stream of data or an image. It is none-linear, i.e. it is not reversible due to the loss of information. This filter is often used to remove noise in an efficient manner. This filter is often used in image processing to remove a noise introduced by defected pixels (noise known as salt-and-pepper). For our application, this filter removes abnormal noise, introduced by digital transmission errors between chips for example, or simply extreme values of unknown source.

The entity description of this filter for raw data is:

```
entity median_filter_raw_data is
  generic(
    FIXPT_SIZE_HIGH      : integer := 16;
    FIXPT_SIZE_LOW       : integer := -16
  );
  port(
    clk_i      : in    std_logic;
    rst_i      : in    std_logic;
    data_i     : in    sensor_raw_type;
    data_o     : out   sensor_raw_type;
    diff_o     : out   sensor_processed_type -- output the difference between incoming signal and outcoming
  );
end median_filter_raw_data;
```

And for the version using sfixed representation:

```
entity median_filter_processed_data is
  generic(
    FIXPT_SIZE_HIGH      : integer := 16;
    FIXPT_SIZE_LOW       : integer := -16
  );
  port(
    clk_i      : in    std_logic;
    rst_i      : in    std_logic;
    data_i     : in    sensor_processed_type;
    data_o     : out   sensor_processed_type;
    diff_o     : out   sensor_processed_type
  );
end median_filter_processed_data;
```

The principle of a median filter is to store a defined number of past data and sort them in an ascending manner. The middle value will be output. As a new data arrives in the process, the oldest is removed and the calculation is performed again.

The challenge in the hardware implementation is to sort a vector of values with the minimum hardware. An elegant version of this is the systolic array for sorting, 7 elements in our case. this type of array is a network of small data processing unit which are computing part of the result. Coupling these units in different stages allows to pipeline the calculation and obtaining a decent operating frequency compatible with the incoming data.

Fig.12.5 present to topology of this systolic array for the calculation of the median value between seven values.

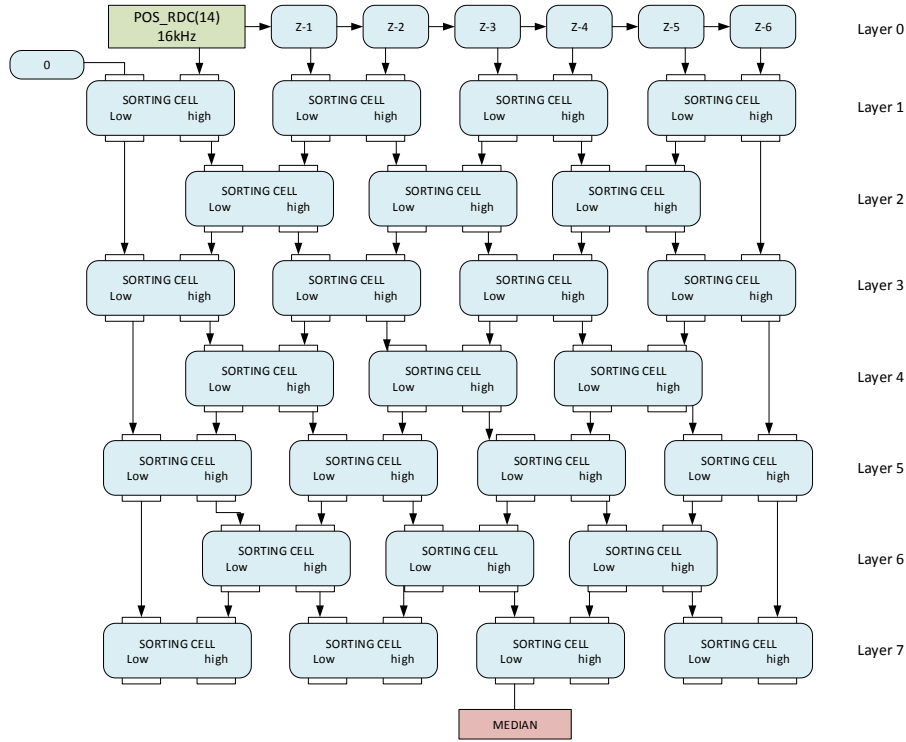


Figure 12.5: Median filtering scheme with 7 layer

Additionally to the median calculation, the difference between the incoming data and the output for this particular data is calculated. In case of rejection of a large or very small value compared to the others, this output will be very large. By comparing it against predefined threshold, it is possible to detect problems in the data flow. Fig. 12.6 shown the implementation of this input filtering and monitoring using the median filter. To be noted that the core 'median filer' contains only the actual filter and the subtraction. The other operations are performed with different components.

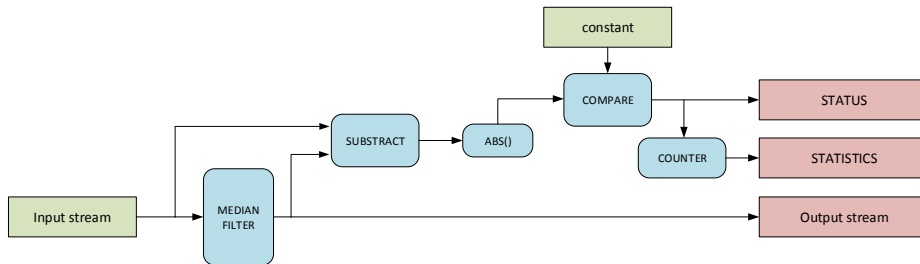


Figure 12.6: Input stream filtering and monitoring using an median filter

The verification of this function has been performed using a dedicated VHDL test bench called:

```
median_filter_processed_data_tb.vhd    -- for duv with sfixed representation
Median_filter_raw_data_tb.vhd         -- for duv with raw binary representation
```

The device under test (duv) is stimulated with a variable input. The output is compared with one value calculated by the test bench and in case of difference, the log is showing an error message along with details. Fig. 12.7 shows the chronograms of this verification for few input signals. To note the influence of the pipeline of the calculation which delays the output.

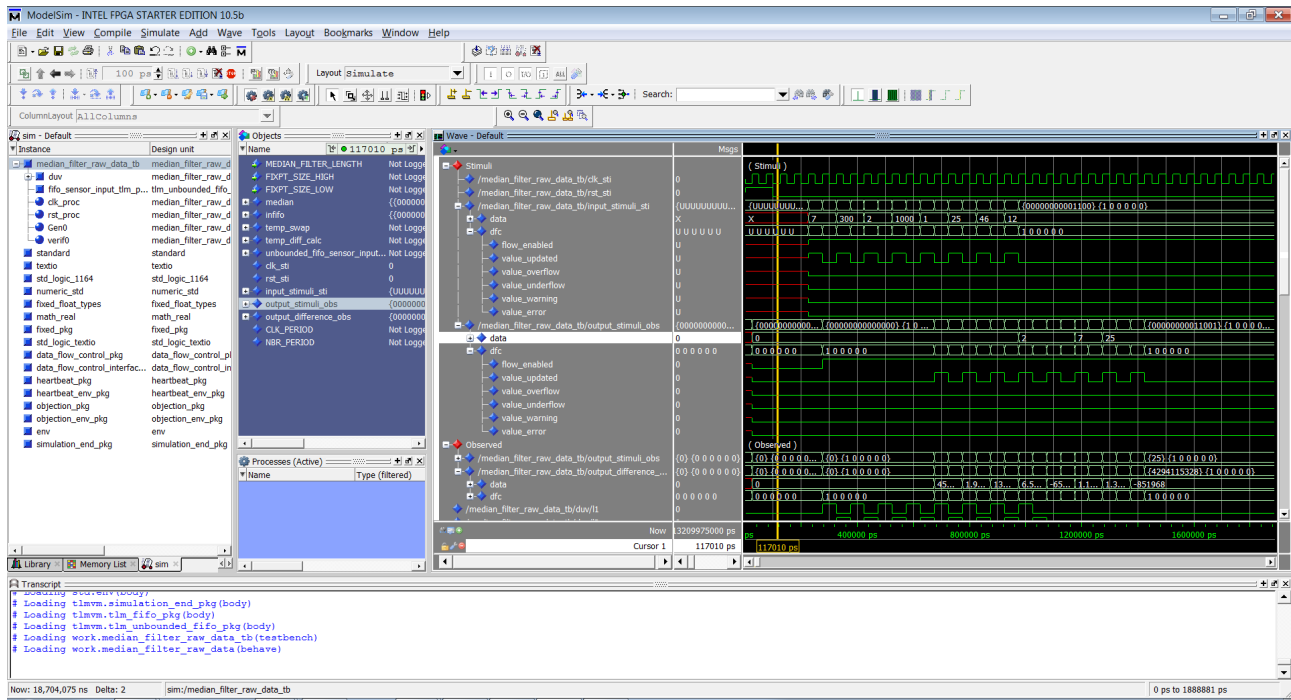


Figure 12.7: Median filter block verification results

Fig.12.8 is also a verification sequence where we see the rejection of extreme values. the input-stimuli-sti presents a continuous offset polluted with jumps which are filtered out by the median filter 'output-stimuli-obs'. The difference output named 'output-difference-obs' shows the result of the subtraction between input and output represented with sfixed.

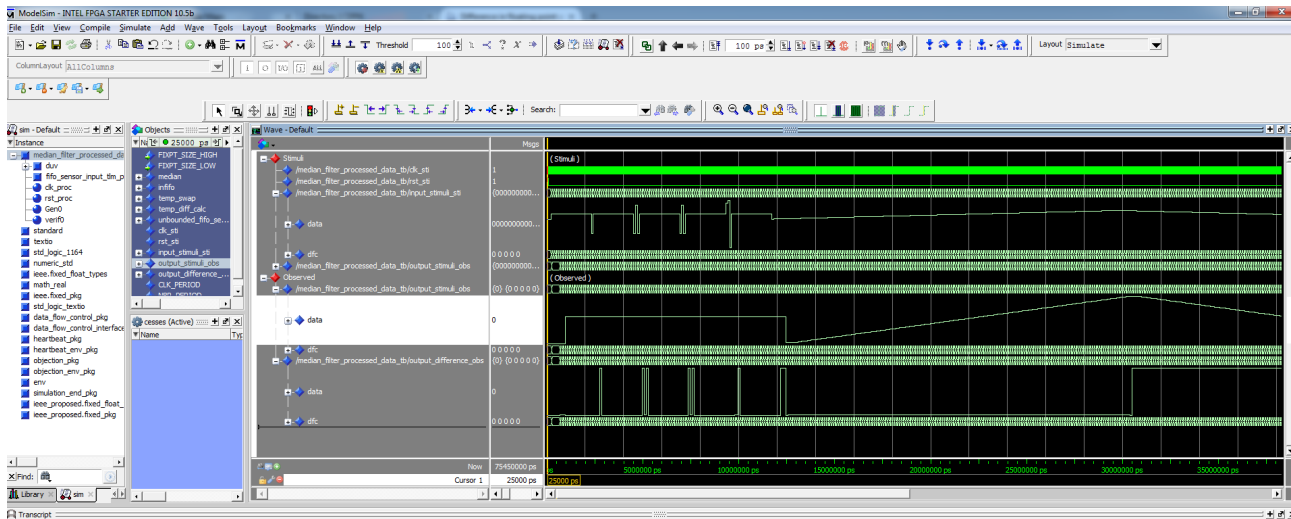


Figure 12.8: Median filter block verification results

12.1.4 Monitoring of raw and sfixed values

The monitoring of the data flow is performed with this component. The connected data is monitored by comparing it to a predefined threshold defined in the generic part of the component. Every time the value goes above this level the status output goes high and a register called 'events' is incremented. The status and the value of the event register is output through the use of status-type record defined in the 'data-flow-control-interface-pkg' package.

```
type status_type is record
    events: unsigned(15 downto 0); -- holds how many times the status went high
    status: std_logic;
    dfc : dfc_type;
end record;
```

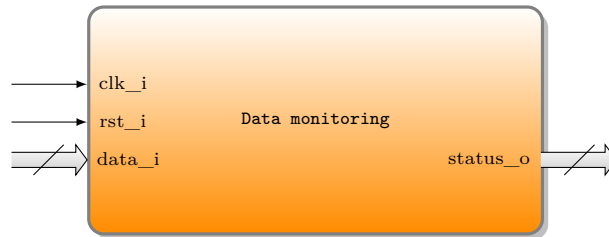


Figure 12.9: Monitoring module

The entity for raw data version is:

```
entity monitoring_raw_sensor_data is
    generic(
        THRESHOLD_STATUS : integer := 1000
    );
    port(
        clk_i      : in  std_logic;
        rst_i      : in  std_logic;
        data_i     : in  sensor_raw_type;
        status_o   : out status_type
    );
end monitoring_raw_sensor_data;
```

And for the version using sfixed representation:

```
entity monitoring_processed_sensor_data is
    generic(
        FIXPT_SIZE_HIGH : integer := 16;
        FIXPT_SIZE_LOW  : integer := -16;
        THRESHOLD_STATUS : real   := 0.0
    );
    port(
        clk_i      : in  std_logic;
        rst_i      : in  std_logic;
        data_i     : in  sensor_processed_type;
        status_o   : out status_type
    );
end monitoring_processed_sensor_data;
```

12.2 Implementation summary

The summary of entities implementation and synthesis is available of the Table 12.1. There is one component per line. Some are occupying multiple lines to details significant improvements made to these modules for the optimisation of the target maximum clocking frequency.

Name	function	test bench	Synthesis
Top entity	top-sensors-processing	file based duv	10413 ALUT, 6838 REG, 48MHz
Top entity final	top-sensors-processing	file based duv	7012 ALUT, 5010 REG, 63MHz
Resolver processing v1	rdc-processing-P100	file based duv	Cyclone IV: 4551 LE, 1343 REG, 12 MHZ
Resolver processing v2	rdc-processing-P100	file based duv	Arria V: 2547 ALUT, 1623 REG, 35 DSP, 30 MHZ
Resolver processing v3	rdc-processing-P100 pipelined convolution	file based duv	Arria V: 3121 ALUT, 1991 REG, 35 DSP, 51 MHZ
rotation-integrator	sum angular position	individual verif	
IOPS processing v1	iops-processing-P300	file based duv	Arria V: 3486/2333 ALM, 2898/2118 REG, 8/15 MHZ
v2	iops-processing-P300 - use LPM-DIVIDE 5 levels pipeline	file based duv	Arria V: 1994 ALM, 2303 REG, 62 MHZ
v3	iops-processing-P300 - use LPM-DIVIDE 10 levels pipeline	file based duv	Arria V: 1924 ALM, 2607 REG, 79 MHZ
threshold-compare-raw-sensor-data	mirrors period detection and pos calc	individual tb	127 ALM, 49 REG
mean-and-period-calc	mean of a period, period and speed calc	file based duv	1417 ALM, 102 REG
store-means-calc-thresholds	3 mean median filter calc high-low thresholds	individual verif	179 ALM, 184 REG
Currents processing	currents-processing-P400	file based duv	Arria V: 1673 ALM, 3160 REG 92 MHZ
raw-current-to-sfixed	from raw to sfixed in amps	file based duv	119 ALM, 63 REG
sum-current-sensor-signals	sum calculation of the 3 currents	file based duv	181 ALM, 117 REG
Sensors fusion	sensors-combination	file based duv	1051 ALUT, 1356 REG
exclude-extremes-and-calc-out	calc the output for the po- sition or speed	verif. using top en- try file base duv	341 ALUT, 535 REG
Individual entities			
data-flow-control-pkg	Package data flow control	-	-
data-flow-control-interface-pkg	Package sensors interfaces	-	-
convolution-sfixed-data-generic-size	smoothing, derivation	individual verif	2067 ALUT, 636 REG, 30 DSP, 30MHz (9 coef)
convolution-sfixed-data-generic-size	smoothing, derivation	individual verif	2622 ALUT, 996 REG, 30 DSP, 70MHz (9 coef)
median-filter-raw-data	exclusion extreme values	individual verif	885 LE, 280MHz / 163 ALM, 735 REG
median-filter-processed-data	exclusion extreme values sfixed I/O		256 ALM, 923 REG
monitoring-processed-sensor-data	monitor value and sum er- rors	individual verif	36 ALM, 23 REG
data-raw-to-sfixed	convert raw to sfixed	in global duv	58 ALM, 25 REG
abs-sensor-processed-data	calc absolute of a sfixed	in global duv	36 ALM, 19 REG
check-overflow-sensor-processed-data	test the overflow detection scheme	individual verif	

Table 12.1: List of implemented components

Module integration into the PSB prototype firmware

[illegible]

To visualise the outputs of the processing module, an new instance of the internal logic analyser called SignalTap was created and integrated in the project. This Quartus project was then compiled until the generation of the binary file for the device programming. The evaluation of the maximum clocking frequency with almost 67 [MHz] is above the target of 50 [MHz] as shown on Fig.13.2.

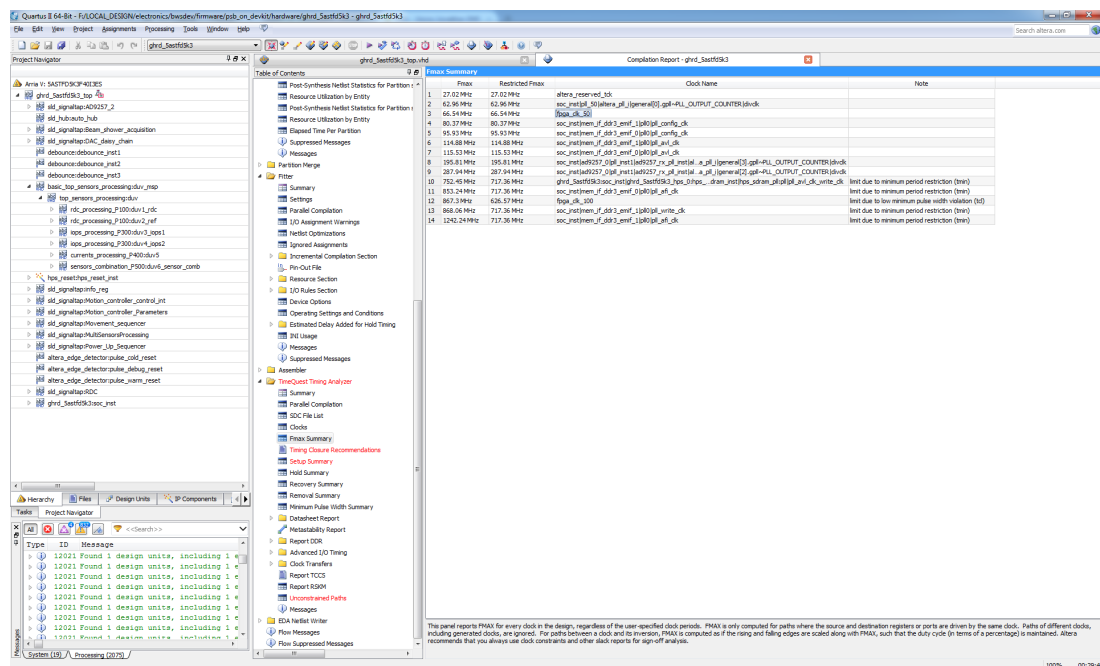


Figure 13.2: Integration of the sensors processing into the operational prototype for the PSB, after optimisation to increase the maximum clocking frequency

The newly compiled firmware was then loaded on the BWS prototype setup inside the motion controller visible under the left table of Fig.13.3. This controller is connect to the electro-mechanical system with copper and optical cables.

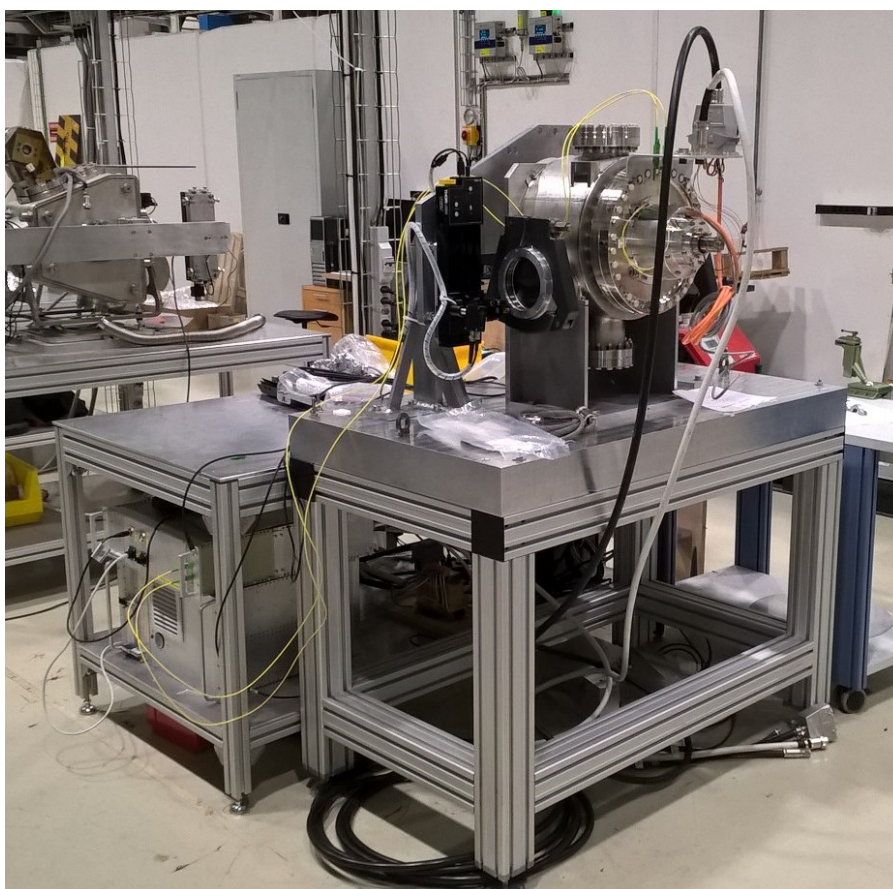


Figure 13.3: BWS prototype setup where the processing module was tested

Few compilations and adjustments were necessary to find the correct synchronisation between the module and the system. In addition, manual manipulations on the existing software to operate this scanner were needed to record the SignalTap screenshot visible on Fig.13.4. Due to the end of the project, there was not enough time to debug this integration and produce more usable results on the physical prototype.

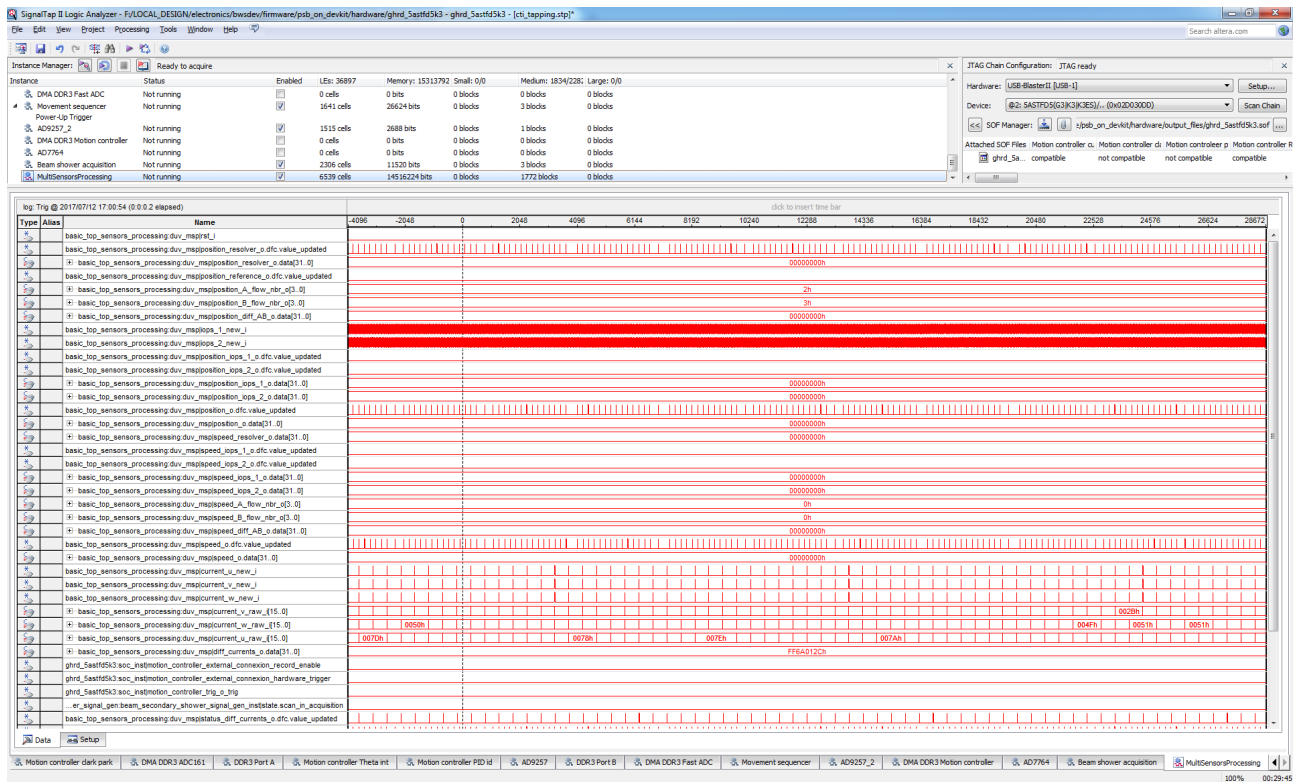
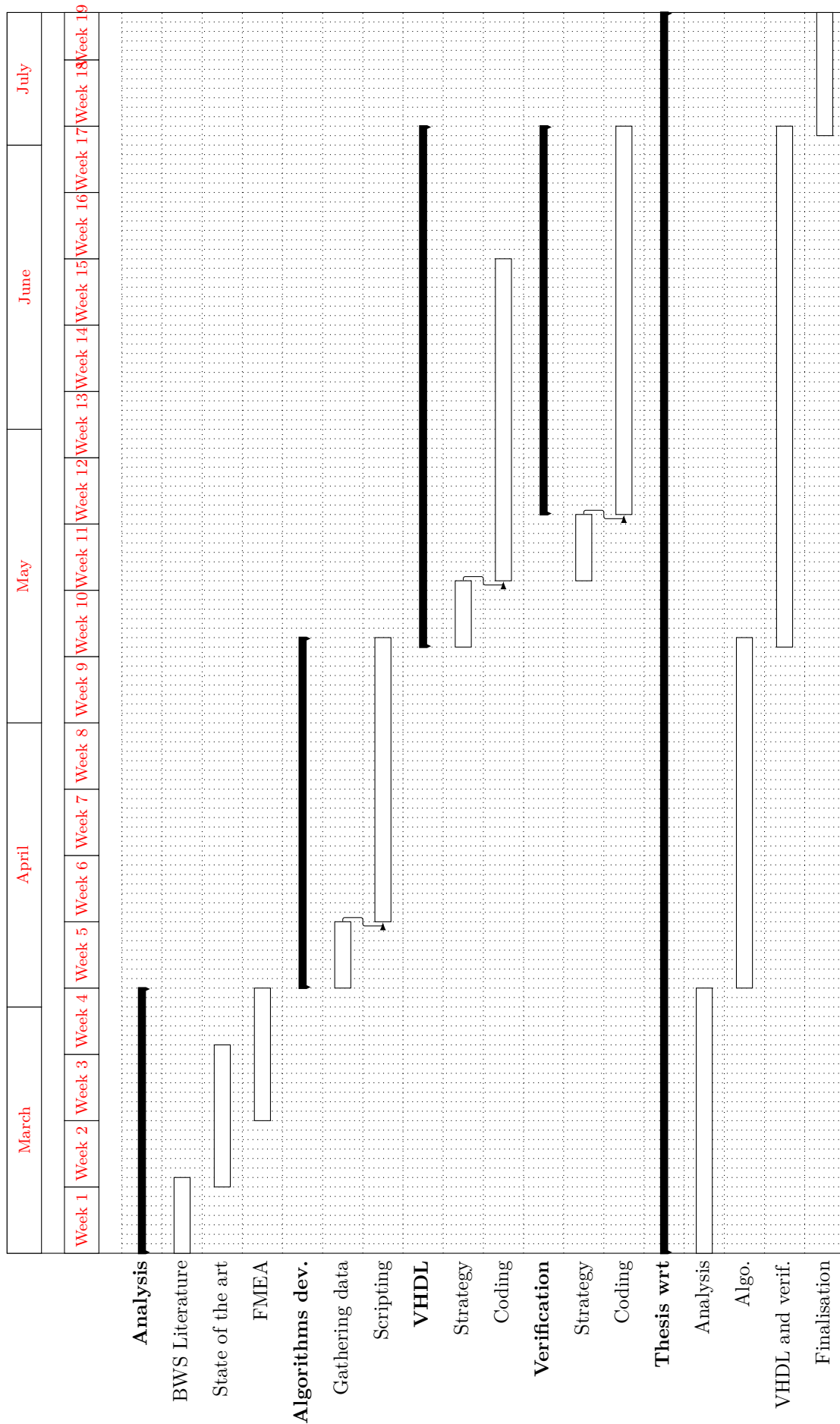


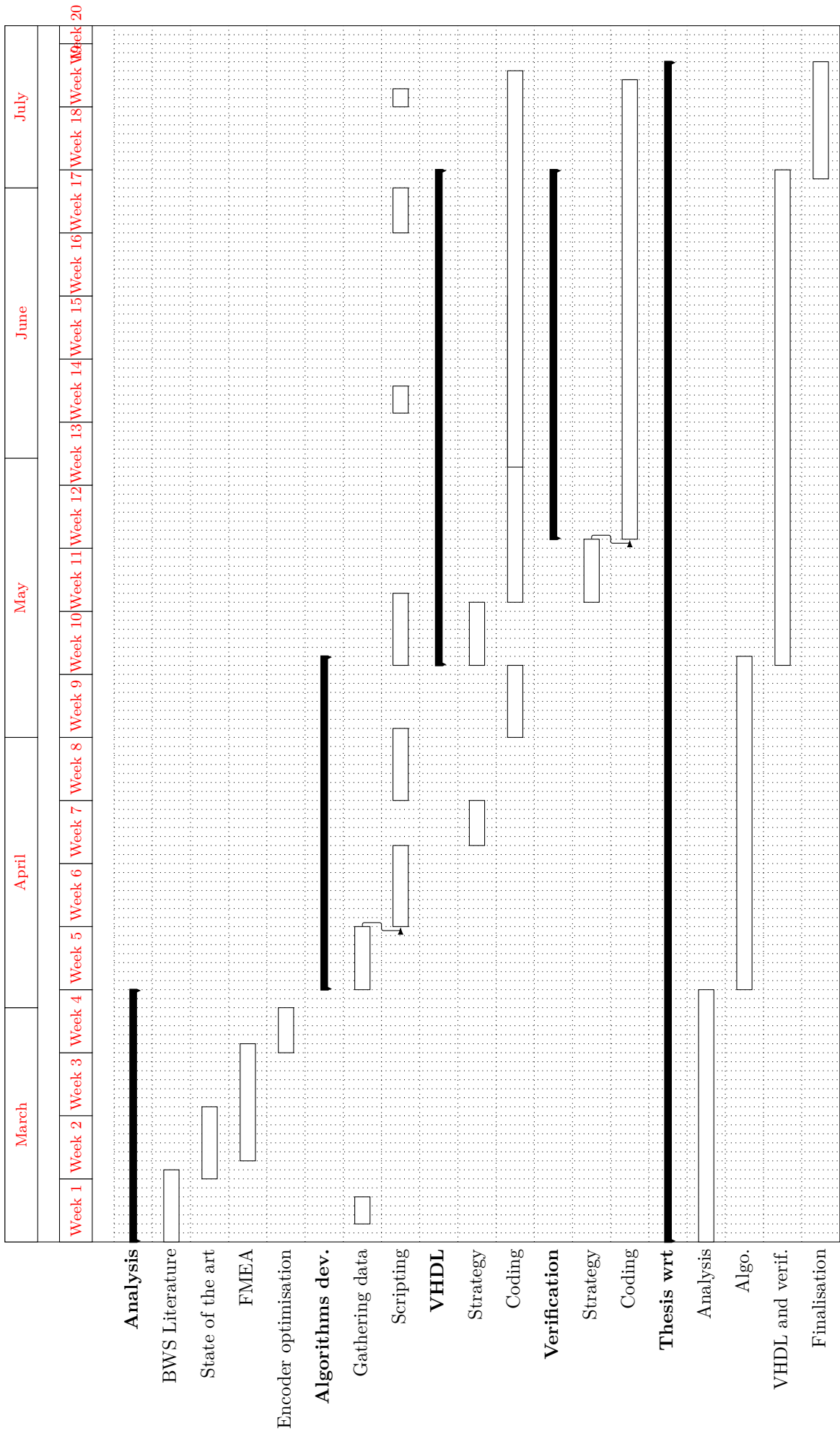
Figure 13.4: Embedded logic analyser (SignalTap) inside the scanner firmware showing the signals of the processing module

Chapter 14

Project management

This chapter essentially provides the project planning information. There is two version of it. The first one was written before the start of the work while the second was made during the work. It essentially shows that multiple tasks were done in a more parallel way than it was expected initially. For example, the algorithms prototyping had to be continued after the planned date, as modifications had to be made after starting the VHDL implementation. In addition, some tasks duration had to be adjusted to fit the time spent on it.





Chapter 15

Conclusion

The Beam Wire Scanner (BWS) new generation heavily relies on sensors to operate and perform particle beam measurements inside the accelerators of the European Organization for Nuclear Research (CERN). The loss of one of these sensors can potentially damage its electro-mechanical part resulting by long system unavailability. In certain cases, this kind of event can even provoke accelerators downtime due to beam pipe obstruction or vacuum leakage.

The way to mitigate these single failure points has been addressed in the functional Failure Mode and Effects Analysis (FMEA) realised in this thesis. The method takes advantage of existing redundancy between the sensors. The resolver type angular sensor can be compared with two Incremental Optical Position Sensor (IOPS) developed in-house and the motor currents measures can be compared to detect failure of one of its hall effect transducer based sensor.

The data processing algorithms designed and developed for a Field-Programmable Gate Array (FPGA) device are able to process sensors measures and detect abnormal behaviour. The system health is checked by monitoring certain signal characteristics and by comparing similar derived quantities such as the position, speed and motor phase current. In addition, the system provides a robust position and speed output computed by the fusion of these informations coming from the multiple sources.

An emphasis was made on the system analysis and the design methodology of the algorithms to ensure the high confidence in the results. The prototyping algorithms have been developed using the software MatLab prior to their implementation in VHSIC Hardware Description Language (VHDL). The simulation vectors are coming from field measurements recorded before this thesis. Extra failure modes have been simulated by creating artificial data starting from existing sets. The output produced by the algorithms written in the two languages have been finally compared to detect differences due to eventual implementation issues or number representation limitation.

In addition to the work on the algorithms, a new type of optical disk for the IOPS was developed to overcome the limitation of this sensor. By embedding the rotation direction and the absolute position directly into the unique mirror track, this incremental angular encoder can be turned into an absolute one and improved the overall system reliability.

Along the way, many challenges have been addressed when developing the necessary tools environment. From existing data measurements analysis, until the comparison of the MatLab and VHDL processing implementation. A plethora of scripts have been created for data manipulation and visualisation and are now ready for continuing further development in this field.

The outcome of this work will allow to deliver to the future users of the BWS a more reliable equipment having higher availability compared to systems of the previous generation.

Bibliography

- [1] ECSS Secretariat. Space product assurance, failure modes, effects (and criticality) analysis (fmea/fmeca).
- [2] Grosjean Colin and Besson Christophe. Motorisation d'un scanner du cern (motor and break design for the wire scanner).
- [3] Jose Luis Sirvent Blasco, Jonathan Emery, and Jose Maria Azorin Poveda. Design of an optical fibre based angular position sensor for wire scanners complying with ultra-high vacuum, high temperature and radiation conditions of the CERN's accelerators, 2012. Presented 2012.
- [4] J. Emery, B. Dehning, C. M. Pereira, J. S. Blasco, S. Cantin, M. Tognolini, B. Schneider, K. Henzer, and M. Starkier. A fast and accurate wire scanner instrument for the cern accelerators to cope with severe environmental constraints and an increased demand for availability. In *2014 IEEE Conference on Control Applications (CCA)*, pages 1139–1145, Oct 2014.
- [5] D. W. Brown, D. L. Edwards, G. Georgoulas, B. B. Zhang, and G. J. Vachtsevanos. Real-time fault detection and accommodation for cots resolver position sensors. In *2008 International Conference on Prognostics and Health Management*, pages 1–6, Oct 2008.
- [6] J. Emery, A. Barjau, B. Dehning, J. H. Alvarez, P. J. Lapray, and M. Macchini. Design and validation methodology of the control system for a particle beam size measurement instrument at the cern laboratory. In *2017 American Control Conference (ACC)*, pages 4221–4228, May 2017.
- [7] MathWorks. Matlab software, 2017. <https://ch.mathworks.com/fr/products/matlab.html>.
- [8] Mentor. Modelsim, 2017. <https://www.mentor.com/products/fv/modelsim/>.
- [9] Intel. Quartus 2, 2017. <https://www.altera.com/>.
- [10] MathWorks. Hdl coder, 2017. <https://fr.mathworks.com/products/hdl-coder.html>.
- [11] RTCA. Do-254, 2017. https://my.rtca.org/NC_Product?id=a1B36000001IcjTEAS.
- [12] RTCA. Do254 user website, 2017. <http://www.do254.com/>.
- [13] REDS. Math2mat, 2017. <http://www.math2mat.ch/>.
- [14] Y. Thoma, E. Messerli, M. Starkier, S. Masle, D. Molla, C. Bardet, G. Trolliet, C. Magliocco, S. Tâche, P. Crausaz, C. Bianchi, O. Gubler, and D. Prêtre. Math2mat User Manual. Technical Report CERN-ACC-2014-0337, May 2011.
- [15] G. Genest, R. Chamberlain, and R. Bruce. Programming an fpga-based super computer using a c-to-vhdl compiler: Dime-c. In *Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2007)*, pages 280–286, Aug 2007.
- [16] REDS. Tlmvm, 2017. <https://reds.heig-vd.ch/>.
- [17] IEEE. floating point, 2008. <http://ieeexplore.ieee.org/document/4610935/>.
- [18] Intel Altera. Support of ieee fixed-pkg and float-pkg within quartus tool, 2016. https://www.altera.com/support/support-resources/knowledge-base/solutions/rd09062016_335.html.
- [19] Andy Rushton. Personal web page where package with the ieee-proposed library by david bishop is still available, 2017. <http://andyrushton.co.uk/vhdl-fixed-and-floating-point-packages/>.
- [20] Github page where the ieee-proposed library by david bishop is still available, 2017. <https://github.com/peteut/vhdl2008c>.
- [21] Admotec. Technologie, 2017. http://www.admotec.com/fr/position_sensor_technology/.
- [22] Matteo Macchini, Sergio Saponara, Luca Fanucci, and Jonathan Emery. Motion control design of a PMSM and FPGA implementation for the Beam Wire Scanner at CERN, Apr 2015. Presented 30 Apr 2015.
- [23] Wikipedia. Savitzky-golay filter, 2017. https://en.wikipedia.org/wiki/Savitzky%E2%80%93Golay_filter.
- [24] Understanding the specifications of your potentiostat, gamry instruments, <https://www.gamry.com/application-notes/instrumentation/understanding-specs-of-potentiostat/>.
- [25] Wilfried Elmenreich. *Sensor Fusion in Time-Triggered Systems*. PhD thesis, 2002.
- [26] Wikipedia. Sensor fusion, 2017. https://en.wikipedia.org/wiki/Sensor_fusion.
- [27] N. Yadaiah, L. Singh, R. S. Bapi, V. S. Rao, B. L. Deekshatulu, and A. Negi. Multisensor data fusion using neural networks. In *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, pages 875–881, 2006.

List of Figures

1.1	Wire-scanner working principle	12
1.2	Illustration of the wire-scanner breakage due to loss of correct position	13
1.3	Sensors combination overview	14
2.1	Wire-scanner control and acquisition architecture	15
2.2	Electromechanical mechanism of the wire-scanner (Courtesy of D. Gudkov, BE-BI-ML)	17
2.3	Motion profiles of the wire-scanner	17
2.4	PSB wire scanner projected position (top) and projected speed (bottom) vs angular position of the scanner shaft	18
2.5	Prototyping electronics used for controlling the wire-scanner system and recoding the data sets processed in this thesis	19
3.1	Block diagram of the actuator with its sensors	24
3.2	Block diagram of the actuator with its sensors	25
3.3	Fault tree of the resolver	31
3.4	Fault tree of the current sensors	31
3.5	Fault trees of the optical position sensor	32
3.6	Fault tree in case of sensors combination	32
3.7	Fault tree in case of sensors combination	33
4.1	Tool chain structure for this thesis	38
4.2	Test bench structure for the verification of individual modules	40
4.3	Field measurement player for the VHDL simulation	41
4.4	Integration of field measurement of the IOPS, resolver and motor currents into the RTL simulation	42
4.5	Co-Simulation setup for the functional verification of feedback systems [6]	42
4.6	Verification of the entity generating the overflow signal	45
5.1	Resolver data from field measurement	48
5.2	Incremental Optical Position Sensor (IOPS) data exemple	49
5.3	Motor phase currents data exemple	49
5.4	Resolver data from field measurement with additional noise apperaring after the start of the movement	50

5.5	Resolver data from field measurement with short jumps artificially added (salt and pepper)	51
5.6	Resolver data from field measurement with long jumps and complete failure artificially added	51
5.7	IOPS sensor failures simulation of CH1 based on field data	52
5.8	IOPS sensor failures simulation of CH2 based on field data	52
6.1	A physical quantity acquisition chain. the blue part represents the scope of this work	53
6.2	Input filtering and monitoring	54
6.3	Sensors processing data flow overview	54
6.4	RDC data processing	55
6.5	IOPS data processing	55
6.6	Motor current data processing	56
6.7	Sensors fusion processing	57
7.1	Solid rotor resolver technological view sourced from [21]	60
7.2	Solid rotor resolver form Admotec used for absolut angular measure of the shaft position	60
7.3	Resolver system overview	61
7.4	Resolver measurement characterisation from the 7 September 2016 at 17h15m05s	62
7.5	Resolver data indicators for the data set (7 September 2016)	63
7.6	RDC processing - plots	64
7.7	The resolver processing componant	65
7.8	RDC processing - P100	65
7.9	Synthesis summary of the resolver processing (P100) on Arria V FPGA	66
7.10	resolver rotation integrator test bench - P140	67
7.11	Intel TimeQuest Timing Analyser used to identify the worst case path	67
7.12	Synthesis summary pipelined version of the resolver processing (P100) on Arria V FPGA	68
7.13	Modelsim simulation of the rdc component - P100 with its input stimulated with the data set S4 artificially parasited with peaks lasting one sample	70
7.14	Comparison of resolver processing implementations, MatLab versus VHDL using the S4 dataset	70
7.15	VHDL simulation results using modified S4 dataset: 5.5 to simulate failure mode 'I01-FM3' (short jumps) c.f.5.3	71
7.16	Comparison of resolver processing implementations using modified S4 dataset: 5.5 to simulate failure mode 'I01-FM3' (short jumps) c.f.5.3	72
7.17	Comparison of resolver processing implementations using modified S4 dataset: 5.5 to simulate failure mode I01-FM3 (short jumps) c.f.5.3	72
8.1	Prototype glass disk of the OPS encoder having multiple tracks of different mirror pitch size	73
8.2	Optical incremental position sensor developped at CERN	74
8.3	IOPS channel one signal recorded on the 7th September 2016 at 17h15m41s	75
8.4	PSB Scanner resolution	75

8.5	SPS Scanner resolution	76
8.6	accuracy-precision-resolution [24]	76
8.7	Exemple of absolut angular position disk	77
8.8	Illustration of a vairable mirror sizes track having constant centroid distance between them (bottom) compared with a constant mirror width track (top)	78
8.9	Physical process of theIOPS with the laser beam at different positions (top) and the resulting signal (bottom)	79
8.10	Laser beam area approximated to create one dimentional vector with 1 μm resolution	79
8.11	Laser beam area approximated to create one dimentional vector with 0.01 μm resolution	79
8.12	Simulation of the Incremental Optical Position Sensor (IOPS) with only one mirror size, the mirrors track (top) is convoluted with the laser beam vector giving a quasi sinusoid (center) and the final sensor simulation with added noise and low frequency modulation (bottom)	80
8.13	Overview of the Simulation process	80
8.14	Simulation of the IOPS with three mirror sizes to determin the rotation direction	81
8.15	Simulation of the incremental optical position system with four mirror sizes	81
8.16	Illustration of a disk track having the position embedded in its mirror width	83
8.17	Simulation of the track with the position encoded with 7, 8 and 11 μm mirrors	83
8.18	Motion profile of the IOPS disk used for calculating the final simulation of the 20MHz digitalisation of the track	84
8.19	Finalised signal with a given position profile (central plot) and amplitude adaptation as the IOPS ADC input (bottom plot)	85
8.20	Analysis of the signal resulting from the absolute disk encoding simulation	85
8.21	Mirror geometry approximation	86
8.22	Disk mask for track photolithography	87
8.23	Zoom on the disk mask for track photolithography	87
8.24	IOPS signals for a displacement πrad on the 7th September 2016 at 17h15m41s	88
8.25	Characterisation of the IOPS channel 1 signal on the 7th January 2016 at 15h43m32s	89
8.26	Characterisation of the IOPS channel 2 signal on the 7th January 2016 at 15h43m32s	89
8.27	IOPS signal processed with a fixed threshold based decision on an optimal data set recorded on the 7th January 2016 at 15h43m32s	91
8.28	Distribution of the mirrors period found using the fixed threshold based decision algorithm on the data set recorded on the 7th January 2016 at 15h43m32s	91
8.29	Exemple on a none optimal data set recorded on the 12th December 2015 at 15m08m46s processed with the fixed threshold based decision algorithm	92
8.30	IOPS signal detection using adaptative threshold on the channel 1 of the data set recorded on the 7th January 2016 at 15h43m32s	93
8.31	IOPS signal detection using adaptative threshold on channel 2 of the data set recorded on the 7th January 2016 at 15h43m32s	94
8.32	IOPS signal detection using minima and maxima patern detection and adaptative windowing	95
8.33	IOPS signal detection using minima and maxima patern detection and adaptative windowing	95
8.34	Mirror detection using the deivative algorithm on the IOPS channel 1 data set recorded on the 7th January 2016 at 15h43m32s	96

8.35	Position and speed recovered by the deivative algorithm on the IOPS channel 1 data set recorded on the 7th January 2016 at 15h43m32s	97
8.36	Mirror detection using the deivative algorithm on the IOPS channel 1 data set recorded on the 2nd December 2016 at 14h50m41s	97
8.37	Position and speed recovered by the deivative algorithm on the IOPS channel 1 data set recorded on the 2nd December 2016 at 14h50m41s	98
8.38	SG derivative method applied to the simulation of absolut track developped in this thesis. Expected nbr of mirrors: 14696	98
8.39	Motion detection using the absolut track developped in this thesis	99
8.40	The IOPS processing componant	100
8.41	IOPS data processing detailed view	100
8.42	IOPS representation error of the angle between each mirror in sfixed of various precision after the dot . .	101
8.43	IOPS P300 entity first version of the synthesis with fixed point division, maximum speed 8MHz	102
8.44	IOPS P300 entity final version of the synthesis with explicit use of the IP core LPM-DIVIDE, maximum speed 79[MHz]	103
8.45	IOPS top processing simulation result using the S4 dataset IOPS CH1: 5.1	104
8.46	IOPS top processing simulation result using the S4 dataset IOPS CH1 5.1, with integer division for the speed calculation	105
8.47	IOPS top processing simulation result using the S4 dataset IOPS CH1 modified for FM3: 5.3, with integer division for the speed calculation	105
8.48	IOPS top processing simulation result using the S4 dataset IOPS CH1 modified for FM4: 5.3, with integer division for the speed calculation	106
8.49	Comparison of IOPS processing implementations using S4 dataset IOPS CH1: 5.1	107
8.50	Comparison of IOPS processing implementations using S4 dataset IOPS CH2: 5.1	107
9.1	Current transducer LA-100-P used in the BWS inverter	109
9.2	Statistics of the currents U, V, W in the stator frame	110
9.3	Third current calculation based on the two others	111
9.4	The current processing componant	112
9.5	Motor current data processing	112
9.6	Currents processing entity synthesis results, maximum speed 25[MHz]	113
9.7	Final currents processing entity synthesis results, with a maximum clocking speed 92 [MHz]	113
9.8	VHDL implementation of the currents processing P400 simulated with Modelsim using the S4 dataset: 5.1	114
9.9	VHDL implementation of the currents processing P400 simulated with Modelsim using the S4 dataset: 5.1	115
9.10	Comparison of the U current processing implementations using S4 dataset: 5.1	116
9.11	Comparison of the V current processing implementations using S4 dataset: 5.1	116
9.12	Comparison of the W current processing implementations using S4 dataset: 5.1	117
9.13	Comparison of the current summing processing implementations using S4 dataset: 5.1	117
10.1	Positions (top plot) and speeds (bottom plot) provided by the reference look-up tables, the resolver and the two IOPS sensors that are used in the sensors fusion	120

10.2 Data driven sensors fusion	120
10.3 Sensors combination processing	121
10.4 Sensors fusion details on values selection	121
10.5 Top entity of the fusion sensor processing	122
10.6 Sensors fusion details on the values sorting process	123
11.1 Overview of the top sensors processing internal components	125
11.2 The top sensors processing componant	126
11.3 Modelsim top sensors processing - position - using S4 dataset: 5.1	127
11.4 Modelsim top sensors processing - speed - using S4 dataset: 5.1	127
11.5 Comparison of the top sensors processing output position, for the two algorithm implementation (Matlab and VHDL)- using S4 dataset: 5.1	128
11.6 Comparison of the top sensors processing output speed, for the two algorithm implementation (Matlab and VHDL)- using S4 dataset: 5.1	128
11.7 Comparison of the top sensors processing output position, for the two algorithm implementation (Matlab and VHDL) - using S4 dataset modified for the failure mode I01-FM4: 5.3	129
11.8 Comparison of the top sensors processing output speed, for the two algorithm implementation (Matlab and VHDL)- using S4 dataset modified for the failure mode I01-FM4: 5.3	129
11.9 Modelsim top sensors processing - position - using S4 dataset modified (see bottom plot of 5.7 and 5.7) to test the failure mode I02-FM4 and I03-FM4: 5.3	130
11.10 Modelsim top sensors processing - speed - using S4 dataset modified (see bottom plot of 5.7 and 5.7) to test the failure mode I02-FM4 and I03-FM4: 5.3	130
12.1 Processing module using the interface from the packages 'data-flow-control-pkg' and 'data-flow-control-interface-pkg'. The input is of type 'sensor-raw-type' and the ouptput 'sensor-processed-type'	134
12.2 Convolution module for FIR filter type processing	135
12.3 Topology of Direct form Finite Impulse Response (FIR) filter, used for discrete convolution of an input stream	136
12.4 Convolution verification	136
12.5 Median filtering scheme with 7 layer	138
12.6 Input stream filtering and monitoring using an median filter	138
12.7 Median filter block verification results	139
12.8 Median filter block verification results	139
12.9 Monitoring module	140
13.1 Integration of the sensors processing into the operational firmware of the PSB scanner	143
13.2 Integration of the sensors processing into the operational prototype for the PSB, after optimisation to increase the maximum clocking frequency	144
13.3 BWS prototype setup where the processing module was tested	144
13.4 Embedded logic analyser (SignalTap) inside the scanner firmware showing the signals of the processing module	145

Written in Latex using TeXstudio 2.12.4 and MiKTeX 2.9.

Compilation date: September 30, 2017

