Design & Implementation of a Control Unit for a Tongue Control System

MASTER PROJECT'S THESIS, AAU

APPLIED SIGNAL PROCESSING AND IMPLEMENTATION (ASPI)

Group 1043
Umberto CERASANI
Jeremy LERESTEUX

AALBORG UNIVERSITY
INSTITUTE FOR ELECTRONIC SYSTEMS

Fredrik Bajers Vej 7 ▪ DK-9220 Aalborg East Phone 96 35 80 80 http://www.esn.aau.dk

TITLE:
Design & Implementation of a Control Unit for a Tongue Control System

THEME:
Medical application
Signal processing
Implementation

PROJECT PERIOD:
10^{th} Semester
February 2009 to June 2009

PROJECT GROUP:
ASPI 09gr1043

PARTICIPANTS:
Umberto Cerasani
cerasani@es.aau.dk
Jeremy Leresteux
jlereste@es.aau.dk
SUPERVISORS:
Yannick Le Moullec (ASPI)
Morten Enemark Lund (HST)

PUBLICATIONS: 5

NUMBER OF PAGES: 95

APPENDICES: 1 CD-ROM

FINISHED: 3rd of June 2009

Abstract

This 10^{th} semester project for the « Applied Signal Processing and Implementation » specialization at Aalborg University is a study on the improvement and implementation of the algorithms of a Control Unit for a Tongue-based Control System. The project focuses on a medical application permitting motion-disabled people to control their environment, i.e. external devices like computers, wheelchair, mobile phones,... The Tongue Control System is developed by the HST Department at Aalborg University where prototypes have been designed. The Mouthpiece Unit (MU), which receives the commands from the user by means of his tongue, is fully developed. This unit sends the different instructions to a Central Unit (CU) for the control of the external devices.
This project seeks, firstly, to improve the algorithms for processing the signals provided by the MU. Indeed, because of its characteristics, noise and drifts in temperature are added to the useful signal. After an analysis of both signal perturbations, the successful custom Kalman filter removes these pertubations without any delay or data loss. The second part of the project deals with the analysis of different platforms to choose the appropriate one for the implementation of the communication and interfaces algorithms. The project group chooses the FPGA Altera DE2 because its flexibility ensures easy additions of future interfaces for many external devices. Moreover, the Kalman filter is implemented on this new platform. The third part explains how to use the tools provided with the Altera DE2 and the results obtained on the implementation. The algorithm performing the logic to control a computer mouse cursor is successfully implemented but the core of the project is the work of this mouse cursor control with the Kalman filter. The process of the signal from 8 sensors, has been successfully implemented and gives good results. Combined with the mouse cursor control, it is possible to control the mouse cursor from the MU and produce smooth and accurate moves. Finally, the conclusion and discussion about the performances are developed in the last part, as well as the future perspectives discussions.

Preface

This report is the documentation for a 10^{th} semester project in Applied Signal Processing and Implementation (ASPI) entitled « Design & Implementation of a Control Unit for a Tongue Control System » at Aalborg University (AAU). This report is prepared by group 09gr1043 and spans from February 9^{th}, 2009 to June 3^{rd}, 2009. The project is supervised by Yannick Le Moullec, Assistant Professor in ASPI at AAU, and Morten Enemark Lund from the Health Science & Technologies Department at AAU. The report is divided into seven parts. These chapters correspond to the introduction of the project, analysis of the prototype problems and signal processing, analysis of the prototype algorithms, analysis of different platforms, implementation, tests and conclusion.

The bibliography is present in the last pages of this report with references to the bibliography in square brackets as in [1]. The accompanying CD contains a copy of this report, all code used in the project and two movies that illustrate the results obtained.

Umberto Cerasani

Jeremy Leresteux

Aalborg, June 3rd 2009

Acknowledgements

The project group would like to thank everyone because of whom this project has been successful. First and foremost we would like to thank our parents who have been by our side and been a continuous source of inspiration. Next we would like to thank our supervisors, Yannick Le Moullec and Morten Enemark Lund for the precious guiding and for supporting us throughout the difficulties encountered.
We would like to extend our gratitude toward Kai Borre and Darius Plausinaitis for their efficient teaching about Kalman filters and finally to Morris Filippi and Jean Michel Lory for their constant help and technical support during all the project.

Contents

Chapter 1

Introduction

The International Classification of Functioning (ICF) [1] defines disability as « the outcome of the interaction between a person with an impairment and the environmental and attitudinal barriers he/she may face » [2].

Even with major disabilities, such as quadriplegia [3], people can still move some part of their body such as eyes, jaws, head, tongue, finger. Adapted control systems have been developed to permit disabled people to control their own environment. With these equipments, they are able to control, for example, their own wheelchair without the help of any third party. They can also use a computer and move the mouse cursor, write emails,. . . Despite the improvement on autonomy, using one part of the body continuously is exhausting and brings pain, headaches for eye controlled systems, neck pain for head controlled systems, etc. Moreover, these systems are visible by all and increase the feeling of difference. Brain control systems are also in development but they still need significant research efforts and improvement to be efficient and aesthetic.

1.1 Introduction to the Tongue-Based Control System

The mouth is the first stage of the digestive system when saliva and jaws chewing transform foodstuffs to be swallowed with the help of the tongue. The tongue is a muscle attached to the floor of the mouth. It permits to detect and to define tastes and temperatures of foodstuffs by means of the papillae and taste buds which are covering the tongue up [4]. The tongue is able to touch every single teeth of our mouth. It is imaginable to reach with the tip of the tongue one sensor for each of the alphabet's letters to simulate a keyboard.

A system based on tongue control has been declared preferred by disabled person in a comparative study [5] even if its efficiency, in terms of response time, is less accurate than a head control system. This preference is due to the fact that these systems are hidden in the mouth and decrease the feeling of difference for disabled people. Different tongue-based control systems have been developed, like described in [6], [7], [8]. These control systems cannot be used while the users eat or drink. This fact accentuates the feeling of being different among the users.

As mentioned above, the tongue environment, i.e. the mouth, is humid due to the saliva. The mouth is also restrained to temperature variations credited by foodstuffs and drinks. A solution which is not sensitive to humidity and temperature must be found to permit the usage of a tongue-based control system while the user eats or drinks hot or cold products.

A new system based on Faraday's law of induction for a coil is introduced in « A Tongue Based

Control for Disabled People » by Lotte N.S. Andreasen Struijk [9]. The advantages of a magnetic solution are its insensitivity to humidity and its small dependency to temperature variations. This system is using multiple tiny coils as sensors. It is then possible to install many of them on a palatal plate to simulate a keyboard or a joystick which can be used to control a computer mouse cursor or a wheelchair.

Based on Faraday's law of induction for a coil [10], the voltage drop ϵ, known as the ElectroMotive Force (EMF) of an ideal inductance, i.e without the resistor [11], is defined as:

$$\epsilon = -L * \frac{di}{dt} \tag{1.1}$$

where :

$$L = \mu_0 * \mu_r * N^2 * \frac{A}{l} \quad \text{[Henrys]} \tag{1.2}$$

i : sine wave current [Ampere]

with :

μ_0 : vacuum permeability

μ_r : relative magnetic permeability of the core material

N : number of turns of the coil of wire

A : the area perpendicular to the magnetic field

l : average length of the magnetic flux path

The value of the inductance L, in equation 1.2, depends on the number of turns N of the coil of wire but above all depends on the relative magnetic permeability of the core material μ_r.
When a ferromagnetic element approaches the coil, it creates an inferred current in the coil which can be added or subtracted to the generator current (depending on the direction of the generated electromagnetic field). This change, according to the Lentz-Faraday formula, equation 1.1, can be interpreted as a voltage variation which can be detected by a voltmeter to define an activation of the sensor. It remains activated until the ferromagnetic material is removed away from the coil.

1.1.1 The Tongue Control System

A research group from the Department of Health Science & Technology (HST) at Aalborg University [9], by using this theory of induction, has developed a tongue-based control system, the Tongue Control System (TCS), which can be used while eating or drinking products. As indicated in Section 1.1, a voltage drop can be detected when a ferromagnetic material approaches a coil of wire, referred as sensor. A ferromagnetic material on the tip of the tongue could, then, produce activations of sensors placed on a palatal plate in the mouth of users. The ensemble composed of the palatal plate and the 24 sensors is called the Mouthpiece Unit (MU). The ferromagnetic material, referred to as the Activation Unit (AU), is placed at the tip of the tongue. Aiming at the sensors implemented in the palatal plate with the AU activates them. These activations of the sensors are detected and transmitted to the Central Unit (CU), an external device, which connects the MU and the environment the users wish to control (computer, wheelchair, lights, doors, . . .). The CU is designed to receive the instructions from the MU, calibrate them , process them and send them to the external devices (mouse cursor, wheelchair, . . .) corresponding to the instructions.

Figure 1.1 shows each sub-systems of the TCS. Paragraphs 1.1.1.1, 1.1.1.2 and 1.1.1.3, based on the Requirement Specification from HST Department [12], define their goals, principles and constraints.

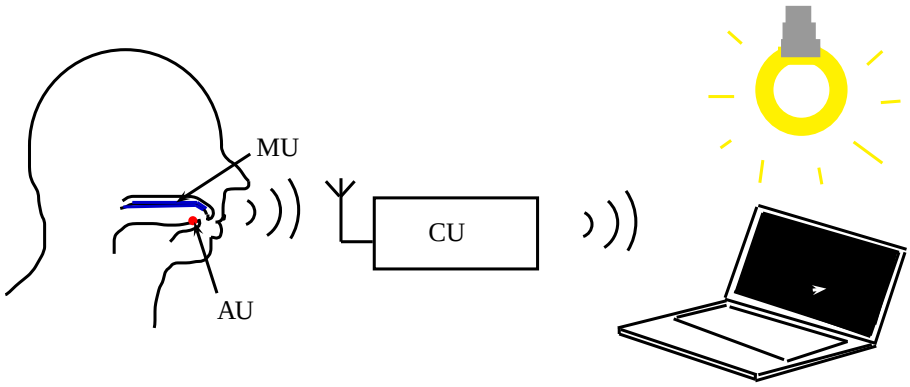


Figure 1.1: *The Tongue Control System (TCS) contains the palatal plate and sensors (MU), the ferromagnetic material piercing on the tongue (AU) and the control unit (CU) that contains software and hardware for the communications between the MU and the external devices.*

1.1.1.1 The Activation Unit

The Activation Unit (AU) is a metal tongue piercing made from a ferromagnetic material which can be detected by the sensors in the MU. The AU is intended to be surgically inserted through the tongue approximately 1cm from the tongue tip.

1.1.1.2 The Mouthpiece Unit

The Mouthpiece Unit (MU) is the palatal plate where the sensors are implemented in. The MU contains also the electrical circuit for detecting the state of the sensors. Figure 1.2 illustrates, for one sensor, the signal path from the scanning step to the transmission step to the CU.

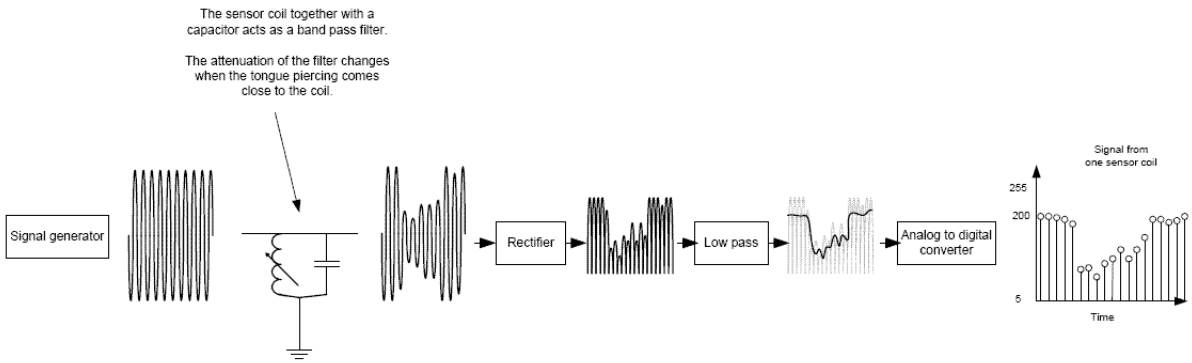


Figure 1.2: *Signal creation path from the activation to the transmission to the CU for one sensor. The signal generator produces the voltage source signal based on the sine wave current. This voltage signal traverses the sensor. If the AU approaches the sensor, the voltage signal is distorted. A rectifier converts the resulting signal to a positive signal followed by a low-pass filter to detect the baseline. The baseline is, then, converted to digital by means of the Analog to Digital Converter (ADC).*

The MU, then, has to send to the CU the position of the Activation Unit from the tongue on the

different sensors implemented in the palatal plate. The retained solution is to use a wireless digital radio which transmits a packet of data representing the state of each sensors. The first prototype, developed by the HST members, contains 24 sensors hence the MU sends a 24 byte sensor data packet. Disregarding the bytes used for the transmission protocols and considering that the transmission is error free, the radio packets received by the CU should have the format shown in Figure 1.3

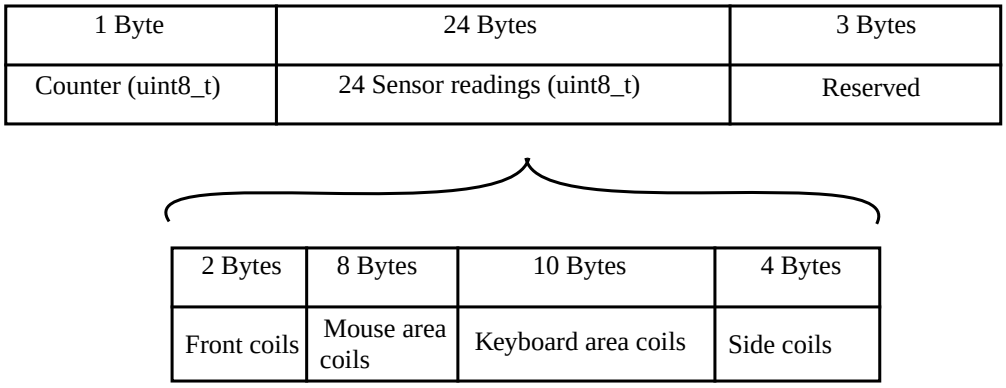


Figure 1.3: *28 Bytes radio packet received from the Mouthpiece Unit by the CU. The 24 Bytes correspond to the 24 coils implemented in the palatal plate. Modified from [12]*

The counter goes from 0 to 255 (1 byte) and then restarts.The sensors values are between 0 and 255 and represent the values of the estimate tension of the coils. All the activations are negatives.

The state of each coil is transmitted to the CU every 30 milliseconds, i.e. the system is working at 33Hz.

The chosen radio transmitter is the nRF24L01 by Nordic Semiconductor [13]. It is a single chip 2.4GHz Transceiver. Its characteristics are as follow:

- Worldwide 2.4GHz ISM band operation. Transmission on this frequency band is Licence free;
- 126 RF channels;
- Common RX and TX interface;
- GFSK modulation;
- Ultra low power operation;
- 11.3mA TX at 0dBm output power;
- 13.5mA RX at 2Mbps air data rate;
- 900nA in power down;
- 26μA in standby;
- Maxi supply voltage 3.6V;
- 5V input tolerant;
- Compact 20-pin 4x4mm QFN package.

1.1.1.3 The Central Unit

The Central Unit (CU) is an external box, close or far from the user, which contains the hardware and software to link the MU and the external devices the user wishes to control. It contains :

- the radio receiver, which receives the data packets, i.e. the state of each sensors, from the MU,
- the inference system, which combines signals from several sensors to create a direction vector, e.g. a joystick,
- the Controller Logic (CL), which monitors and calibrates the signals from the MU. The CL is the element of the CU which, according to the users input, switches to the control of one or another external device,
- the external device interface, which gives an interface (wired or wireless) to each of the external devices.

The CU is still in a research phase but the first prototype designed by the HST Department, see Figure 1.4, permits to a user to move the computer mouse cursor. However, some noise does not allow the move to be very accurate: where even in non active state, i.e. no move is asked, the mouse cursor moves by itself. This problem has to be solved by the Controller Logic. This first prototype, implemented on a 8-bit microcontroller, does not permit to achieve a complete and powerful CU, as no other external devices has been interfaced on it. Those limitations are addressed in this project, as stated in the project goals, Section 1.2.

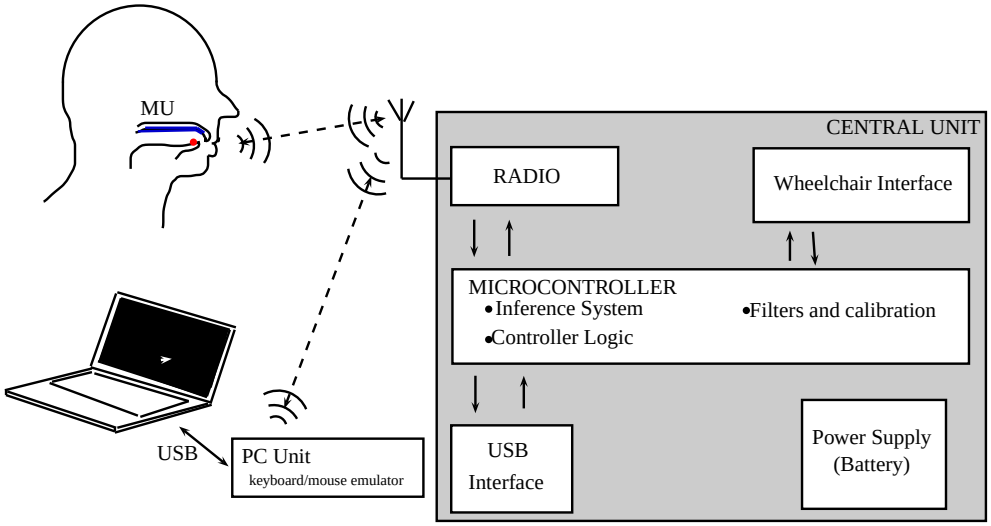


Figure 1.4: *The existing CU prototype receives the data from the MU by means of the radio receiver. The microcontroller processes, filters and calibrates these data to define which external interface needs to be controlled, by means of the Controller Logic. Finally the inference system creates the direction vector. The result is send to the PC Unit, which emulate the computer mouse cursor.*

1.1.2 Conclusion about the context of the project

For disabled people affected by mobility disabilities, the control of their environment is a constant fight. Simple moves as opening a door, moving their wheelchair or switching the light are impossible without

the help of a third person. Access and use of common modern tools, e.g. computer, mobile phone also requires help. New technologies permit to take advantage of every movement disabled people can do. Control systems are now available, based on eyes, jaws or head movements detection. All these systems, as competitive as they can be, do not remove the feeling of difference perceived by disabled people. Indeed, these systems add complex architectures visible by all and increase the feeling of difference.
Some control systems based on tongue movement have been declared preferred by disabled people because they are hidden in the mouth even if their accuracy is less efficient than other control systems. But this kind of systems do not permit a usage during eating or drinking.
Lotte N.S. Andreasen Struijk and the Department of Health Science & Technology (HST) members at Aalborg University (AAU), by using Faraday´s law of induction for a coil, have developed a tongue control system able to be used during eating and drinking. A ferromagnetic material pierced on the tip of the tongue, the Activation Unit (AU), changes the properties of the coils, implemented in a palatal plate, the Mouthpiece Unit (MU), by approaching or moving away from them. This change of properties of the coil is referred to as an activation of a sensor and is sent by wireless to a Central Unit (CU) which gather all the activations, process them and sent the result to the right external device to be control, e.g. a computer mouse cursor. The first prototype is able to control only the mouse cursor of a computer and some perturbations do not make the system accurate enough to be used by disabled people.

1.2 Project goals

As discussed in paragraph 1.1.1.3, the CU prototype realized by HST Department must be improved to give an accurate and complete system which can be used with multiple external devices. An implementation of the Central Unit on a flexible, extendible and powerful platform must correct the perturbations and permit the addition of new future external devices.

To complete the Tongue Control System to a marketable, complete and accurate product, the features that the new CU platform must provide, and which constitute the sub-goals of this project are:

1. processing the raw signal from the MU to produce a better signal quality to enhance the accuracy of the system,

2. executing the algorithms of communication between the other sub-systems of the TCS and the external devices,

3. being as flexible and extendible as possible to permit the addition of new external devices interfaces in the future.

The CU is also a monitor to the users on their actions. It must give a feedback (audio, graphical) about the state of the system and which external device is under-control.

1.3 Design Methodology

The purpose of the design methodology is to supply a structured approach to the analysis of the specifications, implementation of the system, and evaluation of the results obtained in the project.

The A^3 design methodology [14] introduces three domains, Application, Algorithm and Architecture, described next page. Figure 1.5 shows the « generic » A^3 design methodology. Each chapter of this thesis corresponds to one of the domain of this design methodology.

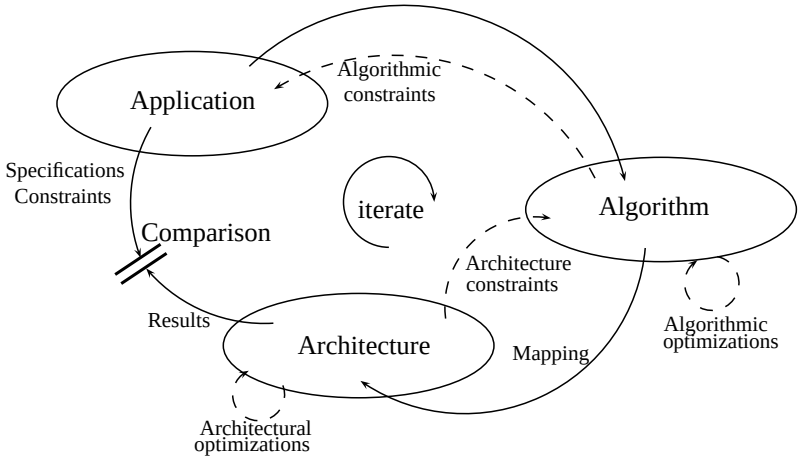


Figure 1.5: *The « generic » A^3 design methodology.*

- Application : Any system with specification requirements and constraints i.e. time constraints, power consumption, area problems, cost, specific domain requirements,... It is the final purpose of a project.

- Algorithm : Any software tool, standardized or created, defining and answering to the application specifications. The improvements and/or changes are only proceeded on an algorithmic point of view.

- Architecture : Any platform (DSP, FPGA,...) used to implement and execute the algorithms to fulfil the application specifications. The result is compared to the specifications/constraints of the application. In case of differences or appearance of new issues, modifications have to be done. The architecture domain allows two different kind of modifications. On one hand, if the algorithms are implemented on a fixed architecture, a modification of the algorithm, in terms of architecture related program (bus control, data transfer control, memory allocation,...) can be done for the specified architecture. On the other hand, if the algorithms are established then a modification of the architecture (VHDL program for a FPGA platform for example) can be done.

For this project, all three domains can be identified as follow, illustrated in Figure 1.6:

- Application : The Central Unit of the Tongue Control System. It has to fulfil the constraints established in Section 1.2.

- Algorithm : The algorithms, done for the first prototype by the HST members, to control the computer mouse cursor. Plus new algorithms developed by the project group to process the raw signals and control other external devices.

- Architecture : In this project, the choice of the architecture/platform is primary. An analysis of different architectures, such as FPGA or DSP, is vital to fulfil the objectives.

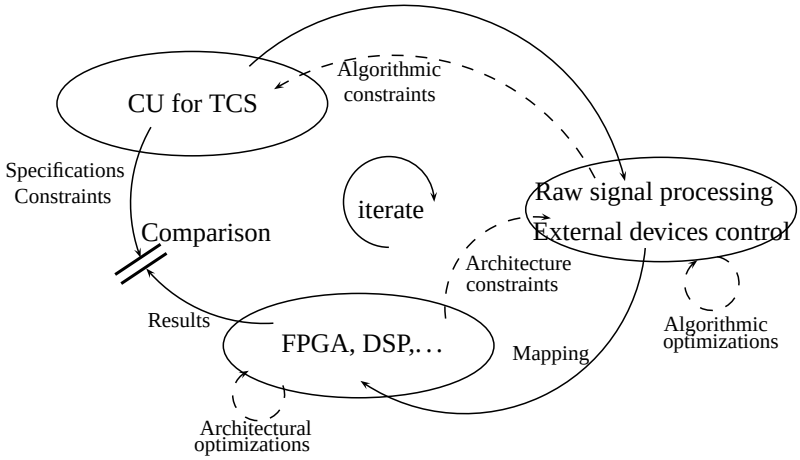


Figure 1.6: *The A^3 design methodology for the CU for TCS project.*

1.4 Time Plan

The following Gant diagram illustrates the time plan followed by the group in order to fulfil all the project goals.
This time schedule was made at the beginning of the project and was almost respected. However in the preliminary Matlab experiences and in the implementation part, the group meets several issues which shift all the time schedule. In consequence, some other parts as the choice of the platform were restricted.

Num	Task	Start	End	Duration	Q1 - 2009			Q2 - 2009		
					January	February	March	April	May	June
1	Preliminary Matlab Experiences (Remove Noises and correct the effect of temperature variation)	1/2/2009	1/3/2009	20						
2	Analysis of the constraints and choice of the platform	1/3/2009	21/3/2009	15						
3	Analysis of the different algorithms - Fuzzy code - raw signal processing - control of the wheelchair	21/3/2009	23/5/2009	45						
4	Implementation on the selected platform - Fuzzy code - raw signal processing - control of the wheelchair	21/3/2009	23/5/2009	45						
5	Optimization of the algorithms	23/5/2009	3/6/2009	7						
	Report Writing	18/2/2009	3/6/2009	75						

Figure 1.7: *The CU for TCS project Gant diagram.*

1.5 Project Delimitations

As students of Applied Signal Processing and Implementation (ASPI), no medical´s norms, totally or partially, are mentioned and/or used in this project thesis.
The study of the MU has already reached its completion. An introduction of its main principle is presented as well as the data used for this project from the MU. No deep explanations are given in this work on the MU.
An evolute study of the digital radio transceiver/receiver permitting the communication between the MU and the CU is not done in this thesis. Only an introduction of it and the receiving algorithm is purchased.

Chapter 2

Analysis and signal processing

This chapter is relative to the Algorithmic domain from the A^3 design methodology, especially to the raw signal processing, highlightened in Figure 2.1. The chapter introduces the first experiments made by the group to get acquainted with the Tongue Control System (TCS). These experiences are done using MATLAB. They are focused on the reception and processing of the data transmitted by the MU to the CU. Moreover, specific signal processing has been applied to the signals to answer the first sub-goal defined Section 1.2:

1. Processing the raw signal from the MU to produce a better signal quality to enhance the accuracy of the system.

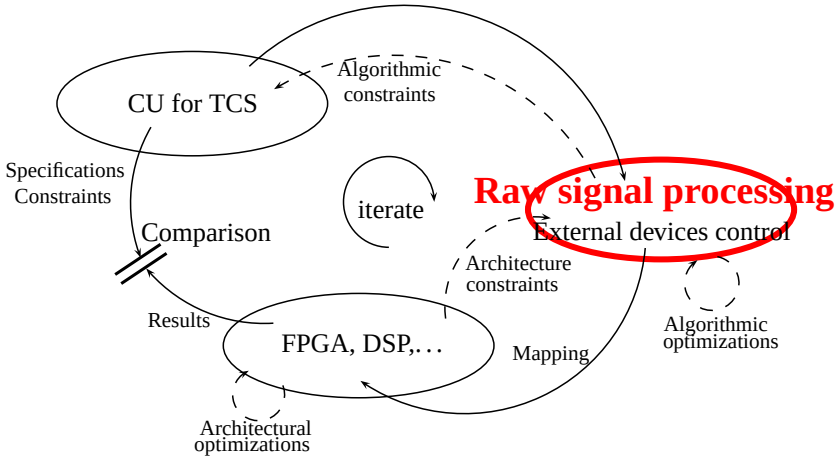


Figure 2.1: *The A^3 design methodology applied to Chapter 2*

2.1 Environment

For these tests, the MU sends a 28 bits data packet to a radio USB stick every 33 milliseconds. This data packet corresponds to the state of each sensors implemented in the MU. This USB stick can easily be interfaced to MATLAB to process the data. Figure 2.2 illustrates the environment.

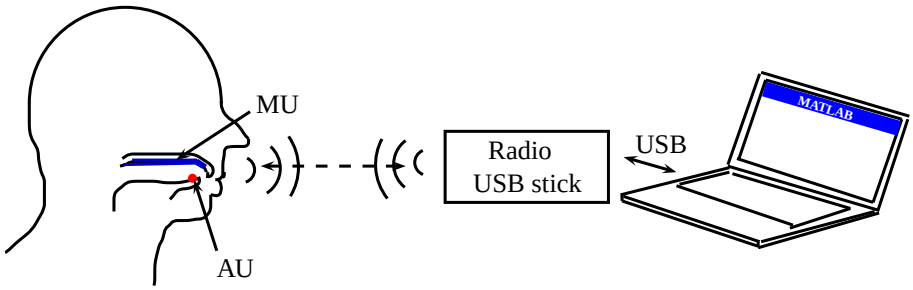


Figure 2.2: *Environment of the MATLAB preliminary experiments. The MU sends the 28 bits data packet to the USB stick every 30 milliseconds. The USB stick collects this data to be usable on MATLAB.*

As introduced Section 1.1, the MU sends the state of each sensors to the radio USB stick. The state of the sensors is defined by the equation 1.1 which gives a voltage difference ϵ between two scans of the MU on the sensors.
The raw signal given by the MU to MATLAB, through the USB stick, from one sensor, has the shape seen in Figure 2.3. It has an offset of 200mV and is significantly impaired by high frequency noise.

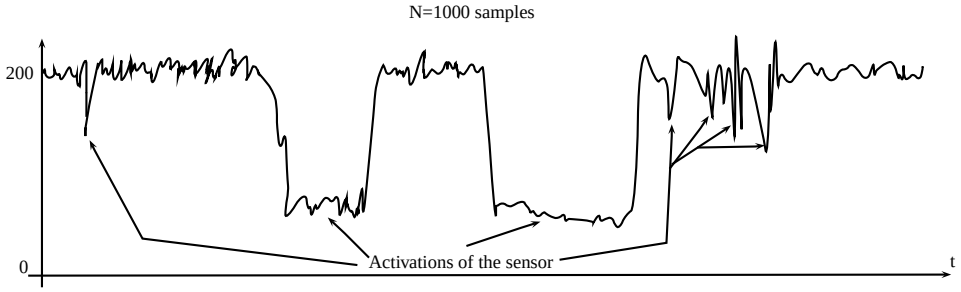


Figure 2.3: *The Raw Signal, from one sensor, given by the MU with different length of activations*

2.2 Problems and Goals

Different problems are observed during the first tests. From these observations, the group members have defined 3 sub goals.

1. The first sub goal is the removal of the 200mV offset.
2. The second sub goal is to remove the noise as much as possible, because it causes activation effect when no activation is asked by the user.
   Noise is high frequencies added to the raw signal by different sources such as transmission channels, reflection of the signal on obstacles, other electrical devices in the environment which are also transmitting signals. As the radio transmitter used in the MU, and the given project scenario, is digital, it is assumed that no noise comes from transmissions. In the case of this project, the high frequency noise comes only from the scanning of the sensors themselves since the scanning is based on the variation of inductance value of the coils. These variations of inductance depends on the temperature applied to them and also on the Electro Magnetic Compatibility (EMC) interference in the sensors.

3. Finally, the previous research done by the Health Science and Technology (HST) members has shown that when a variation of temperature, e.g. the user is having a hot drink, appears on the MU, a variation of the signal delivered by each sensors is also observed. This variation causes an offset which could, once again, be interpreted by the system as an activation of the sensors where no activation is asked. The last sub goal seeks to remove this drift cause by temperature.

Section 2.3 introduces a complete analysis of the perturbations met by the TCS.

2.3 Analysis of the signal perturbations

The analysis of the input signal is of primary importance to distinguish the useful signals generated by the user from the noise perturbations that should be removed. As indicated in Section 2.2, the 2 main problems are the noise perturbation and the baseline wandering. The analysis of the spectrum permits to distinguish and delimit in frequency the perturbations due to the two different phenomena.
Noise is present all the time, and probably caused by thermal noise in the system or EMC interference in the sensors. The characteristics of this noise perturbation are totally different from the characteristics of the temperature shift. The Power Density Spectrum (PDS), presented Paragraph 2.3.1.1, shows that for high frequencies, there is almost only noise because the high frequencies peaks during activations are very short. Indeed, the user cannot perform very quick activation of the sensors (more than 10 activations per second is impossible so one can assume that for a frequency up to 10Hz, noise is highly overriding).
Furthermore, the baseline shift is caused by the variation of the temperature in the mouth (it can be caused by the breathing or during drinking a warm or cold liquid) because the resistance of the sensors changes. Obviously not only one sensor is affected by the temperature changes. For instance during drinking the localisation of the liquid (water, coffee, etc.) is unpredictable and the variation of temperature caused by the temperature variations modifies the baseline value of different sensors not in the same time and not in the same way.

2.3.1 Noise analysis

The group assumes that the noise present is the thermal noise also called white noise. It is a random noise generated by the thermal activation of the electrons in an electronic device. This kind of noise is unpredictable but can be studied through its statistical proprieties. A white noise $B(t)$ with a variance σ is an unpredictable stationary process, centred ($mB = 0$) where the statistical autocorrelation function is a delta function weighted by σ^2. According to digital signal process theorems [15], the PSD function is the Fourrier transform of the autocorrelation function. Therefore the PSD function of a white noise is a constant function with the same weight of the autocorrelation function as shown in equations 2.1 and 2.2:

$$R(t) = \sigma^2 \times \delta(t) \tag{2.1}$$
$$\chi(f) \equiv FT[R(t)] = \sigma^2 \tag{2.2}$$

where R is the autocorrelation function of the white noise, σ is the variance of this noise and χ is the PSD. The same relations are available for a white discrete noise, as seen in equations 2.3 and 2.4:

$$R(k) = \sigma^2 \times \delta(k) \tag{2.3}$$
$$\chi(f) \equiv FT[R(t)] = \sigma^2 \tag{2.4}$$

In the latter case, the Discrete Fourrier Transform (DFT) is used to pass from the sample domain to the time domain.

2.3.1.1 Power Density Spectrum Estimators

Consider a random and stationary process $X(n)$ with N samples. Every sample is a random variable with a probability density of $p[X(n)]$. If α is a parameter of this random process (like its mean, its variance, its PSD,...), $\tilde{\alpha} = F[X(0), X(1), X(2), \ldots, X(n-1)]$ is a random estimator variable created from the N samples. F is called estimator, the $bias$ (eq. 2.5) and the $variance$ (eq. 2.6) of the estimator can be evaluated and in order to have a consistent estimation of the studied parameter the values of the $bias$ and the $variance$ has to tend towards 0 when N tends towards the infinite which means that the estimation error is negligible when the number of sample is sufficient.

$$bias = E[\tilde{\alpha}] - \alpha \tag{2.5}$$

$$variance = E\left[\left(\tilde{\alpha} - E\left[\tilde{\alpha}\right]\right)^2\right] \tag{2.6}$$

From [15]

There are 5 main estimators [15]:

- simple correlogram,
- smoothed correlogram, which is the previous estimator corrected with an apodisation window (Hanning, Blackman,...),
- the simple periodogram which is the FT of the estimation of the autocorrelation function,
- the mean-sized periodogram which consists of the division of the N samples in L section of size M where a simple periodogram is applied and then the mean of all the simple periodograms is performed in order to decrease the variance of the previous estimator,
- the Welch Spectral Estimator, the most complex one that will be explained on the following paragraph.

This study focuses on the two utilised periodograms available in Matlab: mean-sized periodogram and Welch Spectral Estimator.

1. The mean-sized periodogram
   The division of the N samples into L sections of size M is performed and then on each section, a simple periodogram is performed as shown in equation 2.7.

$$\hat{\Gamma}3(f) = \frac{1}{L}\sum_{l=1}^{L}\hat{\Gamma}2l(f) = \frac{1}{LM}\sum_{l=1}^{L}\left[\sum_{k=0}^{M-1}\chi_l(k)\exp -2\pi j \cdot f \cdot k\right]^2 \tag{2.7}$$

From [15]

   where f is the sampling frequency and χ is the value of the signal. $\hat{\Gamma}3$ is an estimator with $bias$ of the PSD: smaller is M, larger is the $bias$. The $variance$ is proportional to the PSD but decreases when L increases. A trade-off has to be found between the $bias$ and the $variance$, which is the main drawback of this estimator.

2. The Welch Spectrum Estimator
   The WSE is a mean-sized periodogram and a smoothed periodogram as shown in equation 2.7.

$$\hat{\Gamma}5(f) = \frac{1}{L}\sum_{l=1}^{L}\hat{\Gamma}5l(f) = \frac{1}{LMC}\sum_{l=1}^{L}\left[\sum_{k=0}^{M-1}\chi_l(k)w(k)\exp -2\pi j\cdot f\cdot k\right]^2 \tag{2.8}$$

From [15]

The division of the N samples into M new samples is performed and the multiplication of these samples with a correction window is carried out. C is a factor of normalisation needed because of the multiplication of the samples with the corrected window.

$$C = \frac{1}{M}\sum_{k=0}^{M-1}w^2(k) \tag{2.9}$$

From [15]

The overlap of samples can be done to improve the resolution, it permits to manipulate more samples.

The function *psd* or *spectrum* in Matlab uses by default $M = 256$, $overlap = \frac{1}{2}$ and the Hanning correction function.

A detailed report about the Power Density estimators can be found in the literature [15].

2.3.1.2 Analysis of the received signal

Figures 2.4 and 2.5 are the results of experiments on the same sensor (sensor number 5 in the MU). The signals received and then processed with a PSD estimator permit in the time domain to distinguish the different problems. The first signal (in blue), illustrated in Figure 2.4, is a signal which could be created by the user with long and small activations. Obviously, noise is present. The second one (in red), shows only the noise without the signal. In order to obtain the variance of the noise (useful for filtering) the PSD on the two different signals is performed, and finally the third one (in pink) shows the noise added with a baseline shift.

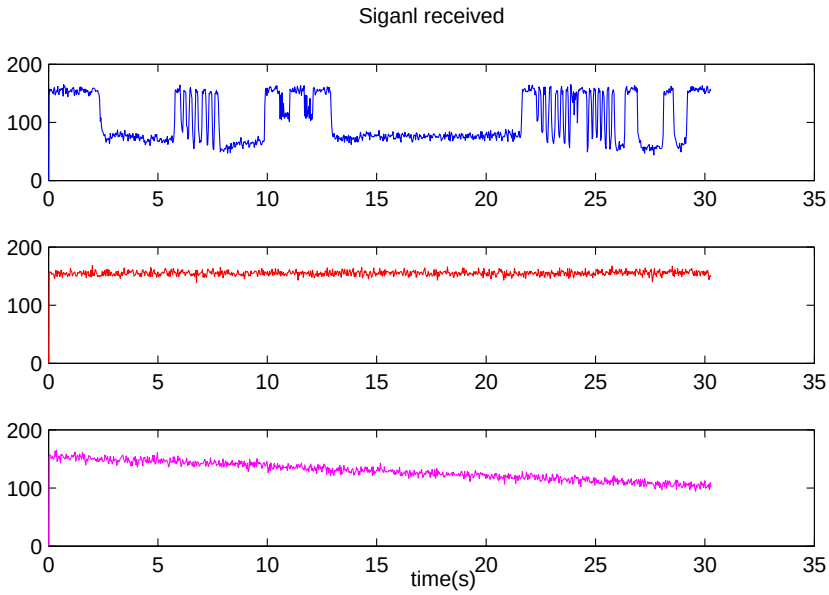


Figure 2.4: *The first signal (blue) could be created by the user with long and small activations. The second one (in red) shows the noise without signal. The third one (in pink) shows the noise added with a baseline shift. The value of each sensor is in mV.*

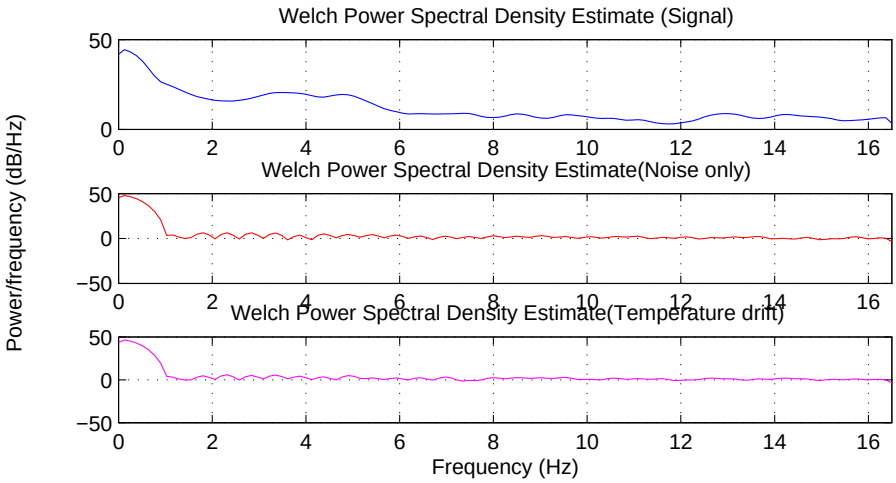


Figure 2.5: *Welch Power Spectral Density of each signals from Figure 2.4.*

In Figure 2.5, the blue signal shows that the PSD becomes negligible above 6Hz and then the continuous value is predominant. Indeed, the higher frequency signals appear only during activations or desactivations but they are really short in time domain. The small variation of the PSD´s values close to 0Hz are due to the baseline wandering. Red and pink curves confirm these commentaries.

2.3.1.3 Tested Solutions to reduce the high frequency noise

To reduce high frequencies noise, an obvious solution is the use of a low-pass filter, which permits to remove high frequencies from a given signal while keeping low frequencies. The cut-off frequency of this kind of filter must be set in advance. Many different low pass filters exists. The two mains kinds of digital filter are the Finite Impulse Response (FIR) and the Infinite Response Filter (IIR). The most common and simplest FIR low pass filter is the Moving Average Filter [16] while the IIR filters are led by Butterworth, [17], Chebychev (Type 1 and 2), [18], and Bessel Filters, [19].

FIR Filter: The Moving Average Filter As mentioned above, the Moving Average Filter is the most common and simplest filter in digital signal processing. It is used to "reduce random noise while retaining a sharp step response" [16]. This filter gives a good smoothing for time domain. However, it is not the best choice for the frequency domain usage. The moving average filter is a FIR filter. It is a digital filter which is used in many applications, such as imaging, video, sounds processing, transmitted signals, where filtering unwanted noises is needed. The moving average filter is an example of convolution where its formula is expressed as in equation 2.10:

$$y[n] = \frac{1}{m} \times [x(n) + x(n-1) + \ldots + x(n-(m-1))] \tag{2.10}$$

To compute an output $y[n]$, the moving average filter uses m previous samples of the current n sample, as seen in equation 2.10, which computes the most likely value of the signal. Obviously, the larger is m, the more filtered is the output. However, a larger value of m increases the computation time, adding more delay. This can be an issue for a real time filtering application. Figure 2.6 illustrates the benefits and drawbacks of the moving average filter on the raw signal from the MU, shown in Figure 2.3. It also shows the one sample delay, in this example, introduced by the filter.

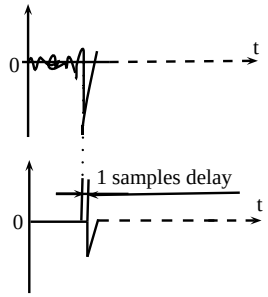


Figure 2.6: *Signal from one sensor after the use of the moving average filter. The filter removes the high frequency noise but add a delay of 1 sample*

The delay seems small, but to use properly a mouse cursor, the delay has to be zero or very close to zero. A mouse cursor on a screen has to be fast to respond to the will of the user. Another point is that the moving average filter, as efficient it could be, still leaves some noise. This small left noise can still produce small activations and the mouse cursor is then moving by itself. The method used to solve this problem is explained in Section 2.4.4.

The use of FIR is not fitting with the requirements of the project. The FIR filters, by performing a long convolution, introduced significant delay group which cannot suit with real time application. Moreover, the IIR filters are very close to analog electronic realizations that can facilitate the implementation on the board.

IIR Filters Recursive filters, such as IIR can perform a very long impulse response where only few coefficients are involved (without doing a very long convolution). In such a way they are executed very quickly but can become unstable because of the feedback. In actual practice, no more than about a dozen recursion coefficients can be used, otherwise the filter becomes unstable [20]. A IIR filter can be expressed as in equation 2.12

$$y[n] = a0 \times x[n] + a1 \times x[n-1] + a2 \times x[n-2] + a3 \times x[n-3] + \dots \tag{2.11}$$
$$+b1 \times y[n-1] + b2 \times [n-2] + b3 \times [n-3] + \dots \tag{2.12}$$
From [20]

where $x[n-i]$ are the values of the input and $y[n-i]$ is the output. The filter response (in frequency) can be studied by the z-transform of the filter's recursive equation [20].

An ideal filter (for instance a bandpass) removes elements with frequencies outside the interesting band and assures that all the frequencies in the pass band will not be distorted. This kind of filter is mathematically a window but physically, a infinite derivative cannot be performed and a non causal function cannot be done. The following analog filters are well known in the literature and can be summarized here.

Butterworth filters: Butterworth filters are very monotonic in the pass band and stop band but the roll off is not very sharp.

Chebychev filters: Chebychev filter is separated into two types. Type 1 introduces more pass band ripple than a Butterworth filter while Type 2 gives more stop band ripple. However, Type 1 and 2 have a stepper roll-of than a Butterworth filter. Chebychev filters are faster than Butterworth but the ripples on frequencies (pass or stop band) are parameters to avoid when accuracy is needed, as this project ask.

Bessel filters: The Bessel filter has a smooth pass band and stop band response, like the Butterworth. For the same filter order, the stop band attenuation of the Bessel filter is much lower than the Butterworth filter. It is not introduced any ripples but the Bessel filter gives more delay than a Chebychev or a Butterworth filter.

The project group tested these 3 recursive filters for the noise removal. Improvement could be seen in comparison with the moving average filter but a delay is still present.

Phase and group delay First of all to perform real time filtering, long group delays have to be avoided. When one sine wave enters in a filter, another one exits from it. There can be modifications of the amplitude and modifications of the phase. Comparing a same point in the input wave and in the output wave can define the phase delay. Suppose that $sin(\omega t)$ is the output signal, then $sin(\omega t - \phi)$ is the output wave with a phase delay $t_p = \frac{\phi}{\omega}$ as shown in the following computation, equations 2.13 - 2.15

$$sin(\omega t - \phi) = sin\ \omega(t - t_p) \tag{2.13}$$
$$\omega t - \phi = \omega(t - t_p) \tag{2.14}$$
$$t_p = \frac{\phi}{\omega} \tag{2.15}$$
From [21]

The phase delay is very useful to compute the time response of a filter when the delay is independent of the frequency.

The group delay is the time delay of the amplitude envelope of a modulation of 2 sinusoids. The group delay is only studied in a bandwidth where the phase response is approximately linear [22]. The group delay, t_g, is the slope of the linear phase around 0 Hz and it is equivalent to the time delay of the amplitude

envelope (because this group delay is only studied in a bandwidth where the phase is linear and a linear phase can be directly interpreted like a time delay), as seen in equations 2.16 - 2.21

$$yt = cos(\omega_1 t - \phi_1) + cos(\omega_2 t - \phi_2) \tag{2.16}$$

$$yt = 2cos\left(\frac{\omega_1 - \omega_2}{2}t - \frac{\phi_1 - \phi_2}{2}\right) cos\left(\frac{\omega_1 + \omega_2}{2}t - \frac{\phi_1 + \phi_2}{2}\right) \tag{2.17}$$

From [23]

Where:

$$cos\left(\frac{\omega_1 - \omega_2}{2}t - \frac{\phi_1 - \phi_2}{2}\right) = cos\left[\frac{\omega_1 - \omega_2}{2}(t - t_g)\right] \tag{2.18}$$

$$(\omega_1 - \omega_2)t_g = (\phi_1 - \phi_2) \tag{2.19}$$

$$t_g = \frac{(\phi_1 - \phi_2)}{(\omega_1 - \omega_2)} = \frac{\Delta\phi}{\Delta\omega} \tag{2.20}$$

$$t_g = \frac{d\phi}{d\omega} \tag{2.21}$$

2.3.2 Temperature drifts analysis

Another problem is introduced by the variation of the temperature. Depending on cold or hot temperature variation, the baseline of the signal, given by one sensor, drifts downwards or upwards, respectively.

2.3.2.1 Tested solutions to compensate the temperature drifts

Experiments, as illustrated in Figure 2.7, show that a decrease of the temperature makes the reduction of the baseline and that the increase of the temperature makes the baseline increase. However, there is an important latency before the baseline drift starts to occur. In fact, the alteration of the permeability of the sensors is not immediate and is different for the different sensors because all the sensors are not located in the same place on the palatal plate (the modification of the baseline is related to phenomena varying slowly, that means that very low frequencies almost stick with the continuous component of the signal spectrum). These considerations make the baseline wandering very difficult to remove without altering the useful information given by the user.

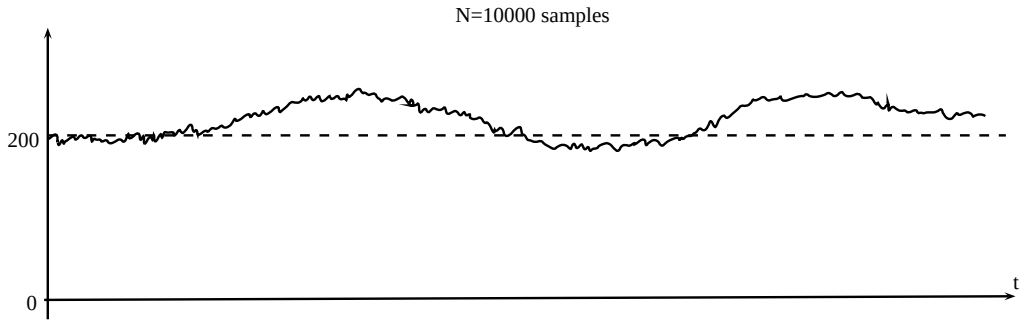


Figure 2.7: *Temperature drift of the raw signal coming from one sensor. It drifts upwards when the temperature increases, and downwards when the temperature decreases*

The moving average filter is useless to reduce these drifts, the noise is removed but the drift due to the continuous component of the signal cannot be cancelled. The use of a high pass filter is needed, like for

instance a high pass Butterworth filter.

Butterworth Filter A Butterworth filter is a IIR. Its frequency response in the pass band is exempt of oscillations. A Butterworth filter is also called "‘maximally flat magnitude filter"’. This property insures no ripples in the pass band and rolls off toward zero in stopband. The filter introduces a smooth filtering but also a delay.

In this project, a second order Butterworth high pass filter has been applied to the signal. The filter has a cut off frequency of 0.05 Hz. But it totally removes the low frequencies. The output signal appears disastrous because the filter removes almost all the activations (which are produced by user who cannot produce 100 activations per seconds which means 100 Hz). This problem is illustrated in Figure 2.8

Solution tested by HST member on a 8 bits microcontroller

Some built in logic (basically a threshold) is added to the filter to detect if the sensor is activated or not. If the signal was deemed "not activated" the filter was applied in the normal way. If the signal was "activated" then the filter was still applied to the signal, but the filter taps were not updated with new values. The effect is that the filter was "‘frozen"’ until the signal returned to baseline. This solution avoids the effect of the high pass filter.

The problems with this solution are:

- Small activations of the signal might not cross the threshold, causing the filter to remove signal activation anyway.

- If the baseline changes too fast it may cross the threshold and the filter may process the signal as if activated while it is actually not. The effect is that the baseline level escapes the filter, and thus are not removed.

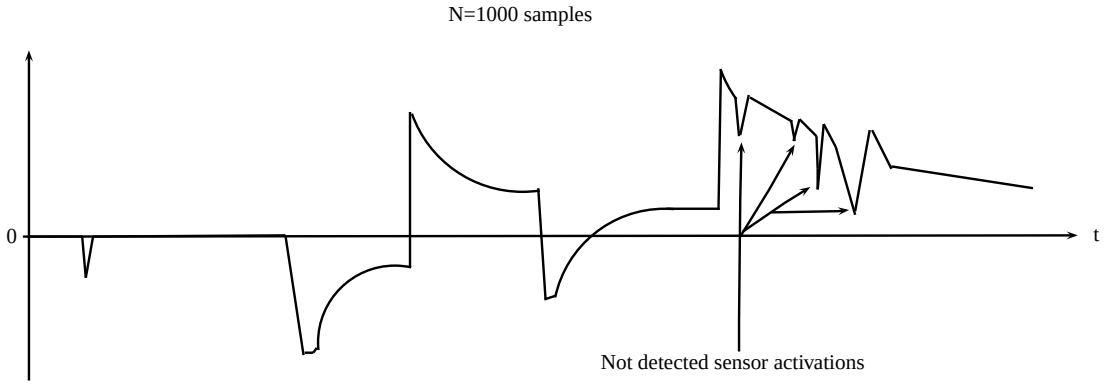


Figure 2.8: *When an activation appears, the high pass filter makes the signal going back to the average value. If users want a long activation on one sensor, e.g. going forward with their wheelchair, user will see its wheelchair stops after some times because the activation is no longer set. There is also a counter-reaction which sets the normal state of the sensor at higher amplitude, and then no activation could be detected for a while. With the example of the user wheelchair, it will stop quickly and could not start after seconds or a minute.*

In order to solve this problem more efficiently, the project group has investigated the possibility to use a Kalman filter, as described in Section 2.4.

2.4 Kalman filter based correction

2.4.1 Kalman filter overview

The Kalman filter is a set of mathematical equations that permits to estimate the state of a linear dynamic system from a series of noisy measurements [25]. The Kalman filter is one of the most efficient recursive filter based on the estimation of user defined states. Very used for removing noise as a smoother filter, it can also support estimations of past and present states but also predict future states based on previous or current observations. The first Kalman filter was performed to estimate the current state from a linear dynamic system corrupted by Gaussian noise [25]. The project system is corrupted by white noise and experiments prove that the Kalman filter is also very efficient with white noise.
The Kalman filter is based on filter theory but also on statistical decision theory. The mathematical foundational concepts are the least square theory, the dynamic systems theory and probability theory. Indeed, the first method for estimating a state from a set of noisy measurements is the method of least squares.

The least square theory can be resumed as follow:
Consider a problem for which the value of a state is unknown from a set of observations. The estimation error given by the Euclidean vector norm is performed. Then the error cannot be minimize directly but the variation of it has to tend to a negligible value which can be represented by the derivatives of this error with respect to the states estimation which has to be closed as possible of 0 [25].

Kalman filters are very used nowadays in GPS localization, geodesy but also in real time applications like the prediction of the trajectory of a high velocity object like in military field. The first applications of the Kalman filter were the control of dynamic systems states like in manufacturing processes, aircraft and ships [25].

2.4.2 Linear Dynamic Systems

Isaac Newton first introduced the concept of dynamic systems. With differential equations, mathematical models can be built in order to describe the dynamics systems around us. The motion of the planets in the solar system was described by Newton in the 17^{th} century, using a set of mathematical equations with only a small number of parameters which are related to the trajectory and also the velocity of these planets [25].

A dynamic system is composed of intercorrelated entities which evolve with time.
Suppose that u is the input of the system, y is the output and the state vector is called x. The internal representation of a linear dynamic system is:

$$\dot{x(t)} = Ax(t) + Bu(t) \tag{2.22}$$
$$y(t) = Hx(t) + Du(t) \tag{2.23}$$

where A is the evolution matrix (A is not linear if A is varying with respect to u)
B is the command matrix
H is the observation matrix
D is the direct transmission matrix
$\dot{x(t)}$ is the first order derivative of the state vector.

The usual resolution of the previous system is given by the following expressions, equations 2.25 and 2.25:

$$x(t) = e^{A(t-to)}x0 + \int_{t}^{t0} e^{A(t-\tau)}Bu(\tau)d\tau \tag{2.24}$$

$$y(t) = He^{A(t-t0)}x0 + \int_{t}^{t0} He^{A(t-\tau)}Bu(\tau)d\tau \tag{2.25}$$

where $x0$ is the initial conditions of the system.

In the case of a discrete system (like in this project case because the data are sampled before being analysed), differential equations become recursive equations:

$$x_{k+1} = \phi x_k + \delta u_k \tag{2.26}$$

$$y_k = \rho x_k + \theta u_k \tag{2.27}$$

Another representation of the linear dynamic system is also used in order to directly link the input and the output. If the initial conditions of the systems are equal to 0, and if the Laplace transform of the states equations and of the output is used then, yield the transfer matrix of the system called G in the equation 2.28 [26].

$$G(p) = H(pId - A)^{-1}B + D \tag{2.28}$$

And for a discrete system, the z-transform of the state and the output is as follow:

$$G(z) = \rho(zId - \phi)^{-1}\delta + \theta \tag{2.29}$$

2.4.3 The discrete Kalman filter Algorithm

In order to understand the Kalman filter theory, notations should be introduced (other notations can be found in other documents).

SYMBOL	VECTOR OR MATRIX NAME	DEFINITION OF NOTATIONAL USAGE
u	input vector	input of the system (directly controllable by the user)
z	measurement vector	measurement of available sensors in the system
x	state vector	vector used to describe totally the system
A	state transition model	applied to the previous state vector
B	control input model	applied to the control vector uk
H	observation matrix	describe the links between the measurements and the current states
w	process noise	Gaussian distribution with a covariance Q
Q	covariance matrix of the process noise	statically describe the noise w
v	sensor noise	Gaussian distribution with a covariance R
R	covariance matrix of the sensor noise	statically describe the noise v
P	Predicted estimate covariance matrix	measure the accuracy of the state estimate
K	Kalman gain matrix	Kalman gain matrix is chosen to minimize the a posteriori error covariance

Table 2.1: *Notations used in Kalman filtering*

The model underlying the Kalman filter is shown in Figure 2.9 where the circles are vectors, the squares are matrices and the other ones (v and w) are noises. In this model, one suppose that the noise corrupting the system can be split into 2 different Gaussian noise: the process noise w with the covariance Q and the sensor noise v with a covariance R. Indeed, in most physical systems the sensor noise which is introduced by the imprecisions of the sensors can be evaluated before running the filter. In fact, an off line model can be created in order to measure the sensor noise. More problematic to evaluate is the process noise defined as the approximations performed to pass from a previous state and the input to the current state.
In either cases, superior statically performances can be achieved by a better selection of the parameters Q and R [27].
In this project, the noise study is out of concerns because the noise is mostly caused by thermal agitations of the electrons or caused by electromagnetic sources which means that it can be assumed as a white noise not necessarily related to normal distribution. However, many dynamical systems are not exactly as described in the figure and Kalman filter can be very useful as well [28].
In the project case, the determination of R and Q is carried out by means of tests. Obviously, the optimal solution cannot be found theoretically.
The process and measurement noise are considered as white with normal probability distribution where the covariances Q and R are chosen by performing tests (the group tests different Q and R in an interval from 100 to 0.01). Suppose that the distribution functions of the noises v and w are following a centered normal law with different covariances Q and R such as shown in equations 2.30 and 2.31

$$p(w) \rightarrow N(0, Q) \tag{2.30}$$
$$p(v) \rightarrow N(0, R) \tag{2.31}$$

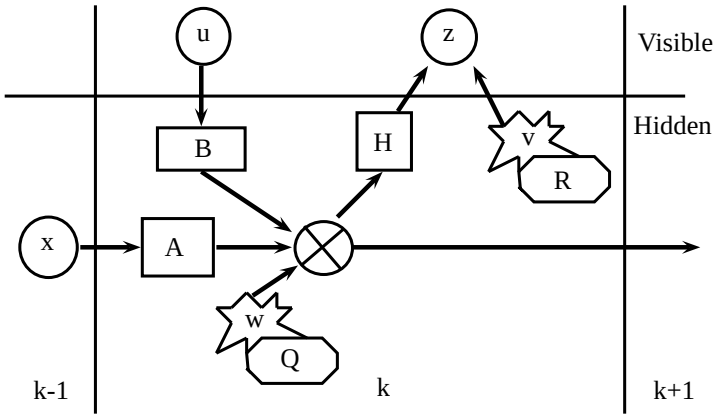


Figure 2.9: *Kalman filter principle*

Let's introduce other notations used to understand the Kalman filter:

SYMBOL	MATHEMATICAL DESCRIPTION	DEFINITION OF NOTATIONAL USAGE
x	state vector	state of the system
x_k	$x[k]$	k^{th} element of the sequence
$\hat{x}$	$E[x]$	estimate of the values of x
$\hat{x}^-$	$x_{k\|k-1}$	estimate of x conditioned by all measurements except the one at time t_k

Table 2.2: *Other notations used in Kalman filtering*

The posteriori estimation covariance is given by the formula 2.32 and it is the covariance of the error made between the estimator and the real value of the state

$$P_k = E[(x_k - \hat{x}_k)(x_k - \hat{x}_k)^T] \qquad \equiv E[e_k e_k^T] \tag{2.32}$$

The posteriori state estimate reflects the mean of the state based on all the previous measurement: $E[x_k] = \hat{x_k}$

Finally the definition of the Kalman gain is fundamental, it is chosen to minimize the a posteriori error covariance [28]:

$$K_k = P_k^- HT(HP_k^- H^T + R)^{-1} \tag{2.33}$$

The Kalman filter is based on a recursive algorithm, hence it uses a feedback control. The principle of the filter can be divided in 2 different set of equations: time update and measurements. First the filter performs an estimate of the process, and then it obtains the values of the sensors corrupted by noise in order to be compared with the previous estimation [27].

- Time update equations are used to project in the next step the estimate of the previous state vector and also the covariance in order to obtain an a priori estimates for these 2 values that can be corrected after.

- Measurements updates equations are responsible of the feedback, that means the a priori estimate of the state and the covariance are compared to the measurements and then a more precise a posteriori estimate of the state and the covariance is found. The importance of the measurement and the matrix of observability H is fundamental in order to obtain new data.

The following equations can be implemented in Matlab in order to obtain a recursive Kalman filter. They are the basis of the Kalman filter and can be split into 2 different set as specified previously:

Discrete Kalman filter time equations [27]

$$\hat{x_k^-} = A\hat{x_{k-1}^-} + Bu_{k-1} \tag{2.34}$$

$$P_k^- = AP_{k-1}A^T + Q \tag{2.35}$$

Discrete Kalman filter measurement update equations [27]

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \tag{2.36}$$

$$\hat{x_k} = \hat{x_k}^- + K_k(z_k - H\hat{x_k}^-) \tag{2.37}$$

$$P_k = (Id - K_k H)P_k^- \tag{2.38}$$

Figure 2.10 shows the principle of the recursive Kalman filter Algorithm [27].

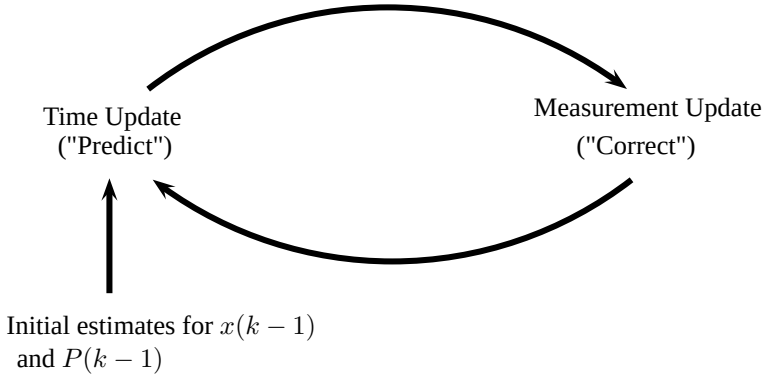


Figure 2.10: *Operations done by the Kalman filter*

2.4.4 Fitted Kalman filter to remove noise and baseline wandering

The main two problems cannot be solved directly by the use of a single low pass system. Indeed, the Kalman filter has the ability to easily remove noise without adding a delay in opposition to other non dynamic system based low pass filters. However, the baseline wandering is caused by low frequency temperature drift and the use of a high pass filter is needed that completely change the signal. The group creates a Kalman filter with its own set of equations to remove the noise but also the baseline wandering.

- Kalman Filter Noise removal

  The system needs to work in real time that means that no delay can be added to the signal but also that the state vector cannot contain too much previous states and obviously not all the previous states (from the start of the simulations). In consequence, a state vector with only the current state

is needed to obtain fast results.
The state vector is defined by the value of the sensor without noise, the measurement z_k is defined by the real value of the sensor coming from the bitstream, the observation matrix H is naturally equal to 1 as the state transition matrix model A (which means that the next state is based on the previous state before the measurement update, it can be seen as the memory of the filter which needs a value to compare with the new measurement). The user cannot do any action to remove the noise that means that the input uk is equal to 0 as the B matrix.
Suppose that $\hat{x_k^-}$ and $\hat{x_k}$ are initially set to 0 because of the causality of the system.
Then statistical relations of the filter have to be set.
The definition of Q and R are set randomly at the beginning and performing many tests shows that the best tradeoff between noise removal and identical signal form after filtering can be reach for $Q = 0.1$ and $R = 0.1$.
The value of P is set to $10e^5$ at the beginning to quickly converge towards a realistic value.
Then the time update equations followed by the measurements equations are performed in an infinite loop to perform the filtering.
The difficulty of the Kalman filter is to define precisely the dynamic system and the statistical initial conditions (summarized in equations 2.40 - 2.45)

$$A = 1 \; and \; B = 0; \tag{2.39}$$
$$H = 1 \; and \; D = 0; \tag{2.40}$$
$$z_k = \text{value of the sensor}; \tag{2.41}$$
$$uk = 0; \tag{2.42}$$
$$\hat{x_0^-} \; and \; \hat{x_0} = 0; \tag{2.43}$$
$$P = 10e5; \tag{2.44}$$
$$Q = 0.1 \; and \; R = 0.1; \tag{2.45}$$

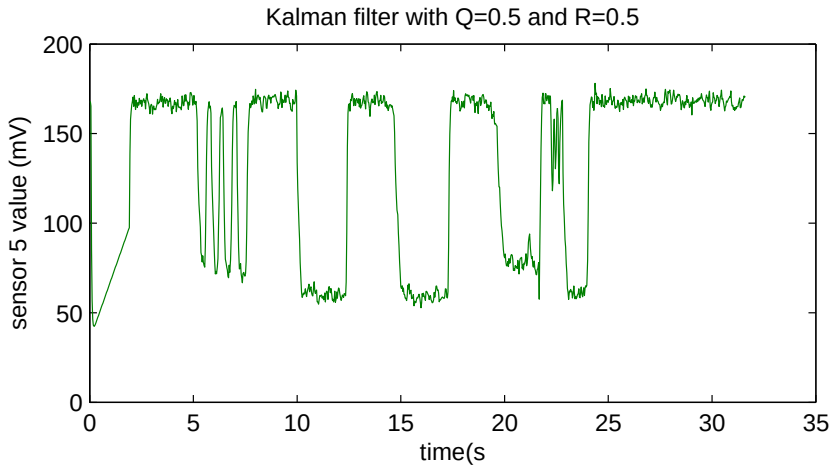


Figure 2.11: *noise suppression with Kalman filter*

The noise removal is highly linked to the covariances of the noises Q and R. Higher are the values of Q and R, higher is the filtering effect and even if no delay is added, the small activations will not

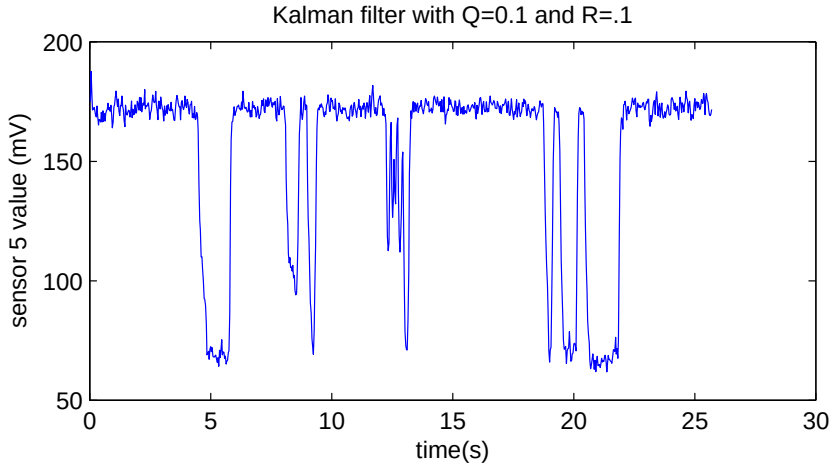


Figure 2.12: *noise suppression with Kalman filter(other covariances)*

be detected.

- Kalman Filter Noise and Baseline wandering removal
  The Kalman filter baseline wandering removal is really more complex because a careful analysis of the system is needed. In this section, the group supposes that the Baseline Shift (BS) can be calculated and available for the computation.
  The next part discusses the computation of the BS and the approaches to obtain it.
  First of all the state vector has to be updated: now, the two informations for describing the system are the Real Value of the Sensor Without Noise (RVSWN) and the BS. In consequence, all the matrices of the system have to be modified.
  The transition matrix model is now equal to $Id_{2\times2}$, it means that the next state is based on the previous state. The value of the input matrix B is equal to $Id_{2\times2}$ but this matrix can also be equal to the null matrix because as in the previous case the user has no command to change the RVSWN or the BS. More complex is the observation matrix H.
  The observation matrix H is used to compare the actual state vector with the measurement vector. The measurement vector has 2 elements: the first one is the Value of the Sensor Received from the System (VSRS) and the second one is the Estimator of the BS (EBS) built as explained in Section 2.4.5.
  In consequence, the equations 2.47 - 2.49 can define the observation matrix H, equation 2.49:

$$H * \hat{x_k} = z_k \tag{2.46}$$

$$\equiv H * \begin{pmatrix} RVSWN \\ BS \end{pmatrix} = \begin{pmatrix} VSRS \\ EBS \end{pmatrix} \tag{2.47}$$

$$\equiv \begin{pmatrix} RVSWN + BS = VSRS \\ BS = EBS \end{pmatrix} \tag{2.48}$$

$$\Rightarrow H = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \tag{2.49}$$

  The addition of the RVSWN and the baseline is equal to the VSRS and the EBS has to be as close as possible to the BS.

Because the statistical values of the sensor determines the noise parameters, the statistical values are the same but now the dimension of the matrices has changed. In consequence, $P = 10e^5 * Id_{2*2}$, $Q = 0.1 * Id_{2*2}$ and $R = 0.05 * Id_{2*2}$.
Obviously, the value of the state vector is 0 at time 0 as the value of the measurement z.
The equations - summarize the parameters of the filter in order to remove the correction and the baseline wandering:

$$A = B = Id_{2\times 2} \tag{2.50}$$
$$Q = 0.1 \times Id_{2\times 2} \text{ and } R = 0.05 \times Id_{2*2} \tag{2.51}$$
$$P = 10e5 \times Id_{2\times 2} \tag{2.52}$$
$$x_0 = x_0^- = z_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \tag{2.53}$$
$$z_k = \begin{pmatrix} \text{Value of the Sensor Received from the System (VSRS)} \\ \text{Estimator of the Baseline Shift (EBS)} \end{pmatrix} \tag{2.54}$$
$$x_k = \begin{pmatrix} \text{the Real Value of the Sensor Without Noise and baseline shift (RVSWN)} \\ \text{the Baseline Shift (BS)} \end{pmatrix} \tag{2.55}$$
$$H = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \tag{2.56}$$

One of the main advantages of the Kalman filter is that there is no delay added, the activations stay very sharp and also that all the computations are performed without introduction of logic i.e. without interruption in the filtering.

2.4.5 Recursive definition of the baseline shift

The removal of the baseline shift can be improved by the removal of all the baseline in order to center the values of each sensor to 0. Indeed the values of the sensors without any activation are not the same and that could be an issue for the detection of the activation based on a threshold. In such a way the removal of the baseline shift but also the removal of mean values of each sensors without activation are advised.
The description of the baseline, defined by an algorithm made by the group, is made recursively, it means that the previous value of the baseline is saved (called $baseline^-$). The principle of the baseline correction made by the group is based on an evaluation algorithm which detects or not the activations.
The signal coming from the sensor passes through a high pass filter with a very low cutoff frequency, it means that in time domain during an activation the values of the filter will make a peak and return very quickly to 0. The difference between the output of the high pass filter and the signal coming from the sensor can be considered as the baseline except during the activation or the deactivation of the sensor (when the output of the filter makes a peak). During these activations, the value of the baseline can be hold only if the activations are very fast. In the other cases, during the activation the baseline shift shouldn' t be present and can be catastrophic for the signal. In fact, if the baseline shift has a negative derivative, there are no problems because the activations will remain, but if the baseline shift has a positive derivative, deactivations can appear because the baseline shift would carry the signal above the threshold detection value. In that case, a correction coefficient should be added to the constant baseline in order to correct the shift during the activations.
A logical cell is built by the group in order to detect the situation and to add the adapted correction.
Figure 2.13 summarizes the principle underlying the correction system.

Both the baseline recognition system and the logical decision system have been created by the group.

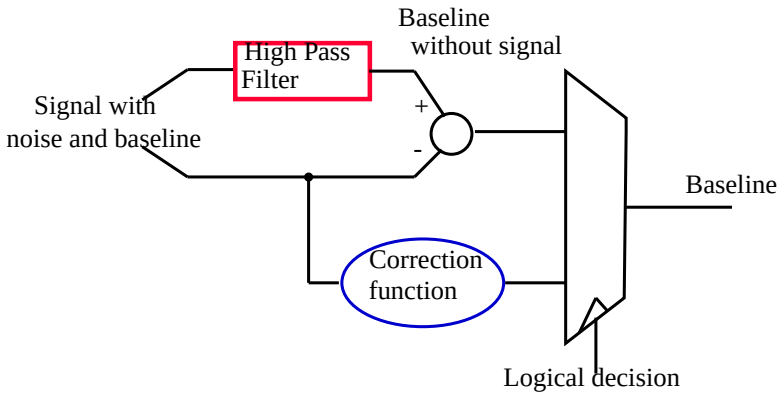


Figure 2.13: *The estimation of the baseline shift performed after the logical decision is made*

2.4.5.1 Logical decision algorithm

The logical decision algorithm is based on the detection of the activations with 2 different thresholds. Obviously those values depend on the high pass filter used and can be tuned easily. Moreover, the algorithm is not linked to the noise filtering process, which allows it to be tuned easily. It is noticeable also that this algorithm can be improved by performing a large number of tests in order to cover all the possible situations.

Definition of the variables:

- $coefficient$ is the value of the correction added during an activation
- z_k is the value of the measurement (=value of the sensor signal)
- X_hat_hp is the output of the high pass filter applied to the signal
- $activation$ is a boolean variable which takes 1 during an activation, 0 otherwise
- The value of $detect$ is given by a running mean of the measurement
- $detect\_fast$ is a detector, using the output of the high pass filter, aimed to detect the peak, it means the activations (the presence of the 2 detectors is justified by the big number of activation forms (long, very high frequency, very repetitive,...).
- $t1$ is the threshold linked to the detector $detect$
- $t2$ is the threshold linked to the detector $detect\_fast$

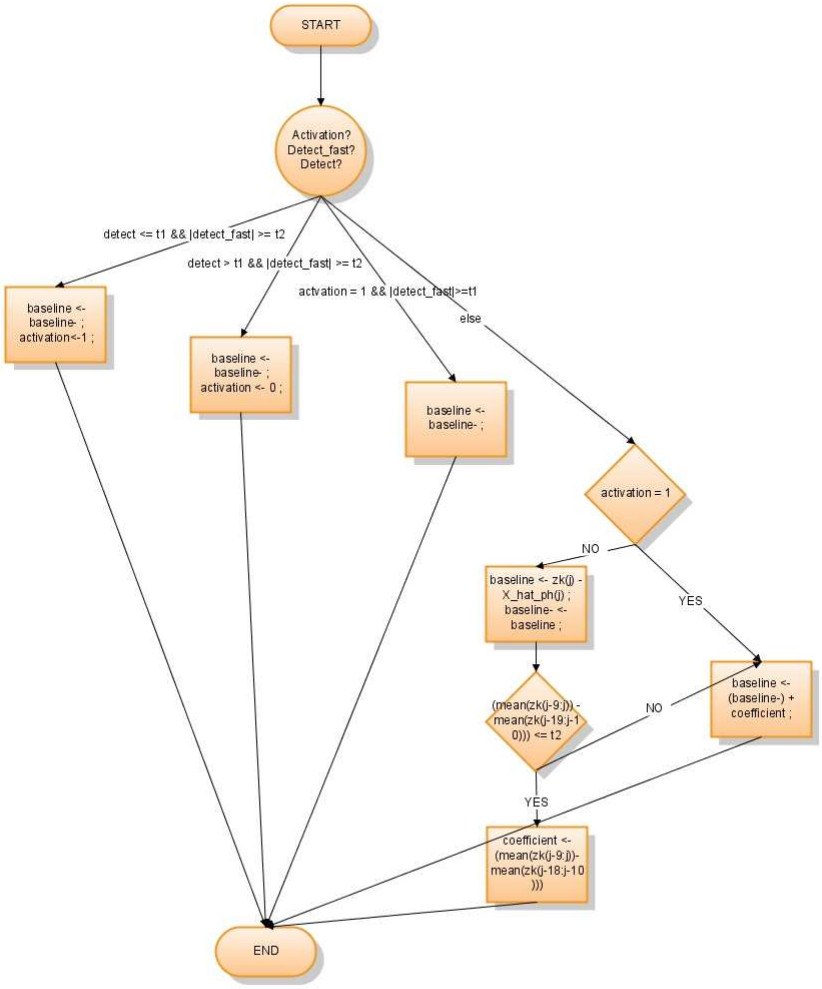


Figure 2.14: *Logical Algorithm Description*

The upper flow chart in Figure 2.14 shows the principle of the Algorithm made by the group.
When $detect$ is larger than $t1$ and $detect\_fast$ is larger than $t2$, an activation is detected. The baseline cannot be updated but has to keep the previous value $(baseline-)$. In fact during an activation, the value of a sensor goes from 200 to 150 for instance. The baseline variable cannot be updated because the true value of the baseline is not 150 but still 200
A deactivation is detected when $detect > t1$ and the absolute value of $detect\_fast$ is larger than $t2$, in this case also the baseline as to be kept as its previous value $(baseline-)$.
Very quick activations can exist but in this case only $detect\_fast$ should be trusted.
The last case are when there are no activation or that a very long activation is performed.

- First of all, the case where there is no activation:
  The baseline can be defined. For example when there is no activation, the baseline is still 200 and

can be trusted. To define it the group choose to compare the value of the sensor with the value of the high pass filter (the value of the high pass filter is 0 when no activation is performed). Moreover the value of $(baseline-)$ is updated.

- Then the case where a long activation is detected but a drift could exist, so the addition of a correction coefficient based on the slope of baseline when there are no activation, is needed.
  Only for long activations, the baseline shift (due to temperature variations) is corrected, not for short activations.
  The negligible error introduced will be corrected during the deactivation because the output of the high pass filter will be flat, so the baseline can be defined very precisely.

The first idea of the algorithm was to run the input signal in parallel with this one passing through a high pass filter. In such a wy the difference between the two signals gives the baseline, but this value is reliable only when no activation is detected. The way that this idea is realised, to save memory space because only the current baseline is stored, introduces a very small error in the baseline definition but the delay is almost equal to zero.
However, this logical decision function should not be created separately from the data reading and the Kalman filter processing, because many parameters are needed and have to be saved. Instead of that, this function should be integrated in the main program which contains the infinite loop used to read the input values from the MU.

2.4.5.2 Butterworth High Pass Filter

The discussion about Buttherworth filter is already done Section 2.3.1.3 with the presentation of its advantages and drawbacks. The requirement for the high pass filter used in the baseline computation is first of all a very small time response, that means that the filter has to return to 0 as quickly as possible (minimum delay). This ensures to maintain a value to compute the baseline which cannot be the case for a high pass filter with very big time constant. For a first order low pass filter the time constant is linked to the cutoff frequency by the following formula, equation 2.57:

$$\tau = \frac{1}{2 \times \pi \times f_c} \tag{2.57}$$

where τ is the time constant and f_c is the cutoff frequency.
Obviously the relations between the cutoff frequency depends on the kind of filter and the order of this one, but for all the cases higher is the cutoff frequency, smaller is the time constant, faster is the return to 0 of the high pass filter.
The high pass filter should have the highest cutoff frequency. Because there is almost only noise above 5Hz, as explained Section 2.3.1.2, the group decides to set the cutoff frequency very close to 5Hz.
Moreover, as the IIR filters become in practice unstable for an order above 12, the group decides to limit at maximum the time domain oscillations during the return of the high pass filter to the value 0, by choosing an order 2 Butterworth filter, as seen in Figure 2.15. Butterworth filters are well known in the literature [17], and they can be easily implemented in hardware with different topologies like Sallen & Key [29] or Cauer [30].

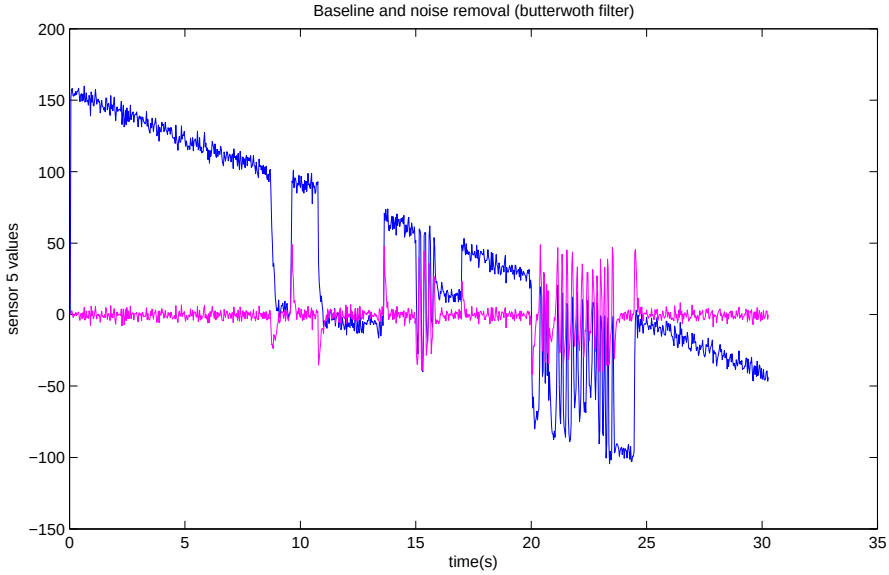


Figure 2.15: *The curve in blue is the input signal with a important shift and the curve in pink is the Butterworth high pass filter running in parallel. A peak with a positive derivative can be noticed after a deactivation and a peak with a negative derivative can be noticed after an activation*

Notice that Butterworth is a recursive filter therefore it can be studied as a linear dynamic system [31].

2.4.5.3 Kalman based High Pass Filter

The study of the Kalman low pass filter is presented Section 2.4.3. However, no study regarding the frequency response of the Kalman filter is done in this document. Because of very low frequency signal and the very interesting time domain properties, especially because no delay is added after filtering especially for the activation, the group limits the study of the Kalman filter only to the time domain. The construction of a high pass filter based on a low pass kalman filter, performed by the group, is new and can be called by extension the Kalman based high pass filter.
For digital filters (manipulating sampled data as here) the creation of a high pass filter can be easily done from a low pass filter. The low pass filter is supposed to ensure that all the signal in the frequencies in the low frequency band will be not distorted and all the frequencies outside this band will be attenuated. For a high pass filter it is the same, but this time all the signal in high frequencies has to be kept unchanged.
In consequence, a high pass filter of the same cutoff frequency can be designed from the low pass filter. One all pass filter, for instance the delta Dirac function (the frequency response of the delta Dirac function is the constant one in all the frequencies) subtracted by a low pass filter running in parallel, give a high pass filter, as shown in Figure 2.17
Obviously the best all pass filter is the input signal itself if the low pass filter introduces no delay as the Kalman filter does. This is why the use of the delta Dirac function is required.

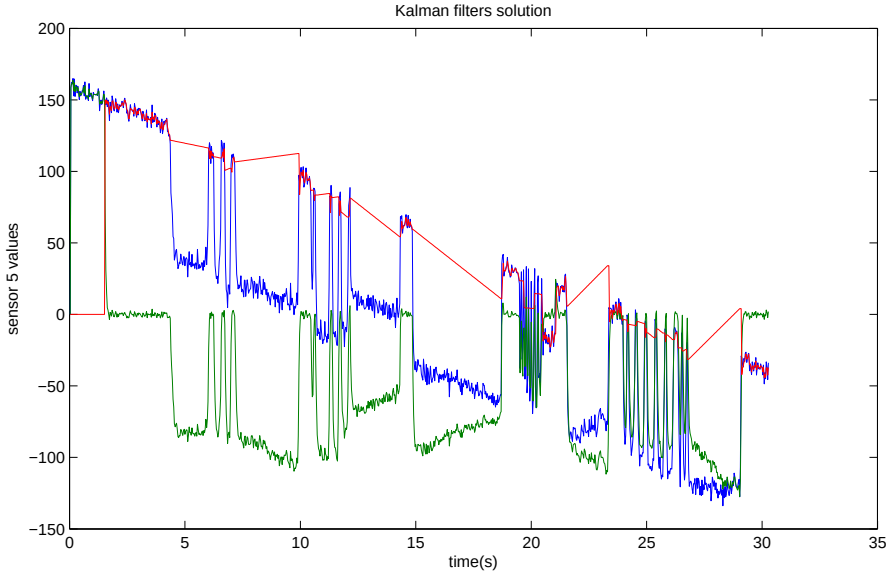


Figure 2.16: *The received signal from the sensor 5 is shown in blue, the baseline estimator in pink and the output of the Kalman filter in green*

However, this high pass filter is practically not the same as a real high pass filter. In fact the real high pass filter have a zero attenuation in the pass band. This high pass filter cannot have a zero attenuation on the pass band because it is the difference between the all pass filter subtracted with the low pass. Obviously, a realistic low pass filter cannot have a zero gain in the stop band.
The principle of the high pass Kalman filter is summarized in Figures 2.17 and 2.18

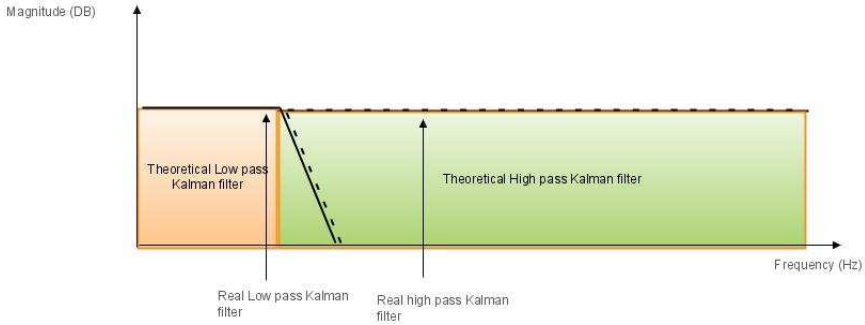


Figure 2.17: *The transfer function of the Kalman filter is theoretically equal to a high pass filter but in practice, the low pass filter cannot be rectangular, so the high pass filter does not have a zero attenuation in the pass band*

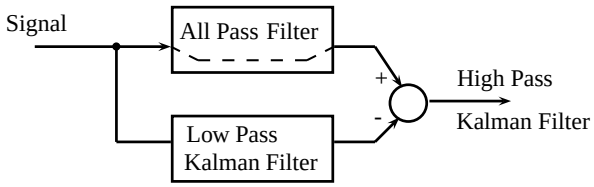


Figure 2.18: *The principle of the created Kalman filter is based on the substraction of 2 filters running in parallel*

Equation 2.58 shows the mathematical principle of the construction of the high pass filter.

$$y(n) = \delta(n) - lp(n) \tag{2.58}$$

where y is the output of the high pass filter and lp is the output of the low pass filter.

The filtering of the signal, received from a sensor corrupted by the noise and the baseline wandering, with the created Kalman high pass based filter is shown in Figure 2.19.

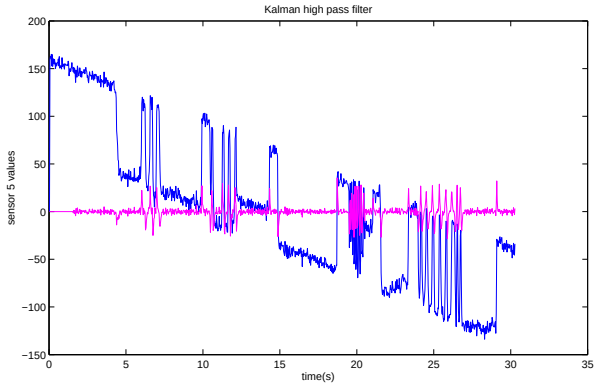


Figure 2.19: *The blue curve is the signal corrupted by noise and the baseline wandering, pink signal corresponds to the Kalman based high pass filter. The figure shows that there is no delay added between the activations, and the deactivations, and the peaks of the Kalman based high pass filter.*

Contrary to the Butterworth high pass filter, this high pass filtering is delay free, which permits to the decision algorithm to be executed in parallel of the received signal.
For a better correction, the description of the covariance of the noise called Q_{hp} and R_{hp} for a Kalman based high pass filter are different from the Q and R of the low pass Kalman filter because the group wants the maximum reactivity of the high pass filter, so the noise removal is not taken into consideration for the Kalman High pass filter.

2.4.5.4 Graphical description of the correction

First of all, the group investigates these correction methods with only one sensor and then extends these corrections to the 28 sensors.

1. Correction with only one sensor
   The summary of the correction with only one sensor is done here. The Kalman based high pass

filter and the logical algorithm are created with the aim to obtain the baseline estimator as accurate as possible. Then this estimator of the baseline and the measurement of the output without correction builds up the measurement vector z_k which is the only external data of the low pass Kalman filter. The output of the low pass Kalman filter is the signal with a minimum of noise corruption and baseline wandering correction.
Figure 2.20 shows the external values used for this filtering and the feedback performed by the Kalman algorithm.

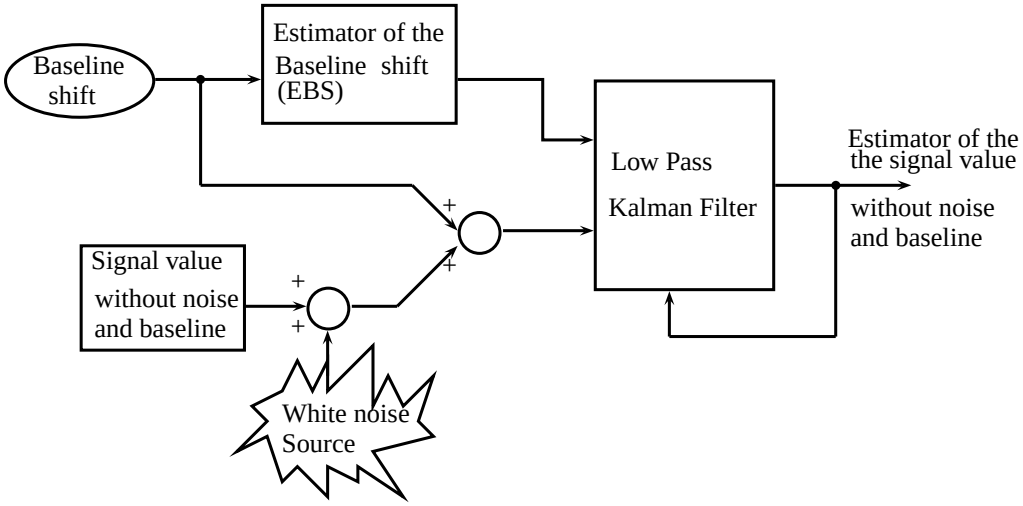


Figure 2.20: *External values used for the low pass Kalman filter*

The baseline estimator is subtracted automatically by the Kalman filter from the signal in order to obtain a signal centred around 0 without baseline wandering as shown in Figure 2.21.

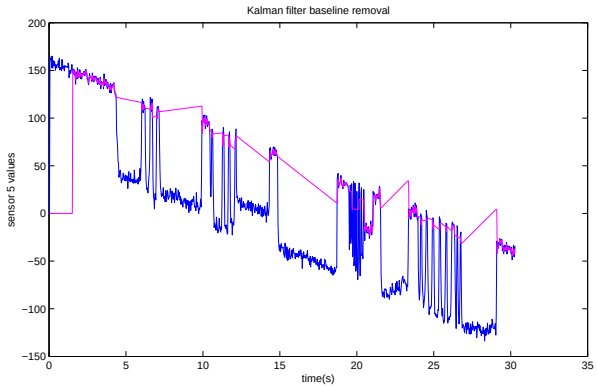


Figure 2.21: *Values received from the sensor 5 (in blue) and baseline estimator (in pink). The baseline estimator is not accurate during the activation but, as said previously, this is not a big issue and can be easily tuned by modifying the coefficient*

The baseline estimator does not follow exactly the baseline of the signal especially during the long

activations. The correction coefficient can be tuned in order to follow more precisely the baseline. However, this is not a big issue because the baseline estimator returns to the right value during the deactivation.
The creation of this Kalman filter and all the baseline estimators (except the one using the Butterworth high pass filter) is done without using any pre existing functions but only with recursive equations. The translation of this Matlab code into C code, as described in Chapter 5 is easier without using any Matlab functions.

2. Correction with all the sensors:
The correction of all the sensors can be done easily. There are 2 ways to correct all the 28 sensors:

   - The simplest way is to set 28 Kalman filters in parallel with 28 baseline estimators.
   - The second way is to define another Kalman filter. The state vector of this new Kalman filter has to be built up with 28 values of each sensor without noise and without baseline and the 28 baseline estimators. Obviously all the relatives matrix to perform the computation have to be made in consequence. Figure 2.22 shows the principle of the Kalman filter for all the 28 sensors and the equations of this one are defined, equation 2.59.

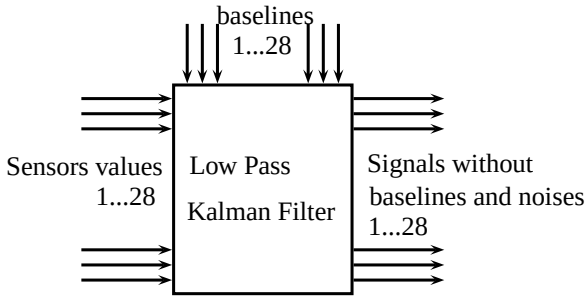


Figure 2.22: *One single Kalman filter can be used for all the 28 sensors*

$$x_k = \begin{pmatrix} value_{\text{without noise and baseline wandering}} sensor1 \\ \dots \\ value_{\text{without noise and baseline wandering}} sensor28 \\ baseline_u sensor1 \\ \dots \\ baseline_u sensor28 \end{pmatrix} \tag{2.59}$$

$$A = B = Id_{2\times 28} \tag{2.60}$$
$$Q = 0.1 \times Id_{2\times 28} \tag{2.61}$$
$$R = 0.05 \times Id_{2\times 28} \tag{2.62}$$
$$H = \begin{pmatrix} 1_{28\times 28} & 1_{28\times 28} \\ 1_{28\times 28} & 0_{28\times 28} \end{pmatrix} \tag{2.63}$$

The first method is chosen by the group because it uses less hardware resources than the previous one.

2.4.5.5 Simulation results

Figure 2.23 and Figure 2.24 show the results before and after applying the Kalman filters on the signal. Figure 2.23 is obtained with a Kalman filtering and a Butterworth high pass filter used to obtain the baseline estimator. Figure 2.24 is obtained with a Kalman filtering and a Kalman based high pass filter to build the baseline estimators. The baseline shift is purely added in real time whereas the noise comes from the received values of the sensor. **The test carried out in this project shows that the baseline shift correction is performed correctly, no matter how the slope of the shift is.**

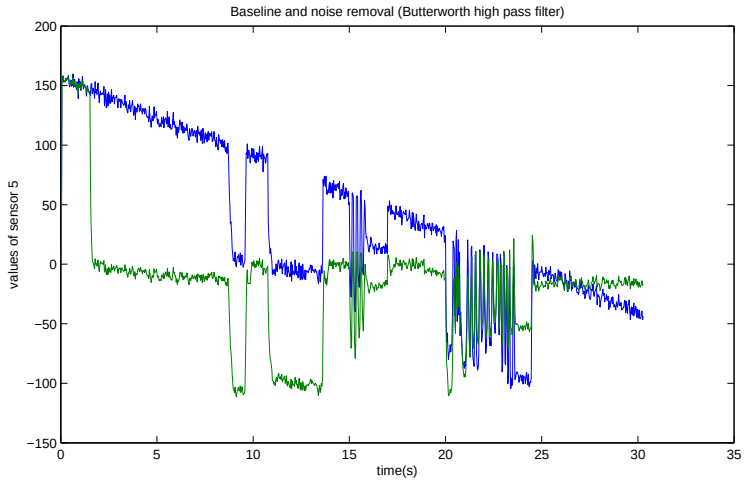


Figure 2.23: *Received signal (in blue) and Kalman filtering using Butterworth high pass filter for the estimation of the baseline (in green)*

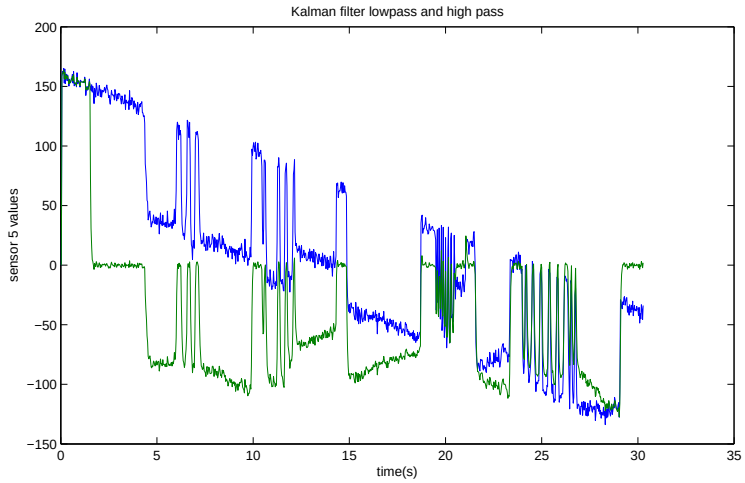


Figure 2.24: *Received signal (in blue) and signal after filtering, with high pass Kalman based filter as an estimator of the baseline (in green)*

The correction with Kalman based high pass filter is advised for the following reasons:

- there is no delay added which is of primary importance for a real time application
- there are no time domain oscillation after a peak to return to the 0 value
- long series of high frequency oscillations are less problematic than in Butterworth filter solution. Tests shown that the baseline estimation has more realistic value in Kalman based high pass filter than in Butterworth high pass filter.
- Very quick activations are detected more easily

2.5 Conclusion about the analysis and signal processing

When the user is performing a move of the AU on the sensors on the palatal plate, the MU sends to the CU the state of each sensor, including the ones activated. The transmission is performed by the digital radio.

The received signals present high frequency noise perturbations. The previous research highlights that the noise is due to the scanning of the sensors, that is the variation of the inductance of the coil which depends on its temperature and the Electro Magnetic Compatibility (EMC) interferences. This noise is removed because it may cause voltage drops which could be considered by the CU as activation of a sensor while no activation is actually asked. To remove the noise, the group project applied the most common and simplest FIR low pass filter, the Moving Average Filter. But its execution on MATLAB shows the introduction of a delay between the raw signal and the filtered one. For a real-time application which requires accuracy and reactivity in relation with the user actions, this delay is not acceptable. The project group tested different common IIR low pass filter as well like Butterworth, Chebychev and Bessel. These recursive filters can be executed very quickly but still introduces unacceptable delay.
Literatures about noise removal speak about the Kalman filter and its efficiency, in terms of time response and noise voltage attenuation. The project group analysed this filter and successfully developed a Kalman filter noise removal.

Other experiments show that the variation of temperature of the MU environment brings the sensor signal to shift up, in an increasing temperature variation environment (hot foodstuff) and shift down in a decreasing temperature variation environment (cold foodstuff). These shifts occur on the baseline of the signal, i.e. the continuous component or the fundamental of the signals. It means that if an activation of a sensor occurs whereas the environment temperature increased, this activation could not be detected because it would not reach the detection threshold. Conversely, if the environment temperature decreased, the CU can consider the baseline shift as an activation. The previous research team used a Butterworth high pass filter to remove the baseline shifts. But this solution introduced an unwanted effect on long activations. Indeed, during a long activation, the filtered signal tends to go back to 0 and when the long activation finishes, a counter-reaction peak appears which tends to go back to 0 as well but slowly. During this slow decrease, activations cannot be considered by the CU as their voltage does not reach the threshold. Then, the research team tried to create an algorithm where the high pass filter is not working during long activations but this solution does not remove the shifts during long activation which can cause the previously mentioned issues.
The project group, after analysis of the Kalman filter, made assumptions that this filter could remove the baseline shifts. The project group, then, successfully created a custom high pass based Kalman filter established from the low pass Kalman filter.

As a real-time application, the processing of the data packets sent by the MU to the CU needs to be delay free, without noise and any other external perturbations. Literatures about filtering mention the

Kalman filter as an optimal recursive data filter. It estimates the state of a linear dynamic system from a series of noisy measurements. Kalman filter is used as an estimator of future state of a system, from past and present state. This possibility to estimate within an efficient method is useful for many applications like GPS.
For this project, the noise must be removed without adding any delay. Using MATLAB, an algorithm is produced, following the theory equations. The Figures in Paragraph 2.4.4 show the efficiency of the Kalman filter in the noise removal process.
A custom Kalman filter producing a high pass filter is created to remove the baseline temperature shifts. Based on the low pass Kalman filter and an all pass filter, the high pass based Kalman filter is created. The Figure 2.16 in Paragraph 2.4.5.3 shows the efficiency in removing the baseline shifts of the high pass Kalman filter.

Then once the noise and the baseline shifts are removed by means of the Kalman filter, simulations are successfully performed on the 28 sensors of the MU where no delay and no shifts are observed on the filtered signals.

2.5.1 Possible improvements of the Kalman filter algorithm

The tests made by the group are insufficient to quantify or qualify the solution. Not all the cases are tested i.e. in not all the possible environments of the tongue system. However, the solution found shows very promising results and the logic besides this solution is minimal, it means easy to reuse, modify or even upgrade. The theory underlying the low pass Kalman filter should be studied more, especially in the frequency domain (although it is an unusual approach of the Kalman filter [25], but can be a good starting point for the comparison with other filters).
Finally the statistical theory and the study of the white Gaussian noise should be completed in order to define mathematically the covariance of the noise process Q, the covariance of the noise sensor R of the Kalman low pass filter but also the covariance of the noise process Q_{hp} and the covariance of the noise sensor R_{hp} concerning the high pass based Kalman filter.

Chapter 3

Algorithms Analysis

This chapter is relative to the Algorithmic domain of the A^3 design methodology, especially to the analysis of the existing algorithms for controlling the external devices, highlightened in Figure 3.1. This chapter introduces the algorithms mentioned in the second sub-goal defined Section 1.2:

2. Executing the algorithms of communication between the other sub-systems of the TCS and the external devices.

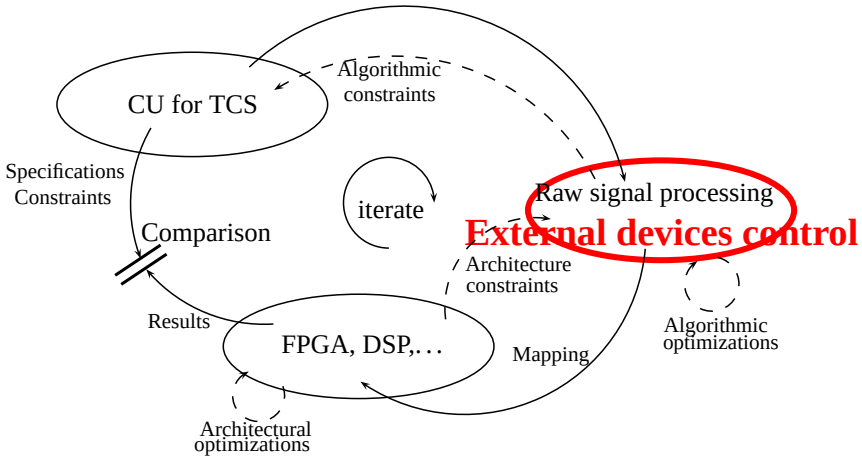


Figure 3.1: *The A^3 design methodology applied to Chapter 3*

3.1 Overview

The HST members have developed a prototype of the TCS where the CU is realised by a small PCB board, on which a 8 bit microcontroller ensures the processing of the raw signal provided by the MU and the communication interface with a computer mouse cursor, illustrated in Figure 1.4, paragraph 1.1.1.3. This chapter introduces the algorithms used in this prototype. The algorithm of the radio establishing the communication between the MU and the CU is presented Section 3.2. It is followed, Section 3.3, by

the one creating the coordinates to emulate a joystick and finally the one which, from these coordinates, permits the control of the mouse cursor in Section 3.4. These algorithms have all been translated in C language before their implementation on the microcontroller.

3.2 The radio communication

The MU sends the state of each sensor to the CU every 30 milliseconds by means of 28 byte data packets. The CU receives these data packets, processes them and sends the instruction to the selected external device. In the case of the user wants to control the computer mouse cursor, the CU calculates the coordinates (x and y) the mouse cursor has to follow.

The radio transmitter implemented in the MU is a nRF24L01 by Nordic Semiconductor. The CU prototype developed by the HST members is using the same device for the receiver part, so that, the communication protocol is easier to develop. It is interfaced with a SPI bus to read out the data. Nordic semiconductor provides many C codes for correct use of the device. The algorithm for the radio receiver of the CU is showed in Figure 3.2

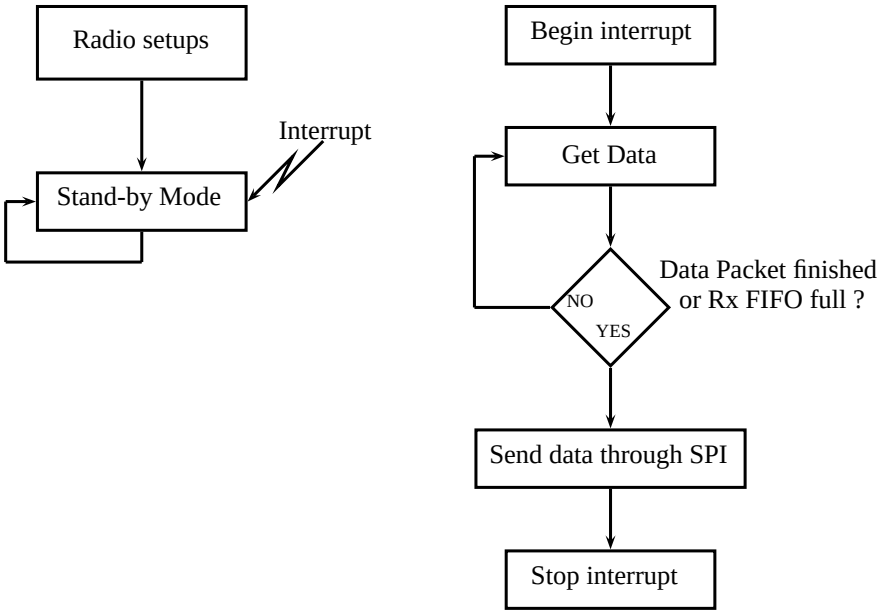


Figure 3.2: *The algorithm for the radio receiver. The receiver part is waiting until a data packet interrupt occurs. The data are placed in a Receiving (Rx) First In First Out(FIFO) buffer until the end of the packet or the buffer overflow. Finally, the collected data are sent through the SPI bus to the microcontroller. Then, the radio receiver goes back to the stand-by mode until another interrupt occurs.*

3.3 Creating the coordinates

After receiving the different state of the sensors by the radio receiver and applying the signal processing described in Chapter 2, the next step for controlling a computer mouse cursor is the creation of the joystick, i.e. the x and y coordinates.

The palatal plate on the MU has 8 sensors which are used to emulate a joystick, as illustrated in Figure 3.3. When the MU is sending the state of each sensors to the CU, the states of these 8 sensors are used as inputs for the Inference system of the CU. This system is using the $Fuzzy\ Logic$ to compute the coordinates, as decribed in 3.3.2.

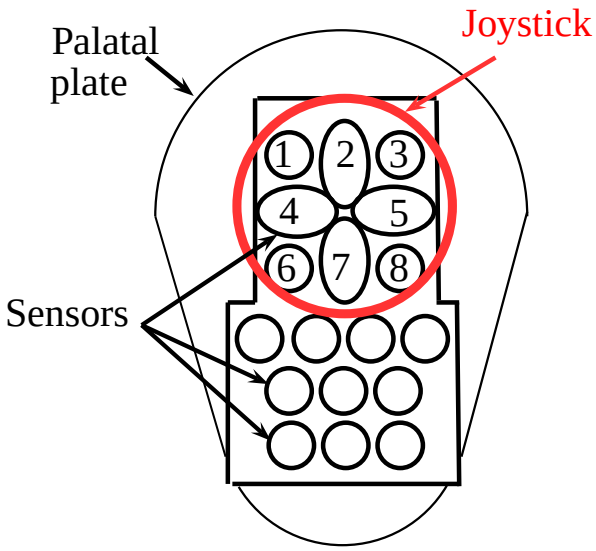


Figure 3.3: *The 8 sensors forming the joystick in the palatal plate.*

3.3.1 The inference system

The inference system is a software program which has as inputs the state of the 8 sensors which are creating the joystick, and has as outputs the coordinates x and y needed to the displacement of the computer mouse cursor. When the Activation Unit (AU) gets closer to a sensor, for instance the sensor 1 in Figure 3.3, the computer mouse cursor should operate a move to the up-left, or north-west, in the smoothest way possible.
Using a boolean estimation method would attribute a value only when the AU is close enough to the sensor, i.e. a threshold is set. The boolean method would only attribute 0 or 1 (true : activation, false : no activation) to any values from the sensors. It would lead to that the mouse cursor would jump from a position to the next one.

As mentioned in Chapter 1, the values of the state of each sensor are ranged between 0 and 255. The closer the value is to 255, the stronger is the activation. The fuzzy logic would attribute to all values an estimated value contained between 0 and 1. The fuzzy logic algorithm is based on the storage of the maximum value that can reach an activation (which is updated when a new frame is received). The difference between the value of the sensor without activation and the value of the sensor when there is an activation is compared with the maximum activation gap in order to obtain a number between 0 and 1. This number can be used as input of the fuzzy logic system.

Paragraph 3.3.2 explains the theory behind the fuzzy logic and paragraph 3.3.3 the application to the coordinates creation.

3.3.2 Fuzzy Logic

This paragraph is inspired by the documentation provided by the Mathworks website [32].

Dictionaries are defining fuzzy as an adjective synonym to vague, imprecise, not-clear, blur [33]. The $Fuzzy\ Logic$ (FL) is an evaluation methodology of input values to obtain an output, function of these inputs. This logic is closer to the human logic based on experiences where nothing is only 0 or 1 (true or false) but can have different degrees.
Where the boolean logic declares that for any value of inputs, the output is either 0, or 1, the FL interprets the input values to give an output (known as μ) ranged between 0 and 1. For instance, illustrated in Figure 3.4, considering the qualifier "'Tall"' for people, a boolean logic would just attribute $\mu = 1$ when height is above 1m75 and $\mu = 0$ when height is below. FL would attribute to height between 1m60 and 1m90 different degree of membership (μ).

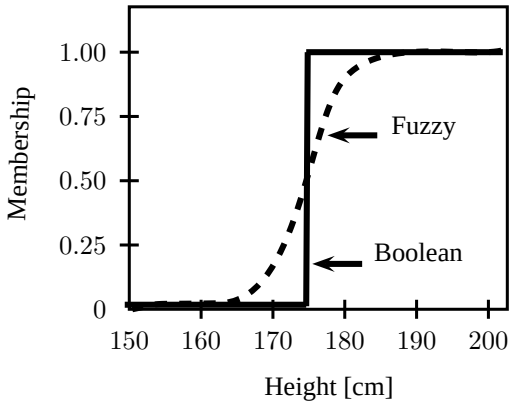


Figure 3.4: *" Tall " fuzzy variable. Modified from [32]*

Even if it is the possible to implement the "tall" fuzzy variable with conventional logic by defining "tall" as a function of heigh either as an analog or digital variable, fuzzy logic introduces a higher level of abstraction by providing more intuitive operators between fuzzy variables. These operators are the AND, OR, NOT logic and the Implication.

- AND is the minimum ($min()$) of the two fuzzy variables e.g. the degree a person is tall and small has its max of μ in between 1m40 and 1m90;
- Similarly OR is the maximum ($max()$) of the degrees of both variables;
- NOT is $1 - degree(\mu)$;
- IMPLICATION (IF a THEN b) (Equivalence is the AND of both directions implications) can be approximated in different ways e.g. Zadeh (creator of Fuzzy logic), Gödel, Mamdani and Takagi-Sugeno.

FL is well suited in case of inter-related analog variables that cannot be unambiguously modelized into a mathematical function. In this case the program will be more flexible, more easily maintained and rule-based programming is more user-friendly.
The software MATLAB handles the FL by following these 5 operations:

1. fuzzyfying the inputs : this operation interprets the value of each inputs and assigns an output ranged between 0 and 1;
2. applying the fuzzy operator : this operation applies, to the fuzzyfied inputs, the operator ($min()$, $max()$ or $1 - degree(\mu)$) which permits to defined the bounds the result is contained in;

3. applying implication method : this operation applies to the result of each operator an operator to truncat this result (generally $min()$)

4. aggregating all outputs : this operation is the combination ($max()$, probalistic OR or a simple sum) of the result of each operator appliance (+ implication method)

5. defuzzyfying : this operation gives the final result, a simple number function of the last aggregation. This function permits to get an average value of the aggregation result. Five methods are supported : centroid, bisector, middle of maximum (the average of the maximum value of the output set), largest of maximum, and smallest of maximum

Figure 3.5 shows an example of application of the FL by MATLAB. This example, based on the speed of a cyclist depending on the weather and wind, i.e. 2 inputs, permits to understand what each step of the FL is doing. It gives a unique output depending on the inputs.

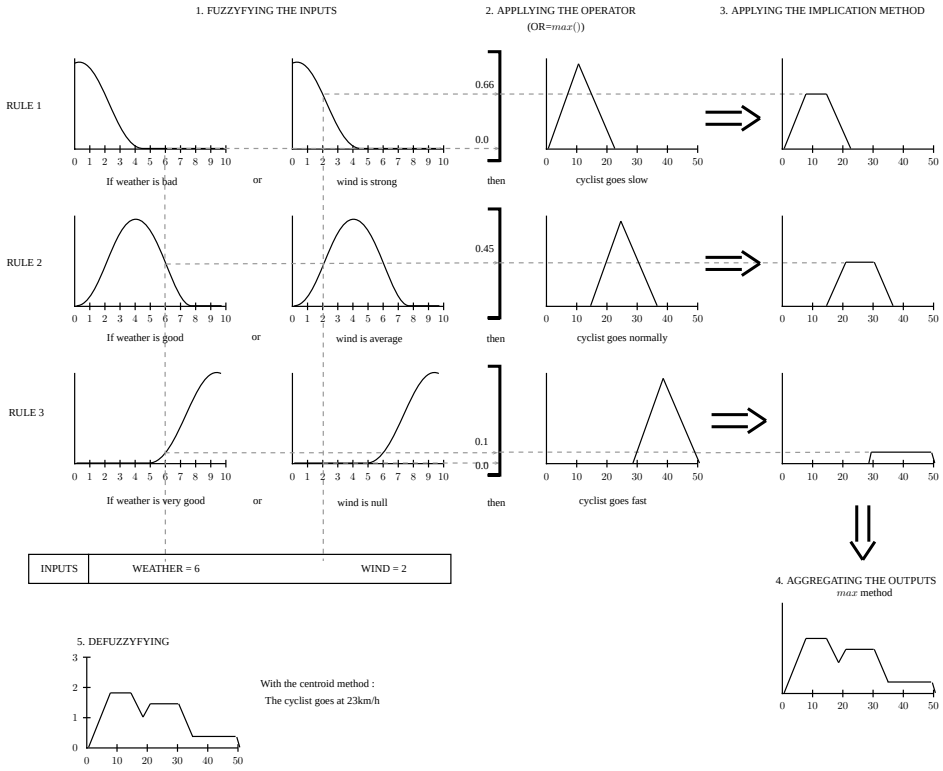


Figure 3.5: *This application example (speed of a cyclist) shows the principle of the FL. 1. Fuzzyfying the inputs, to get a number between 0 and 1. 2. Applying the fuzzy operator, here a OR (=$max()$) operator is applied. 3. Applying the implication method. 4. Aggregating all the outputs, here the method used is $max()$. And finally 5. Defuzzyfying to get the final result.*

3.3.3 Fuzzy logic in the project

The previous paragraph introduces the theory of FL which is now applied to the project to create the x and y coordinates regarding to the position of the AU on the palatal plate i.e. the values of the 8 joystick sensors. According to the value of the 8 sensors to express the activation of the sensors, and following the application example, the rules are:

- IF sensor 1 is not activated and IF sensor 2 is not activated and ... and IF sensor 8 is activated THEN AU is close to sensor 8

- IF sensor 1 is not activated and ... and IF sensor 7 is activated and IF sensor 8 is not activated THEN AU is close to sensor 7
  ⋮ ⋮

- IF sensor 1 is activated and IF sensor 2 is not activated and ... and IF sensor 8 is not activated THEN AU is close to sensor 1

According to the value of each sensor, and after being fuzzyfied, the output of the fuzzy logic gives an estimation of the position of the AU on the palatal plate i.e. to which of the sensor the AU is closer.

A MATLAB method is used on the 8 inputs to get several outputs defining the x and y coordinates. MATLAB provides several C-coded functions used by the HST developers to design the FL code applied to the first prototype.

However, the MATLAB C-coded functions need an array of 8 values between 0 and 1 to produce the coordinates. As said in the previous paragraph, the values received are in 8 bits format without sign (between 0 and 255). So before running the fuzzy logic algorithms provided by MATLAB, a function called *Fuzzy_getXY* was created by HST research team to detect the max activation gap and then to normalize all the activations with this gap. The function takes for inputs the raw values of the sensors in the mouse area and produces as output the values of the sensors between 0 and 1 depending on the degree of the activation.
A for loop is used to perform the same computation for the 8 sensors because the baseline or the degree of the activation can change from a sensor to another as the sensors are not exactly the same in the MU.
Then the current baseline is compared with the stored max baseline. If the current baseline is higher than the value of the stored baseline, then the baseline is updated. The same computation is carried out for the activation gap which is the difference between the baseline of a sensor without activation and the value of the sensor when an activation is produced.
In such a way when a small activation is produced, the baseline is not updated and the activation gap as well but the ratio between the maximum baseline minus the value of the sensor during the small activation and the maximum activation gap, gives a value between 0 and 1.
0 is produced when the current value of the sensor is equal to the maximum baseline ($\Rightarrow$ no activation at all) and 1 is produced when the activation is so strong that the difference between the baseline and the current activation can reach the max activation gap.
All values between 0 and 1 can be obtained depending on the degree of the activation. The output value for each sensor is given by the equation 3.1.

$$\text{output } Fuzzy\_getXY = 1 - (x - (B - MaxActGap)) / MaxActGap \tag{3.1}$$

where x is the current value for one sensor, B is the baseline and $MaxActGap$ is the maximum activation gap.
A threshold can be also defined to take only the values up to a certain range (0.3 in practice) in order to remove the noise component.

The two main problems with this algorithm are the following:

- The baseline stored can only increase and never decrease (the update of the baseline when the current baseline is lower than the stored baseline is impossible because with this algorithm no distinctions are made between a small activation and a baseline drift created by the change of the temperature for example).

- The sensors have to be activated at the beginning and if the max activation gap is wrong due to an unpredictable error, the system should be restarted again because the output will have no sense. For instance, if the max activation gap goes to 200 suddenly it will be almost impossible to obtain a value in output higher than 0.8 also when a strong activation is performed.

The running of the Kalman filter or any other high level solution to remove the noise and the baseline should be considered before applying this function.

3.4 Controlling the mouse cursor

The input values are coming from the sensors, the Kalman filter algorithm, developed by the group, is used to remove the noise and the baseline is performed before running the function created by HST research group (*Fuzzy_getXY*). Then, the fuzzy logic functions provided by MATLAB runs to obtain the coordinates. Finally those 2 coordinates are used in MATLAB functions to create a cursor on the screen as summarized in Figure 3.6

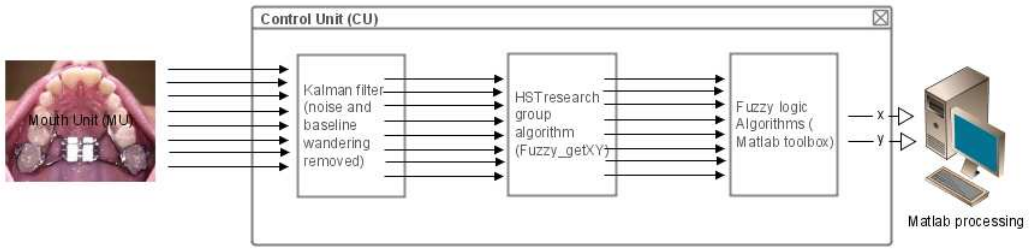


Figure 3.6: *Cursor command algorithm using 8 sensors in the MU*

3.5 Conclusion about the algorithm analysis

The research team from HST department created for the CU prototype the algorithms of communication and interfaces with the MU and the mouse cursor control. These algorithms have been implemented on the 8 bit microcontroller in C coded language. The algorithm for the digital radio receiving the data packets from the MU is simple and based on the Nordic Semiconductor nRF24L01 datasheet which introduce all the C functions needed to its good working. Once the state of each sensors have been received, the CU needs to process them to remove the perturbations as explained in Chapter 2. Then, the research team created an algorithm to detect the position of the AU on the 8 sensors of the palatal plate forming the joystick pad and create the coordinates x and y which will be used to control the computer mouse cursor. This algorithm is based on the $fuzzylogic$ (FL) which claims that for each value of the state of a sensor (which could be between 0 and 255) a unique output value is set (ranged between 0 and 1). The fuzzy logic is close to the human logic where any abstract concept is not only true (1) or false (0). Applied to the project, the fuzzy logic gives an estimation of the position of the AU on the palatal

plate. MATLAB provides C functions to create a fuzzy logic program to have these x and y coordinates. Once the coordinates have been established, the research team used other MATLAB tools to used these coordinates to control the mouse cursor.

Chapter 4

Platforms Analysis

This chapter is relative to the Architecture domain from the A^3 design methodology, shown in Figure 4.1, especially to the analysis and choice of different platform to execute the algorithms defined in Chapter 2. This analysis of different platforms must answer the third sub-goal defined Section 1.2:

> 3. Being as flexible and extendible as possible to permit the add of new external devices interfaces in the future.

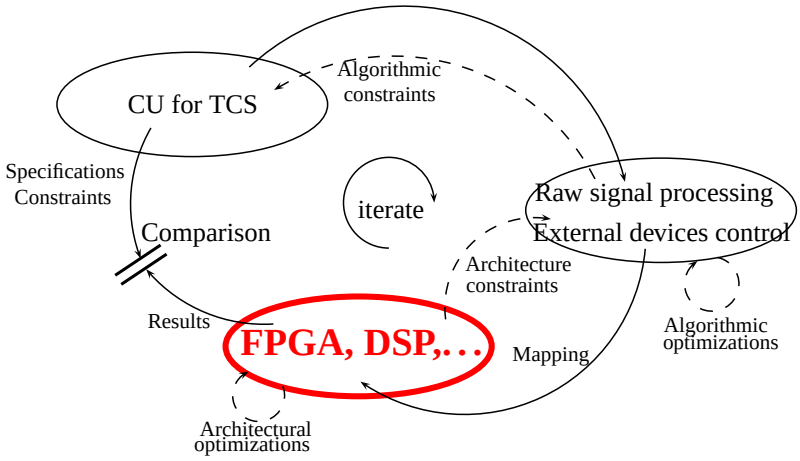


Figure 4.1: *The A^3 design methodology applied to Chapter 4*

4.1 Overview

Real-Time imaging and signal processing applications are becoming increasingly important. When implemented on hand-held devices they also impose constraints on physical size, power dissipation and price of the solution. Ease of reconfiguration of the solution is another key differentiator. The most used architectures fulfilling these constraints are Field Programmable Gate Arrays (FPGAs) and Digital Signal Processors (DSPs).

These two platforms have advantages and drawbacks for the implementation of the algorithms of the project. An introduction and analysis of these platforms is performed to define which platform suits the best the algorithms of the application, in Sections 4.3 and 4.4. Section 4.5 presents the main differences between each platforms and Section 4.6 defines the chosen platform.
But first and foremost, Section 4.2 reminds the constraints, defined in the previous Chapters that, the platform has to fulfil.

4.2 The constraints on the platform

Chapter 3 introduces the algorithms which have been used for the realisation of the 8 bit microcontroller prototype. The radio communication protocol is one of the prior algorithm to implement on the new platform. The MU sends to the CU the data over a 28 bytes packet by means of a digital radio transmitter. The radio on the platform have to be able to receive the data from the MU.

Figure 1.4 shows that the CU needs to gather each control interface for each external devices. This constraint leads to the need for an extendible platform which can be upgraded with new future external devices in the simplest way possible.

The algorithms, designed in Chapter 2, to reduce noise and the temperature drift, based on the Kalman filter, have to be implemented in the platform with a maximum efficiency, in terms of rapidity of execution of the algorithm. These computations require from the platform a quick and powerful computation unit to be a real-time application.

The CU is an external box, powered by a battery, close or far from the user, but it is an embedded system. Power consumption, dimensions and price of the platform are constraints to consider as well since they have to be as small as possible.

4.3 Introduction and Analysis of FPGAs

A FPGA is a Programmable Logic Device composed by programmable gates and programmable interconnections between the gates, hence any application from a simple logic function to FFT can be realised. Two constructors are sharing the FPGA market of general usage; Xilinx [34] and Altera [35]. Other manufacturers, such as Atmel [36] or Lattice Semiconductors [37] are dedicated to niche domains as transport, aerospatial or military. Xilinx created the first platform referred to as a FPGA in 1985.

Before presentation of the detailed architecture of FPGAs, their key design principles are highlighting.

Most FPGAs use Look-Up Tables (LUT) as their elementary blocks. LUTs allow the creation of any logical functions between logic variables because they firstly demultiplex the input variables combinations and then implement a one bit wide memory containing the value of the output. Then it is possible to interconnect basic logic functions to combine them into complex ones. In this way circuits design can be changed without modifying hardware.

The following Figures 4.2 and 4.3 give an overview example of the FPGAs principle.

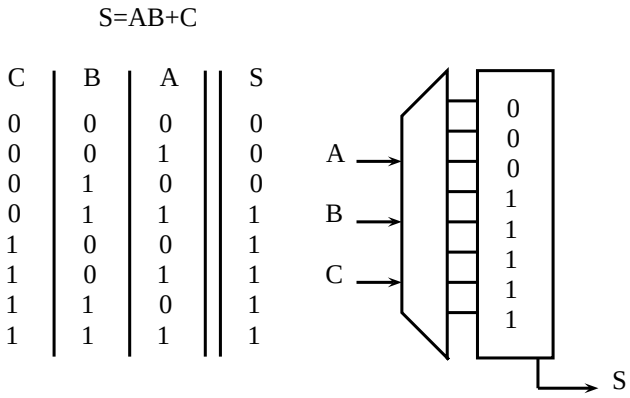


Figure 4.2: *Logic with Memory - LUT. Modified from [38]*

The AND and OR logic functions are implemented with two LUTs as shown in Figure 4.3 where the un-programmed LUTs are in grey (LUTs on the Figure) while those programmed are in yellow (AND and OR). The Figure is also an indication of the implementation of programmable connections that are controlled by memory, the "Memory Cell" controlling the gate of the MOS FET.

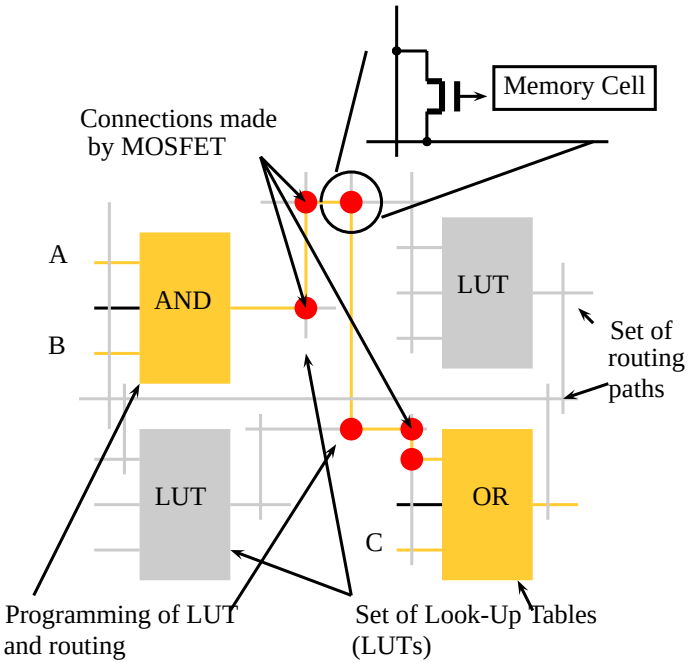


Figure 4.3: *FPGA cells architecture and Logic cells interconnections. Modified from [38]*

4.3.1 General architecture of FPGAs

The general architecture of FPGAs, illustrated in Figure 4.4, is composed of:

- Kernel components which are Circuit Logic Blocks or Logic Blocks(CLBs or LBs) that provide LUT function to achieve Logic function, programmable Multiplexers and two Flip-Flops for se-

quential logic;

- Input Output Blocks (IOBs) are interface modules. Interface variables can be stored in Flip-flop gates;
- Connection lines and connection matrices that allow programming of interconnections between CLBs and IOBs. Routing delay is dependant of physical layout of source and destination but is usually less than 10 ns

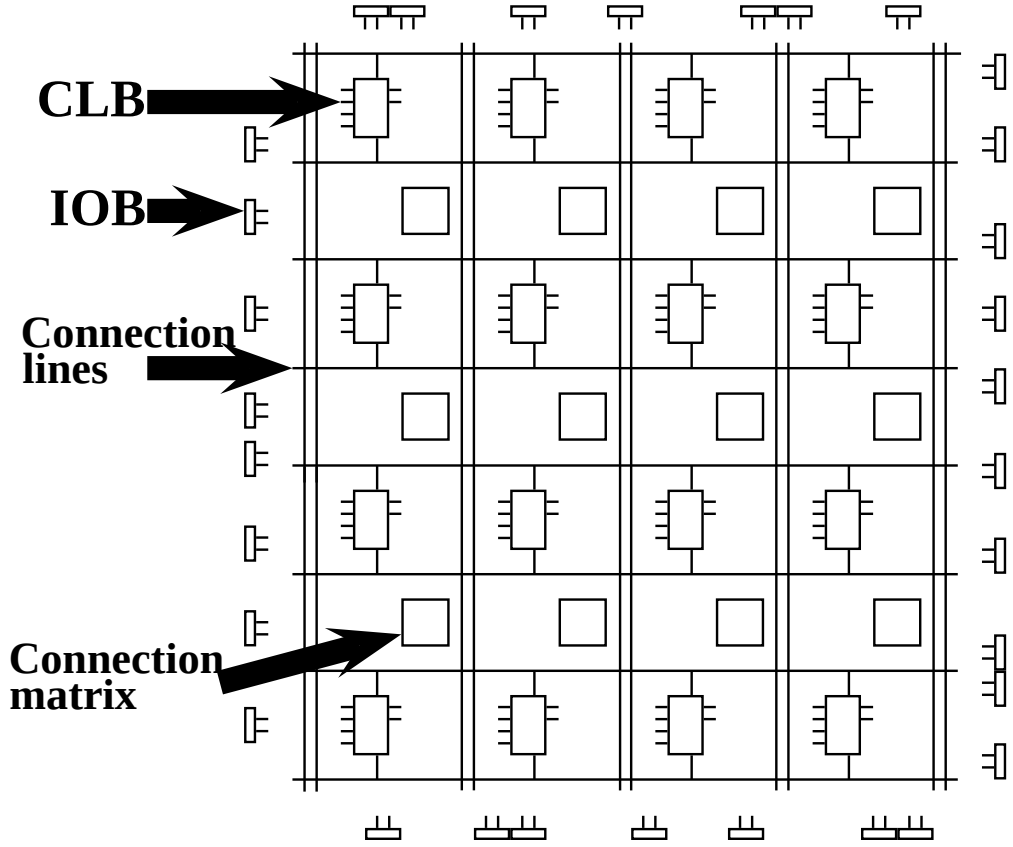


Figure 4.4: *General Architecture of FPGAs. Modified from [38]*

In fact, there are four FPGA categories as presented in Figure 4.5:

- Sea-of-gate, Figure 4.5(a): This type is similar to the first Xilinx FPGA. The main difference lies in the type of interconnection used which is implemented as an overlay of the whole LB.
- Symmetrical Array, Figure 4.5(b): This type uses 100 to 1000 LBs. Multiplexers (MUXs) are used to choose which LB and Interconnections are on use. The example presented in Figure 4.4 is from this category of FPGAs.
- Row-Based, Figure 4.5(c): This type too uses MUXs to choose which LB and Interconnections are on use.

- Hierarchical PLD, Figure 4.5(d): This type is used by Altera. With only 20 LB, it is creating an array of logic gates (20,000 logic gates) between the blocks.

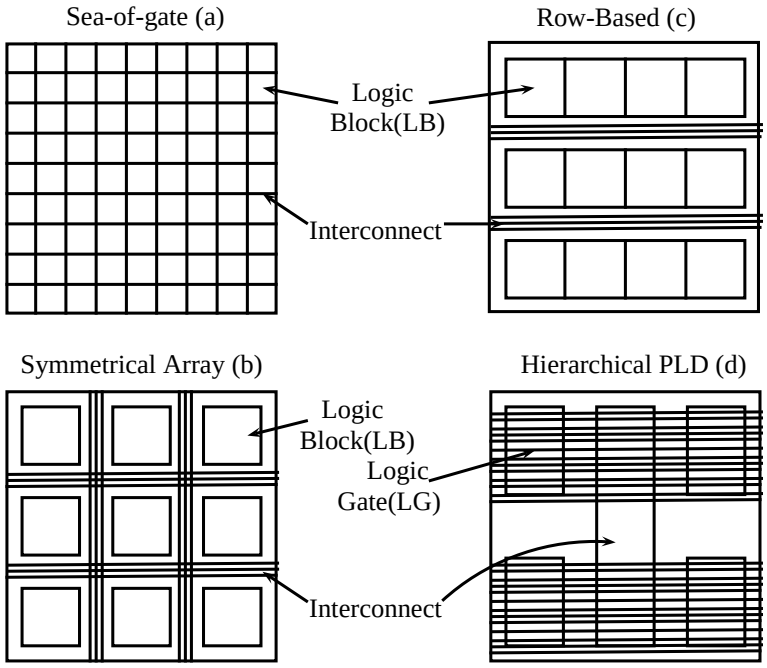


Figure 4.5: *The 4 different types of FPGAs. (a) is a representation of the Sea-of-gate where all the LB are interconnected to form the desired architecture. (b) represents the Symmetrical array, (c) the Row-based and finally (d) is a schematic of the Hierarchical PLD type.*

Each of these types are still used nowadays besides Xilinx's design are using the symmetrical array. Recent FPGAs can contain several hundreds of thousands CLBs and more than 40 multipliers.

4.3.2 Design and Programming on a FPGA

As mentioned in previous paragraph, FPGAs are programmable components working on the concept of Circuit Logic Blocks, Input/Output Blocks and Interconnections. All of them are programmable to create different more or less complex circuit. The process to create a circuit is called "'design development process'"' and consist of a six step sequence, as follow:

1. Design Specification;
2. Conversion of the specification into a logical consistent description suitable for entry into a CAD (Computer-Aided Design) system;
3. Compiling the entered design;
4. Programming the target device;
5. System commissioning and testing.

A CAD system permits to assist the designer in the conversion of the specification into a logical description. This system handles the compiling, synthesising and also simulating of the design. Xilinx or Altera provide such CAD systems to use at ease their FPGA board.

Once a design has been completed, it is time to program it. To program the board, different method can be used but the main ones are to use a special programmer which transfer the data from the PROM to the FPGA, or to download from a computer the design to the FPGA using an interface. In most case, the link between the computer and the FPGA is a simple cable. The computer connection is more and more used as the interface using, mostly, the JTAG protocol (Joint Test Action Group) is simple, the cable is connected to serial ports. However, the download of the program is needed each time the FPGA is turned off as the design is downloaded in the RAM of FPGA. But once the program reached its completion, it can be copied to the EEPROM for a permanent storage.

4.3.3 FPGA solution

By their architecture, FPGAs are high flexible platforms, in terms of Input/Output rate and with high performance by means of the possibility to parallelise the computations. Depending to the constraints on the platform defined in the project, FPGAs are a candidate solution for the questions of extendible platform for integrating interfaces to future external devices. Moreover, the latest FPGAs can provide an efficient computational power which can be used for the execution of the Kalman Filter operations.

The implementation on FPGAs are usually done in VHDL to create accurate and optimized design. However, Altera, or Xilinx, provides with their products some software to easily create the willing design. In the project case, illustrated in Figure 1.4, input/output blocks for radio are needed as well as for the USB interface.

4.4 Introduction and Analysis of DSPs

Digital Signal Processors (DSPs) are microprocessors designed for efficient execution of digital signal processing algorithms, often in real-time. DSPs are usually programmed in C, though optimizations sometimes require to use assembly language. Their architecture, as for example the DSP2199X illustrated in Figure 4.6, is well suited for fast execution of DSP algorithms:

- Fast Multipliers;
- Multiple Execution Units (parallelism);
- Efficient Memory Access;
- Direct handling of interruptions (with low overhead for real-time);
- Efficient Zero-Overhead Looping;
- Specialized Instruction Sets.

DSPs have a fixed architecture composed of a Control Process Unit (CPU) which use memory to store and read the instructions or the data. Other modules can be also present such as SPI, timer, ... The DSP is very efficient to implement complex algorithm and has higher clock frequencies than FPGAs. However, the data in a DSP are mostly processed sequentially (although some DSPs manage Very Long Instruction Word (VLIW), the degree of parallelism is far less than that of FPGAs) that is a serious drawback for computation with high parallelism such as finite loop computation. Moreover, the architecture of a DSP is fixed so the choice of the DSP is highly dependent on the application [41].

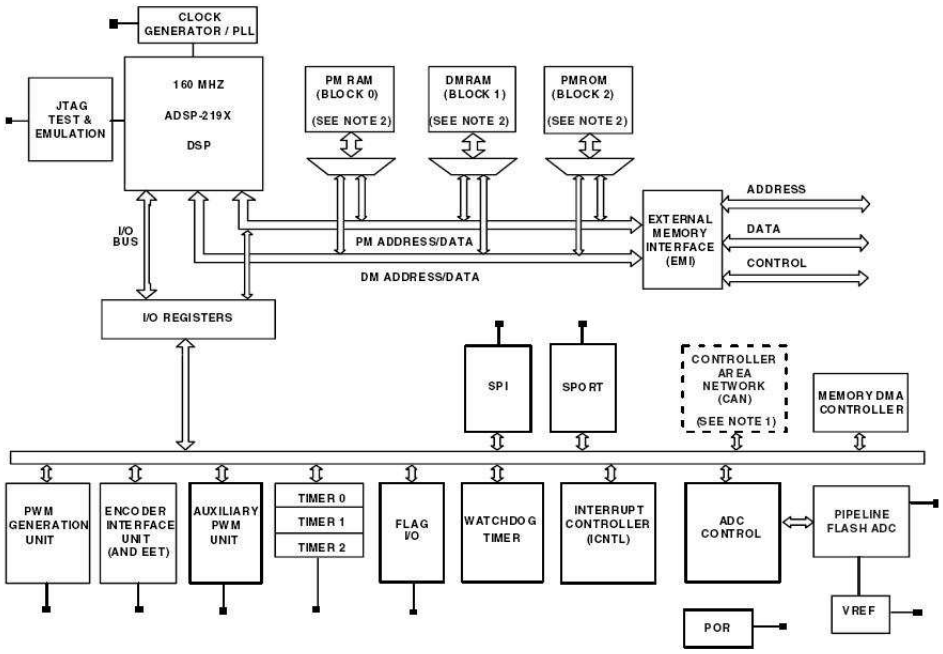


Figure 4.6: *General Hardware Architecture of a DSP2199X family. Modified from [39]*

DSPs are used for applications which need high accuracy such as floating point inversion, matrices division or matrices inversion. The development time is shorter than the development time on a FPGA and DSPs consume less power. They are well suited with real time applications. In fact DSPs lead the market of mobile phones, CD recorders, modems,...
The principle manufacturers of DSPs are Texas Instrument, Analog Devices and Motorola [41].

4.5 Comparison between FPGAs and DSPs

Table 4.1 summarizes the advantages and drawbacks of FPGAs and DSPs and lists some of their typical applications.

	Advantages	**Typical Applications**
FPGA	- Higher performance (very high parallelism) - High Input/Output rates	- Applications that can be split into basic logic modules e.g. Digital Image Processing such as MPEG4 decoder and encryption
DSP	- Less power consumption - Less expensive - Easier programmation (can be directly programmed in "C", conditional execution is simplified) - Re-use of modules is simplified	- Applications that can present high complexity and require high flexibility e.g. Mobile Phones with their multiple interfaces

Table 4.1: *Comparison table between FPGAs and DSPs*

4.6 Chosen platform

The restricted availability of the boards in laboratory obliged the group to choose between few different platforms. DSPs were not available, implementation of the solution on FPGAs was compulsory. Nevertheless, the chosen platform is the FPGA Altera DE2 with a NIOS II soft-core processor implemented on it. In such a way, the group had a platform combining the flexibility of software (NIOS II) and the computational power of hardware (FPGA part which is not used for the NIOS II).
This choice allows to directly program the FPGA with C code which is very attractive because C is higher level than Hardware programmation (algorithms with more complexity can be implemented in less time).

4.7 Conclusion about the platform analysis

The main goal of the project is to implement on a powerful platform, the existing algorithms of communication and interface with the MU and the external devices and also the algorithm of the Kalman filter processing the received data from the MU. A need of a high flexible platform to allow the easy addition of interfaces to different external devices combined with the need of a powerful computational unit for the execution of the Kalman filter require the analysis of the 2 main platforms; FPGAs and DSPs.
FPGAs are high flexible and extendible platforms but do not allow complex computations. The language used to program a design is the HDL which requires from the designer to think in hardware. FPGAs are more and more used but still cost a lot.
DSPs are recommended for high complexity computations but do not allow many input/output. They can be directly programmed in C language, are less expensive and the power consumption is less important than FPGAs.
However, FPGAs are now developed with a DSP softcore processor to allow more complex computation, as the chosen board for the project, the Altera DE2 FPGA.

Chapter 5

Implementation

This chapter is relative to both Algorithmic and Architecture domains from the A^3 design methodology, shown in Figure 5.1, especially to the mapping on the platform, chosen in Chapter 4, of the algorithms defined in Chapter 2 and Chapter 3. This Chapter corresponds to the main goal of the project:

Completing the Tongue Control System to a marketable, complete and accurate product.

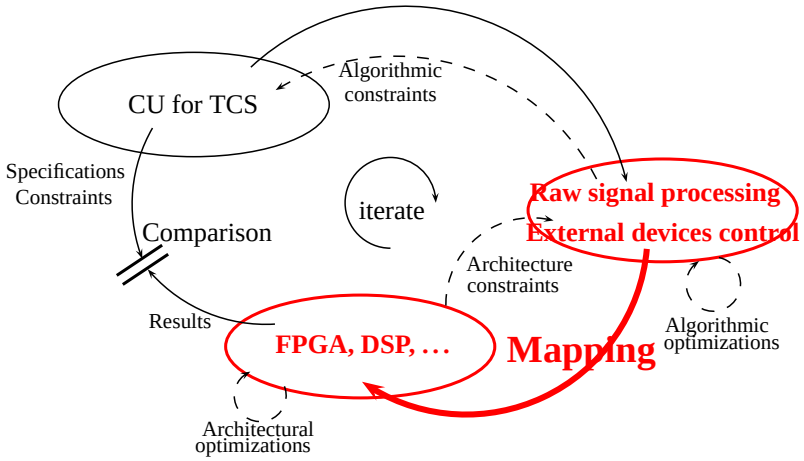


Figure 5.1: *The A^3 design methodology applied to Chapter 5*

5.1 Overview

As mentioned in Chapter 4, the board the group uses is the Altera DE2 with a microprocessor on it called Cyclone II. This FPGA board, which is introduced Section 5.2, is used by the project group to implement the algorithms detailed in Chapter 3 then, the Kalman Filter from Chapter 2. The way the group done these implementations and the difficulties met are explained Section 5.3.

5.2 Introduction to NIOS II

NIOS II core is a family of Software DSPs developed by Altera, one of the leaders in this market (other leader is Xilink). Software DSPs also called embedded processors, are implemented using FPGA Hardware i.e. registers, buses, peripheral controllers, ... are implemented using internal FPGA logic.

Embedded DSPs offer significant advantages over hardware DSPs:

- Can be customized e.g. specific I/O interface;
- More perene hardware solution (just update software);
- Remaining FPGA hardware can be used for Hardware acceleration.

But they also provide drawbacks:

- They are less cost effective when standard Hardware DSPs can do the same;
- More complex to design (additional complexity introduced i.e. Definition and Configuration of the software DSP according to the application).

Altera NIOS family gather 3 implementations (Economy, Standard and Fast) according to price and performance. Figure 5.2 is the overview of the architecture:

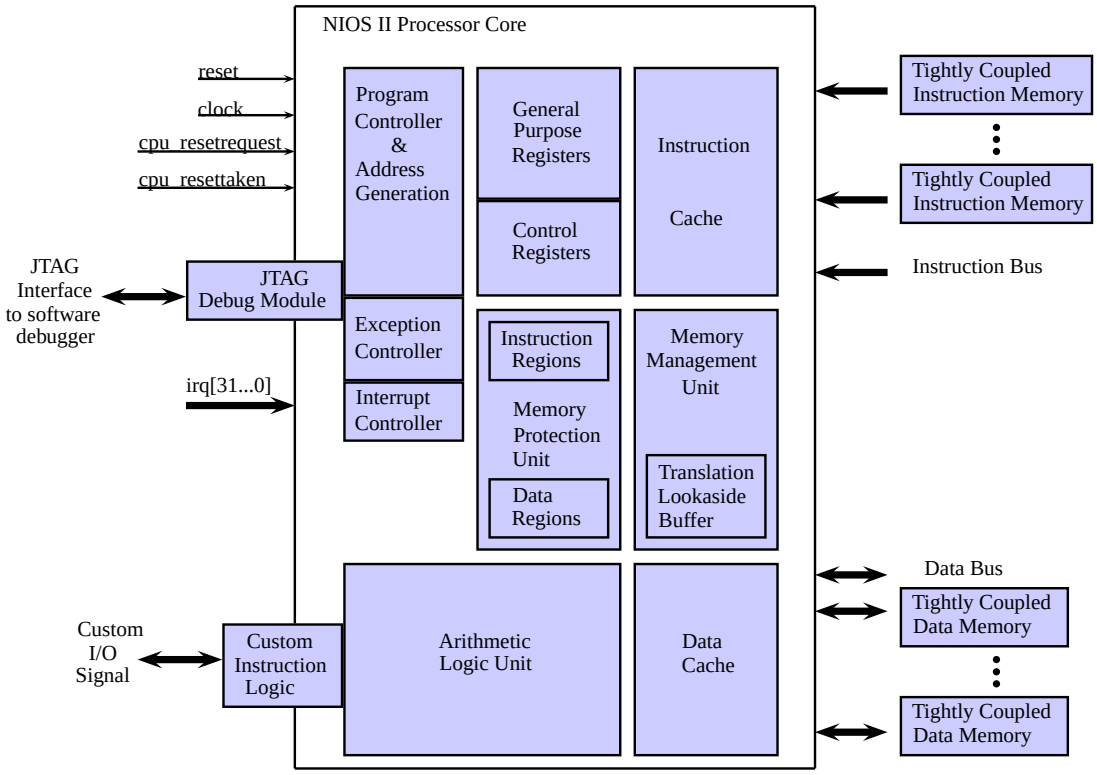


Figure 5.2: *Architecture of NIOS II DSP. Modified from [42]*

It contains two separate buses: one for Instructions and one for Data. It can handle of up to 32 Interrupt request levels and allows memory-mapped I/O interfaces to external devices. An optional Memory

Protection Unit would allow the control of Data and Instructions addresses of up to 32 different areas. The JTAG (Joint Test Action Group from IEEE) is a software based debugger.

Development on Software NIOS DSPs is similar to the one on any other Hardware DSP. NIOS development suite includes the Embedded Design Suite (EDS) which provides a Board Support Library or Package (BSP) implementing the Application Programming Interface (API) between the hardware on board and the controlling program. The executable file generated by NIOS has a ".elf" suffix.

IP Megastore is a library of standard functions already implemented in Altera Hardware such as Speech, Audio and Video processing, Fast Fourrier Transform (FFT) and modulation/demodulation with(out) encryption/decryption.

NIOS environment is made of:

- Executable code and data files;
- Specific code used to load Flash memory into RAM, and then execute program from RAM;
- Hardware Abstraction Layer (HAL) file subsystems described here after;
- FPGA hardware configuration data.

5.2.1 Introduction to SOPC Builder

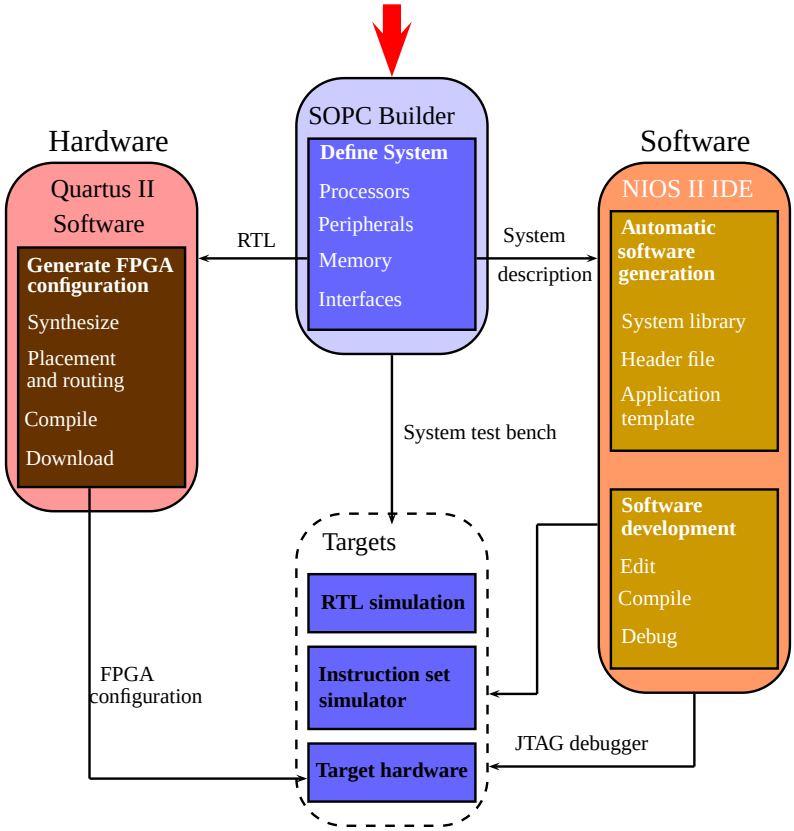


Figure 5.3: *Altera Embedded processors family. Modified from [43]*

SOPC Builder allows the configuration of the Software DSP i.e. to define its architecture in terms of definition of components and their interfaces:

- On the left hand side of the previous figure is the Altera FPGA hardware configuration tool i.e. Quartus II software.
- On the right hand side of the previous figure is the development of the Software to be run by the embedded DSP to achieve the desired application.

SOPC combines the Hardware description Language (HDL) of the components of the system. The goal of the HDL is to describe the transfer function of the component, its structure and simulation for testing purposes. The main component in HDL is time, which is missing in conventional programming. Main HDL implementations are Verilog, VHDL and SystemC.

SOPC Builder interfaces with Quartus II software in order to allow compilation of FPGA configuration commands and then downloads code into FPGAs hardware components.

The SOPC Builder component editor allows the creation and editing of files describing the behaviour of each component. They are named "Hardware Component Description File (_hw.tcl) files". Each "*.tcl" file describes a component of the system. These kernel files include:

- Specification of the Verilog HDL or VHDL files that describe the component and most importantly its interfaces signals and their behaviour;
- Definition of eventual relationship(s) between interfaces;
- Declaration of the parameters of the components e.g. Clock speed for an oscillator;
- Optionally Component Driver files for testing purposes.

SOPC uses the "Interconnect fabric" to connect components and then build the architecture of the system. The system Interconnect Fabric defines the connections between the various components of the system. It also provides additional functionalities like:

- Multiplexing of Data buses;
- Decoding of address ranges;
- Arbitration on Multiple Masters configurations;
- Handling of Interrupts;
- Pipelining of requests.

There are two types of "Interconnect Fabric (IF)": either Streaming (ST) or Memory Mapped (MM). ST Interconnect fabric applies to only one source/destination pair while MM allows multiple sources and destinations. In opposition ST-IF typically provides higher throughputs compared to MM Interconnections.

SOPC Builder allows the definition of the architecture of the system and will generate the FPGA configuration data. It also provides information to NIOS II for its recompilation in order to adapt to new hardware configuration.

5.2.2 Architecture used in this project

The standard architecture of the FPGA Altera DE2 were not working when the group tried to implement easy code in order to get used with the board. The tutorials were not working probably because of a an older version of Quartus II. The group decided to create its own Hardware architecture with SOPC Builder, motivated by the 3 main reasons:

- A better understanding of the working of the board will be performed.
- The addition of new modules depends on the future goals requirements.
- Minimal resources are added in order to implement only the solution.

The Figure 5.4 shows the hardware existing on the board (added with SOPC Builder)

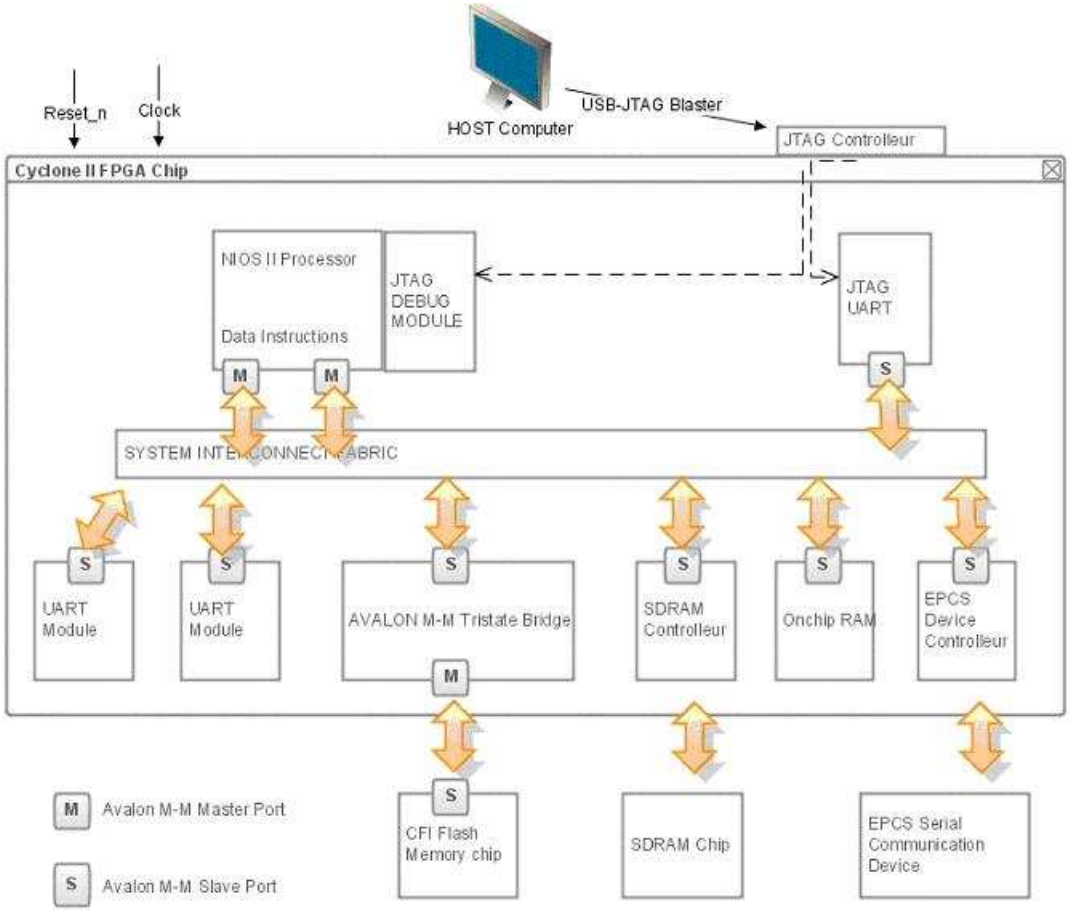


Figure 5.4: *Hardware existing on the board (added with SOPC Builder)*

First of all, the addition of the NIOS II Processor is needed with the JTAG UART connection to program and debug it. The System Interconnect Fabric is created and updated by SOPC Builder each time a new component is added. Onchip memory such as RAM is required to load and store instructions and data from the processor. The UART Module implements an interface to communicate serial data between the FPGA and another external system. The UART module is using Recommended Standard 232 (RS-232) protocol. The RS-232 protocol can transmit synchronously or asynchronously with a constant baud rate (that is the number of symbols/s). To minimize the transmission loss, in this project, the baud rate is set to 38400, there are two stop bits to separate the frames sent and the data of each frame are coded on 8 bits. Moreover to simplify the transmission between the different architectures, only the signals TxD (Transmit Data) and Rxd (Receive Data) are used. However, in the RS-232 standard the voltage levels that

correspond to logical 1 and logical 0 levels are between 3V and 15V or between -3V and -15V. On the board Altera DE2 there are only one connector RS-232 and only one module to shift the voltage level.

The project group decided to communicate with an external input using RS-232 protocol and to communicate with an external output using also an RS-232 protocol. Although the RS-232 protocol is limited by important power consumption, reduced noise immunity and limited transmission distances [44]. The group project decided to choose this transmission protocol because of its simplicity and the time gain. However in the final solution other communication interfaces have to be integrated. For instance the Serial Peripheral Interface Bus (SPI) should be preferred because it will permit to directly transmit data from the MU to the FPGA without passing by the microcontroller as explained in the following section. The second level shifter needed by the second UART module has been built by the group and the second RS-232 connector was also created by the group, using the General Purpose Input/Output (GPIO). The GPIO is an interface which can communicate with external devices (reading digital data as input or sending digital data as output).

In order to implement directly C code on the FPGA using the HAL libraries created by SOPC Builder as explained in the following part, a huge space memory is needed. The addition of an SDRAM module is required and no code C can be directly implemented with only the Onchip memory. The group project chooses to add a Synchronous Dynamic Random Access Memory (SDRAM). SDRAM can only transfer one world data per clock cycle. In general the clock cycle is around 133MHz (which is upper than the Altera DE2 maximum clock frequency of 50MHz). Moreover, the data bits are well protected each one is separate with security capacitors and writing pipelining is also possible [45].

SDRAM has a synchronous interface which means that a very precise clock is needed as input. The Phase Locked Loop (PLL) Module is needed to synchronize the SDRAM. PLLs circuits apply a negative feedback to a local oscillator, lowering or raising the frequency of it, until the phase and the frequency perfectly match with the desired signal [46]. In order to safety add the SDRAM controller in the SOPC Builder and to add the PLL, the Altera tutorial should be followed [47].

The SDRAM is a volatile memory in opposition to the flash memory. Flash memory is very present in portable applications and allows faster access time than the SDRAM. Moreover, the flash memory can face high temperature or even high pressure [48]. NIOS II IDE provides a flash memory programmer which allows to store the hardware architecture used for the FPGA on it, the code that is running on the processor or program code that the user wants to store. Addition of the flash memory in the FPGA Hardware is motivated by the real environment simulation where the CU will continuously run. Addition of the flash memory controller cannot be done without adding an Avalon MM Tristate bridge. An Avalon Bridge modifies the way the data are transported. A bridge is composed by an Avalon Memory Mapped (MM) Master and and Avalon Memory Mapped (MM) Slave. It is used to increase the transfer performances between the NIOS II Processor and the Flash Memory. Avalon MM Tristate Bridge is used to connect the NIOS II Processor with external chip components like the flash memory. The Avalon MM Tristate Bridge has to share data, instructions, reading and writing requests between the Processor and the Flash Memory. It is strongly advised to follow the Altera tutorial, which can be found in [50].

The Erasable Programmable Configuration Serial (EPCS) are the cheapest configuration devices in the programmable logic gate industry [49]. EPCS have the capability of In-System Programmability (ISP) and are also very used to access flash memory for large memory requirements. In this project, they are used because they can store the Altera DE2 configuration but also executable code for NIOS II. Once again to add the flash memory and the EPCS it is strongly advised to follow the Altera tutorial [50].

SOPC Builder can directly affects Base Addresses for all the components and also manage the Interrupt Requests. The automatic affectation of Base Addresses and Interrupt Requests is also strongly recommended

The table 5.1 regroups the modules used for this project:

Component Name	Main Informations
NIOS II Processor	Standard RISC Architecture Data and Instructions coded on 32 bits
SDRAM	Memory Size: 4194304 $\times$ 16 = 64Mbits
Flash Memory	Memory Size: 4 Mbits Timing: - Setup: 40ns - Wait: 160ns - Hold: 40ns
UART1	-Baud Rate: 38400 -Stop Bits: 2 -Data Bits: 8 -No parity
UART2	-Baud Rate: 38400 -Stop Bits: 2 -Data Bits: 8 -No parity

Table 5.1: *Modules used for the project*

5.2.3 Instantiation, pins assignment, compilation and hardware programming

Once the SOPC Builder system is created, a HDL entity has to be created to instantiate the SOPC Builder System. Instantiation is a very important part and permits to create the input and output signals for the SOPC Builder System (created on the Cyclone II chip), but also to add components necessary to the good working of this system.

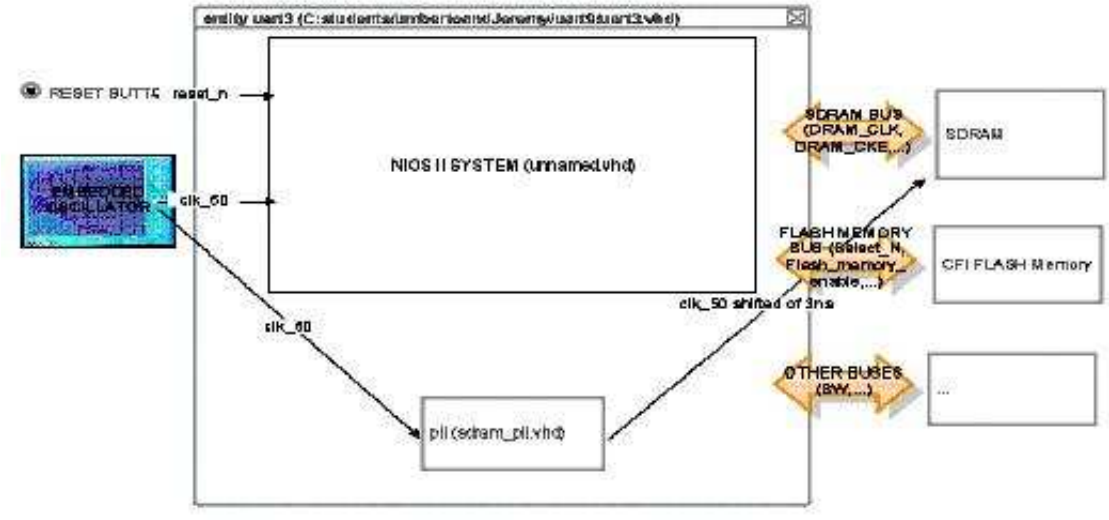


Figure 5.5: *Instantiation of the SOPC Builder system*

The group project uses VHDL language (because already studied) as HDL to define the different entities (the HDL language in the project as to be the same as the HDL used to describe the SOPC Builder

System). The NIOS II system (created with SOPC Builder) needs the addition of a PLL as explained in the previous section. The entity uart3.vhd has been created by the group.

The Pins assignment is the next step before compiling the system. It is needed to physically connect the pins between the Cyclone II Chip and the other modules in the FPGA. Pin Assignment Editor (PAE) in Quartus II permits to link the pins with the name of the top entity (here uart3.vhd). If the name of the pins are the same as the name of the input and output signals of the the top entity, PAE directly connects the Pins with the signals. Otherwise, manual assignment has to be done but it is a very long and complex work.

Then, the compilation of the Quartus II project should be done. The compilation can be split into 4 main steps [51]:

- Analysis and Synthesis: during this step the HDL is compiled and performs technology mapping using the resources of the selected board.
- Fitter: the fitter tool selects optimized interconnection paths, assigns logical cells on the FPGA in relation with the project and performs the Pin assignment.
- Assembler: the assembler tool creates a programming image for the device.
- Timing Analysis: if some timing constraints are defined by the user, the timing analysis will try to optimize the logic in order to respect them. In this project, no timing constraints were imposed.

The last step is the programming of the device with the tool Programmer in Quartus II. The Programmable Object File (.pof) or SRAM Object File (.sof) can be directly downloaded into the FPGA.

The table 5.2 shows the resources occupation of the created hardware on the FPGA. The memory occupation is almost complete (because a lot of memory is allocated to implement C code on the NIOS II), the others resources are not really used in this project, that means than other functionalities can be added easily.

Flow Status	**Successful - Mon May 18 19:09:12 2009**
Quartus II Version	**7.2 Build 151 09/26/2007 SJ Full Version**
Revision Name	**uart3**
Top-level Entity Name	**uart3**
Family	**Cyclone II**
Device	**EP2C35F672C6**
Timing Models	**Final**
Met Timing Requirements	**Yes**
Total Logic Elements	**4,419/33,216 (13%)**
Total combinational functions	**4,149/33,216 (12%)**
Dedicated logic registers	**2,314/33,216 (7%)**
Total registers	**2415**
Total pins	**97/475 (20%)**
Total virtual pins	**0**
Total memory bits	**313,216/483,840 (65%)**
Embedded Multipliers 9-bit elements	**4/70 (6%)**
Total PLLs	**1/4 (25%)**

Table 5.2: *Compilation report*

5.3 Implementation of the existing algorithms on the Altera DE2 board

5.3.1 NIOS II IDE and the HAL libraries

The Hardware Abstraction Layer (HAL) provides a simple device driver interface for programs to communicate with various I/O devices. It allow access to devices using simple C commands like $printf()$, $fopen()$, $fwrite()$, ... After SOPC Builder has generated a hardware system, the Nios IDE is able to generate a custom HAL BSP matching the hardware configuration. Changes in the hardware configuration automatically trigger the HAL device driver (re)configuration.

Figure 5.6 presents the HAL environment:

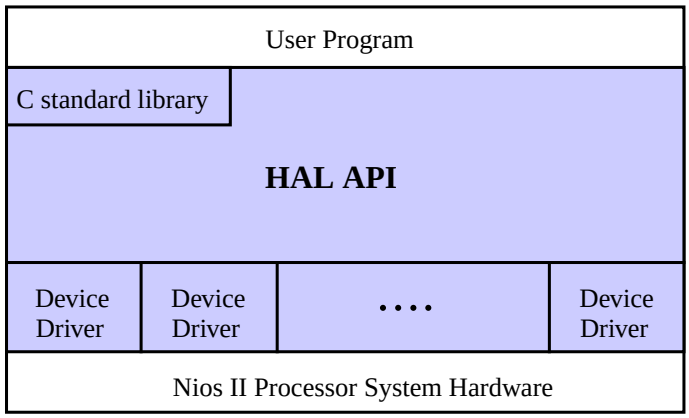


Figure 5.6: *HAL environment. Modified from [52]*

HAL API can be either directly accessed by the user program or can be accessed through standard I/O primitives of the C library. The C standard library used by HAL is called "Newlib" and must be added to the main program.

The NIOS II Integrated Development Environment (IDE) is a software used to program and run code with the NIOS II Processor. Moreover it is composed by a C/C++ compiler, debug tools, and C2H acceleration compiler. Each NIOS program is split into two projects:

- The application project,
- The HAL BSP project that will define the I/O drivers used by the application. It depends on the SOPC Builder information file.

The "system.h" defines the hardware and an interface is provided for I/O primitives such as $read()$, $write()$, $fread()$, $fwrite()$, $printf()$ and $fopen()$.

5.3.2 TCS prototype and project group solution implemented on the board

The TCS prototype built by the HST research team is composed by a MU which transmits data packets to the CU. In the CU, there is a module radio linked with a microcontroller. In the microcontroller all the computations are processed before sending back the data to a PC Unit where data can be verified. The microcontroller should store all the programs to command the mouse, the keyboard, the wheelchair, ... The main advantages of the use of a microcontroller are that C code can be implemented and that

microcontrollers are very cheap. However, complex algorithms cannot be implemented because of the reduced space of memory and the limited computations capacities. The more the project progresses and the more algorithms are developed which means that the choice of a new architecture is required.

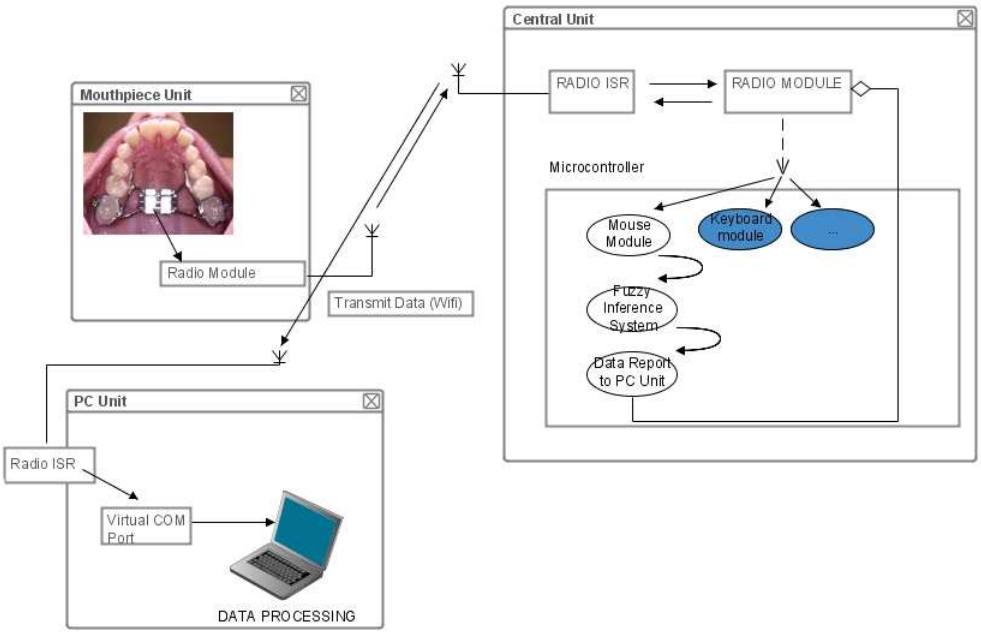


Figure 5.7: *The TCS prototype developed by HST*

Based on the previous discussion about the different kind of architectures and the University availabilities, the group project is using a FPGA with microprocessor (Altera DE2 board). However, the connections between the different entity with the new architecture are different. Indeed no radio module is present on the FPGA, so the microcontroller and the radio module linked are kept. This choice was made for commodity, and because the price of the microcontroller is negligible. In the final solution, the communication between the MU and the CU should be carefully studied in terms of reliability and price. The communication between the microcontroller and the FPGA is done with the RS-232 protocol. Because the solution running on the FPGA has to be tested, the communication between the board and the PC Unit needs to be created. Once again the project group decides to use the RS-232 protocol. However, as explained in the previous section, only one physical RS-232 connector and only one level shifter module (needed to reach the voltage norms) is present on the Altera DE2 board. The group project handles it with the use of the GPIO and constructing itself the level shifter with the board circuit documentation. This solution is yet the easiest one to get the data out from the board. In fact, for real time application few modules can be considered such as USB, SPI and UART (RS 232). SPI can directly handle communication between the radio and the FPGA (without passing by the microcontroller) but it is more complicated to use than the UART. The USB solution was not retained because the driver interface is not provided with the board.

In the PC Unit, the data arrive by the serial COM Port and then they are processed with different MATLAB algorithms depending on the solution tested.

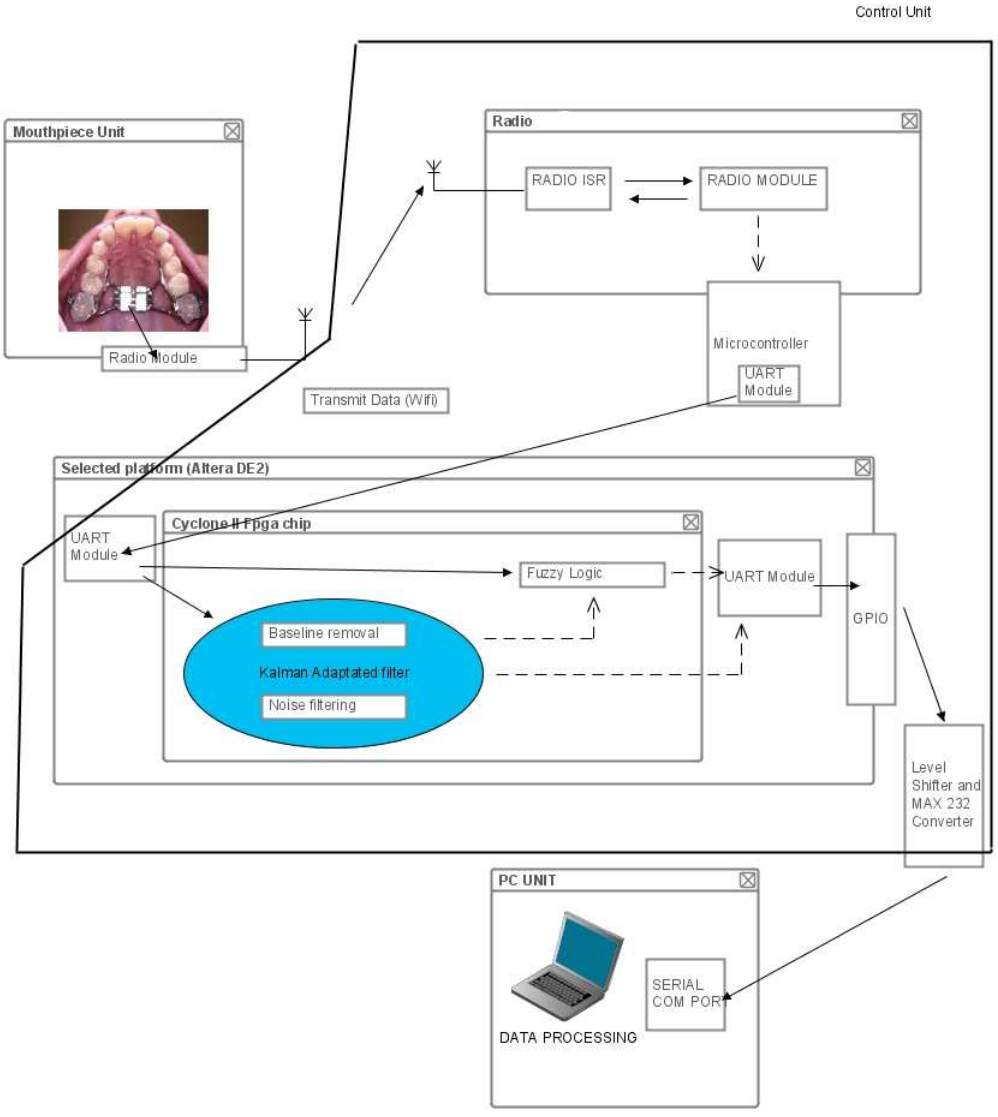


Figure 5.8: *The system put forward by the project group*

5.3.3 Fuzzy Logic implementation and Kalman filter implementation

5.3.3.1 Fuzzy Logic Implementation

The fuzzy logic toolbox in MATLAB provides already C function. The HST research team simplifies the C function provided by MATLAB in order to put it in the microcontroller. The $Fuzzy\_getXY$ function has been also already programmed in C. The group changes some standard used by the microcontroller in order to put these functions in the FPGA. The modifications were minimal.

5.3.3.2 Translation from MATLAB to C of the Kalman filter

The Kalman filter requires high power computation and the solution was totally defined by the group, so no MATLAB predefined functions were used. The main advantage is that the translation from MATLAB into C rests easier. On either side the Kalman filter algorithm is using a lot of matrices (that can be at maximum with 48 rows and 48 columns). Moreover any variables are using 30 previous values of the input.

However, the definition of the matrices, their initialisation is more complex in C. In fact, a matrix in C can be defined into two different ways: a double pointer or a double array, but in all cases the plot of a single value needs two loops, the first one covers the rows and the second one the columns.
More complex are the mathematical operations on matrices. In MATLAB, the operators can work (for an important majority of them) on scalars as on arrays or matrices. It is obviously not the same for the C language. The creation of all operators has to be done before performing any operation with matrices in C. The two difficult ones are the multiplication and the inversion of the matrices. The multiplication is done by the following formula: Suppose that A is a real matrix with m rows and n columns, the multiplication is only possible if B has n rows.

$$A \in \Re^{m\times n}, B \in \Re^{n\times p} \text{ so } (AB) \in \Re^{m\times p} \tag{5.1}$$

where the elements of $A.B$ are given by:

$$\forall i,j : c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} \tag{5.2}$$

Where cij is the element of the $A \times B$ matrix at the row i and column j, aij is the element of the A matrix at the row i and column j, bij is the element of the B matrix at the row i and column j.

From [53]

To compute one element, n products and $n-1$ addition are needed, which will be very slow for the multiplications of several matrices of 48 rows and 48 columns.

The inversion of matrices is the tricky part. Only squares matrices with non null determinant are consider so they are invertible.

The 2 main matrices inversion methods are:

- Gaussian elimination: This method uses the computation of the solutions of a system created by the following equations: Suppose that A is the matrix to invert and can be inverted. In this example A is a square matrix with 2 rows and 2 columns:

$$X = \begin{pmatrix} x1 \\ x2 \end{pmatrix}, Y = \begin{pmatrix} y1 \\ y2 \end{pmatrix} \text{ and } A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \tag{5.3}$$

$$AX = Y \rightarrow X = A^{-1}Y \tag{5.4}$$

The resolution of the equation $AX = Y$ permits to find the coefficient of A^{-1} as shown in the following computation:

$$A \times X = \binom{ax1+bx2}{cx1+dx2} = \binom{y1}{y2} \Rightarrow \binom{x1}{x2} = \frac{1}{ad-bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}\binom{y1}{y2} \tag{5.5}$$

- Cramer solution: Suppose that A is the matrix to invert, A^{-1} is given by the following formula:

$$A^{-1} = \frac{1}{|A|}(C_{ij})^T = \frac{1}{|A|}(C_{ij}) = \frac{1}{|A|}\begin{pmatrix} C_{11} & C_{21} & \cdots & C_{n1} \\ C_{12} & C_{22} & \cdots & C_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ C_{1n} & C_{2n} & \cdots & C_{nn} \end{pmatrix} \tag{5.6}$$

From [54].

where $|A|$ is the determinant of A, Cij is the matrix cofactor, and A^T represents the matrix transpose. The cofactor is the determinant obtained by deleting the row and column of a given element of a matrix or determinant. The cofactor is preceded by a $+$ or $-$ sign depending whether the element is in a even or odd position.

The computation for the determinant for a 2×2 matrix is very easy and implies 2 multiplication and one subtraction. For squares matrices $n \times n$, the determinant can be computed by the formula called "determinant expansion by minor" summarized in equation 5.7:

$$|A| = \sum (-1)^{i+j} a_{ij} M_{ij} \tag{5.7}$$

The coefficient aij represents the component of the matrix A at the row i and column j, Mij is the matrix obtained crossing the row i and the column j of the matrix A. Many algorithms of computation of the determinant are recursive and compute only Mij when Mij is composed by only 2 rows and 2 columns The Gaussian elimination cannot be implemented in C because the algorithm to find the solutions of a system are very complex and take time. The second solution is usually use by many matrix inversion algorithms. However, for a general square matrix A with n rows and n columns, the number of computations is very high. The number of computations of a matrix Mij is :

$$\left((n-1)^2\right)! \times (\text{2 products and one subtraction}) \tag{5.8}$$

The 2 products and one subtraction are needed to compute the determinant of a 2×2 matrix, then for a matrix of size 3×3 there are 3^2 Mij matrices of size 2, so the number of computation needed is:

$$3^2 \times (\text{2 products and one subtraction}) \tag{5.9}$$

For a 4×4 matrix, there are 4^2 Mij matrices of size 3, so the number of computations is:

$$4^2\left(3^2 \times (\text{2 products and one subtraction})\right) \tag{5.10}$$

The number of computation for a matrix of size $n-1$ is

$$\left((n-1)^2\right)! \times (\text{2 products and one subtraction}) \tag{5.11}$$

In the formula 5.13, the number of computations for finding the inverse matrix of a square matrix of size n is:

$$\begin{aligned}&(n+1)\ \text{additions} + n\ \text{multiplications} + \left((n-1)^2\right)! \times (\text{2 products and one subtraction}) \ldots \\ &\ldots + n^2 \times \left((n-1)^2\right)! \times (\text{2 products and one subtraction})\end{aligned} \tag{5.12}$$

The $(n+1)$ additions $+ n$ multiplications $+ \left((n-1)^2\right)! \times (\text{2 products and one subtraction})$ are used to compute the determinant and the other computations are used for the n^2 cofactors. This number is a max number of computations and has no sense practically because many algorithms are finding the rows or the columns with the largest number of 0 before computing the determinant or the cofactors. However, matrix inversion requires a lot of computations and is problematic in our project because the matrices used for the Kalman filter have 48 rows and 48 columns. The group project decided to modify the Kalman filter. In the first solution, the same Kalman filter was applied at all the 24 sensors. In order to create a C program that can run on the computer and also on the FPGA, the group decides to create 24 Kalman filters in parallel. The MU is composed of 24 sensors that should be corrected by the Kalman filter. 24 separates sensors are possible so, and that permits to manipulate matrices at maximum of size 2×2. Invert matrices will be easily found. The C code was implemented with success using a library of matrix operators function (joined in appendix) found on internet and using 24 Kalman filters in parallel instead of only one.

5.3.3.3 Implementation of the Kalman filter on the FPGA

The C code created for the FPGA cannot be directly implemented on the FPGA. All the functions or the instructions cannot be compiled by the NIOS II IDE compiler. For example instructions for the pre-processor in C are not understood by the NIOS II IDE because there is no pre-processor in the Altera DE2. The real matter is that the frequency of the FPGA is limited to 50MHz than is really less than the frequency of a modern computer. Furthermore the maximum space of the FPGA is 4 MBits with the previous architecture (the solution is run with the SDRAM). In the C code the matrices are using integer format for the numbers. An integer is composed by 32 bits in this architecture. So the maximum number of matrix cell that can be saved is:

$$\frac{4194304}{32} = 131072 \tag{5.13}$$

Considering that with the TCS prototype, almost all the matrices were composed by 48 rows and 48 columns, which means $48 * 8 = 2304$ elements, no more than $\frac{131072}{2304} = 56$ matrices can be stored. Less than 56 matrices where present in the TCS prototype but for complex matrices operations the definition of other matrices were done in order to facilitate the work of the Arithmetical and Logical Unit (ALU), so to gain time during the computations. Although that the group project meets difficulties with the memory space, in the solution with the 24 small Kalman filter, the problem is fixed. The mean matter was the computation time. The first paragraph will discuss about the different ways to find the time needed by the algorithm. In order to reduce the time computation , the group made few modifications:

- All the call to external functions were deleted, because the Program Counter (PC) needs to move to a specific address, saves the current address (it is the address to return in the calling function after the call) in the stack memory and then return to the calling function.
- The group project deletes successive loops, and gather the code in more complex loops
- The Kalman filter algorithm needs to manipulate real numbers. However, the use of float format gives faster performances than the use of double format. In fact, the floats are coded with 32 bits in this architecture while the double are coded with 64 bits.
- The complex matrices computations are modified to scalar equations. For instance, the equation 5.14 was modified:

$$K = Pmin \times H^T \times \left(\left(H \times Pmin \times H^T + R \right)^{-1} \right) \tag{5.14}$$

Where $K = \left(\begin{smallmatrix} k00 & k01 \\ k10 & k11 \end{smallmatrix}\right)$, $H = \left(\begin{smallmatrix} h00 & h01 \\ h10 & h11 \end{smallmatrix}\right)$, $Pmin = \left(\begin{smallmatrix} Pm00 & Pm01 \\ Pm10 & Pm11 \end{smallmatrix}\right)$ and $R = \left(\begin{smallmatrix} r00 & r01 \\ r10 & r11 \end{smallmatrix}\right)$

First let´s evaluate

$$\left((H \times Pmin \times H^T + R)^{-1} \right) = \left(\frac{1}{(-h01(h11Pm11 + h10Pm10) - h00(h11Pm10 + h01Pm00) - r10)} \right) \cdots$$

$$\cdots \times \begin{pmatrix} h11(h11Pm11 + h10Pm01) + & -h10(h10Pm10 + h00Pm00) - \\ h10(h11Pm10 + h10Pm00) + r11 & h11(h01Pm11 + h00Pm01) - r01 \\ \vdots & \vdots \\ -h01(h11Pm11 + h10Pm10) - & h00(h00Pm00 + Pm10 + h10) + \\ h00(h11Pm10 + h01Pm00) - r10 & h01(h01Pm11 + h00Pm01) + r00 \end{pmatrix} \tag{5.15}$$

In order to simplify the notations, let´s call

$$detK = ((-h01\,(h11Pm11 + h10Pm10) - h00\,(h11Pm10 + h01Pm00) - r10)) \tag{5.16}$$

$$A = \begin{bmatrix} h11\,(h11Pm11 + h10Pm01) + \\ h10\,(h11Pm10 + h10Pm00) + r11 \end{bmatrix}$$

$$B = \begin{bmatrix} -h10\,(h10Pm10 + h00Pm00) - \\ h11\,(h01Pm11 + h00Pm01) - r01 \end{bmatrix}$$

$$C = \begin{bmatrix} -h01\,(h11Pm11 + h10Pm10) - \\ h00\,(h11Pm10 + h01Pm00) - r10 \end{bmatrix}$$

And $D = \begin{bmatrix} h00\,(h00Pm00 + Pm10 + h10) + \\ h01\,(h01Pm11 + h00Pm01) + r00 \end{bmatrix}$

The value of K is

$$K = \frac{1}{detK} \times \begin{pmatrix} (Pm01h11 + Pm00h01) \times C & (Pm01h11 + Pm00h01) \times D \\ + (Pm01h01 + Pm00h00) \times A & + (Pm01h11 + Pm00h00) \times B \\ \vdots & \vdots \\ (Pm11h11 + Pm01h10) \times C & (Pm11h11 + Pm01h10) \times D \\ + (Pm11h01 + Pm01h00) \times A & + (Pm11h01 + Pm01h00) \times B \end{pmatrix} \tag{5.17}$$

It is now possible to identify $K00$, $K01$, $K10$ and $K11$. By doing directly these computations, no algorithm to invert or multiply matrices is used and the program reacts faster.

5.3.3.4 Timing analysis

The MU sends data packets to the CU each 33ms. The CU processes them and sends them to the PC Unit where the data can be interpreted. The processing part made by the CU cannot be greater than 33ms-transmission time # 30 ms. The Kalman filter computed and implemented by the project group was first really slower than that (about 1 minute to compute the data). With all the modifications made by the group and reducing the numbers of filtered sensors to 8 (the 8 sensors used in the fuzzy logic) the group manages to compute the data in less time than 33ms. There are 3 main different ways to compute the time needed for the algorithm in the FPGA to run:

- The first one is very theoretical and cannot be apply for long algorithms where the result found is more than inaccurate. The frequency of the microprocessor in the FPGA is 50MHz that means that the time for an instruction to be performed is $\frac{1}{50e^6} = 2e^{-8}$s = 20ns. Figure 5.9 is an array which gives the number of stages for the main used operations:

	Nios II /f Fast	Nios II /s Standard	Nios II /e Economy
Pipeline	6 Stage	5 Stage	None
Multiplier *	1 Cycle	3 Cycle	None
Branch Prediction	Dynamic	Static	None
Instruction Cache	Configurable	Configurable	None
Data Cache	Configurable	None	None

Figure 5.9: *number of stages for the main used operations. Modified from [35]*

  By founding all the number of different operations and multiplying this number by the number of cycles and then by the frequency, the time needed to perform the algorithm can be found but it is very imprecise.

- The second method is to use the profiling tool in the software NIOS II IDE. To use it, the SOPC module performance counter has to be added in the hardware. Figure 5.10 shows the results obtained by this counter.

  The function highlighted is the C function with the Kalman filter for only 8 sensors. The time needed to perform this function is 37.99 ms. This amount of time is very problematic because, it is higher than the time between two frames that means that 1 frame on 2 will be lost, as summarized in Figure 5.11. This report does not validate these results. However, the group project notice an important slow down of the system when the linking to profile libraries is made in the NIOS II IDE project.

```
Each sample counts as 0.001 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 20.05      0.09     0.09        1    85.30    85.30  alt_busy_sleep
 12.56      0.14     0.05     7310     0.01     0.01  _fpadd_parts
 10.85      0.18     0.05        2    23.09    23.09  alt_file_locked
  8.93      0.22     0.04        1    37.99   264.71  function_kalman_8_sensor_without_baseline
  7.23      0.25     0.03    21120     0.00     0.00  __unpack_f
  7.11      0.28     0.03    10190     0.00     0.00  __pack_f
  5.95      0.31     0.03     2240     0.01     0.02  __mulsf3
  5.33      0.33     0.02        1    22.68   107.98  alt_sim_halt
  4.63      0.35     0.02        1    19.68   137.98  exit
  2.91      0.36     0.01                             mcount_allocate
  2.42      0.37     0.01        1    10.31   118.29  _exit
  2.39      0.38     0.01     5390     0.00     0.02  __addsf3
  2.00      0.39     0.01     1920     0.00     0.02  __subsf3
  1.75      0.40     0.01      560     0.01     0.02  __divsf3
  1.56      0.41     0.01                             __mcount_record
  1.03      0.41     0.00      160     0.03     0.03  __eqsf2
  0.70      0.41     0.00     1913     0.00     0.00  __muldi3
  0.46      0.42     0.00       10     0.19     0.19  altera_avalon_uart_write
  0.28      0.42     0.00      165     0.01     0.01  __gtsf2
  0.23      0.42     0.00       10     0.10     0.20  __srefill
  0.23      0.42     0.00       10     0.10     0.10  __swrite
  0.23      0.42     0.00       10     0.10     0.10  altera_avalon_uart_read
  0.23      0.42     0.00        1     1.00     1.00  _malloc_r
  0.23      0.42     0.00                             alt_get_errno
  0.19      0.42     0.00                             alt_find_file
  0.16      0.42     0.00                             alt_alarm_stop
  0.13      0.42     0.00                             mat_tran
  0.07      0.42     0.00        5     0.06     0.06  alt_irq_register
  0.04      0.43     0.00        2     0.09     0.09  alt_find_dev
  0.01      0.43     0.00                             alt_get_errno
  0.00      0.43     0.00     2462     0.00     0.00  alt_irq_handler

...
```

Figure 5.10: *Report of the time needed by the different functions used in the project. The needed time for the C function "function_kalman_8_sensors_without_baseline" is highlightened and shows that it requires 37.99ms.*

Indeed, the request between modules in profile libraries and modules used in the project, discord with the good working of the entire system. The group decided to remove the linking libraries of the project.

- The third method uses the scope to found the time of a frame going out of the CU. The frame coming out from the control unit is composed by 8 packets of 8 bits, that are the data filtered with the Kalman filter for only 8 sensors. The time of a frame is 2.1ms that is really lower than 33ms. However this time does not give the time used for the computations in the FPGA. However, if the distance between two frames is 33 ms that means that the time needed for sending data from the MU, computing data on the CU and sending them is lower than the time between two frames. So no overlapping between signals is present that means that the computation time is lower than 30ms. The group manages to implement the Kalman filter alone, the fuzzy logic alone and the Kalman filter with the fuzzy logic. In the case of the Kalman filter with the fuzzy logic, the calibration time needed by the Kalman filter is around 5 minutes but then the algorithm is working fine. For more than 8 filters, hardware acceleration should be considered as explained on the following section.

Figure 5.11 summarizes the time between two packets send by the MU that is not the time needed by the FPGA to process the input values filtered by the Kalman filter (on only 8 sensors). Above the

results given by the report, using a software profile and below the time of the output frame after the FPGA processing.

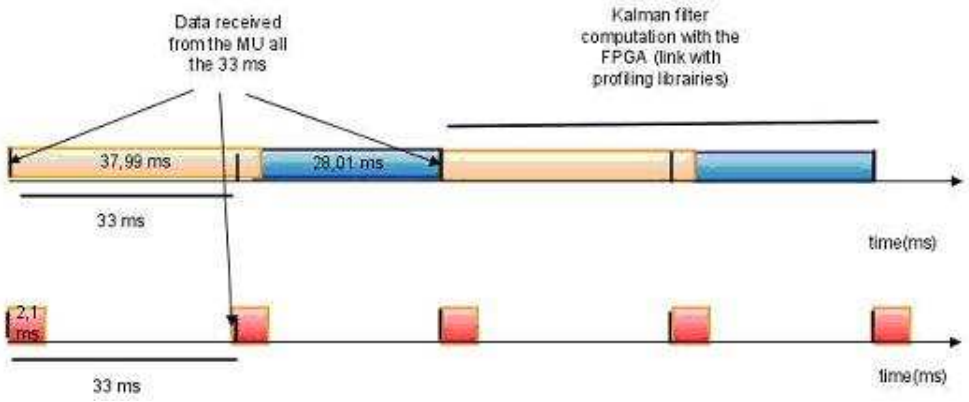


Figure 5.11: *Time between two packets sent by the MU*

5.4 Conclusion about the implementation

The project design on the chosen platform Altera DE2 is done using SOPC Builder. The softcore processor NIOS II is set to execute the C translated code of the fuzzy logic and Kalman filter. The fuzzy logic, alone, is successfully implemented. It runs for the 24 sensors. The Kalman filter implementation, alone, is also successfully done. However, a translation from MATLAB code to C code has firstly been made. The matrices computation functions, as matrices multiplication or matrix inversion, have been done using a tool box found on the Internet where the matrices are declared as $double$. But the slowness of execution of some of the matrices computation functions obligate the project group to create its own functions declaring the matrices as $float$ to reduce the computation time. An other modification has been made over the matrices multiplication and matrix inversion functions because of the major use of computation instead of implementing one Kalman filter for the 24 sensors, the group project decided to run 24 Kalman filters in parallel. This modification permits as well to save memory space usage.
Then, the combination of the fuzzy logic preceded by the Kalman filter is conducted on the 8 sensors forming the joystick on the palatal plate. A timing analysis concludes to that the execution time of both programs is less than 33ms which is the time between two scans of the sensors. It means that the CU has enough time to receive the state of the sensors, process their signals and apply the mouse cursor control before another data packet is sent.

Chapter 6

Tests and validation

This chapter is relative to both Architecture and Application domains from the A^3 design methodology, shown in Figure 6.1, especially to the discussion and comparison of the results of the implementation, explained in Chapter 5, with the constraints established in Chapter 1.

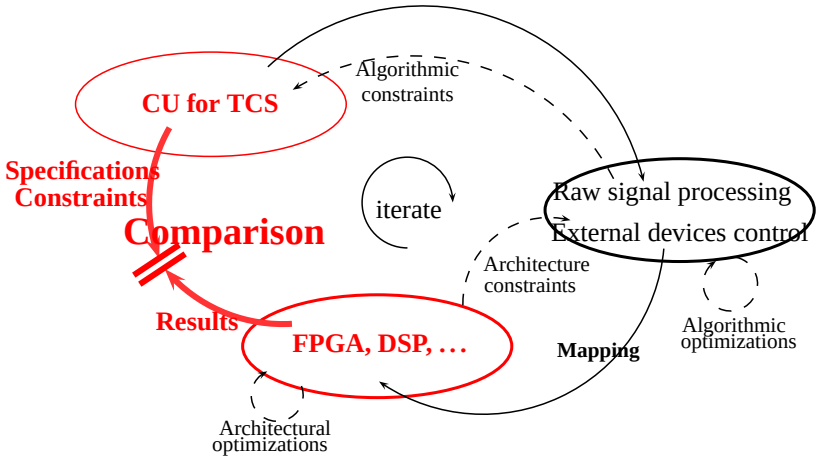


Figure 6.1: *The A^3 design methodology applied to Chapter 6*

6.1 Overview

Once the implementation is done on the FPGA board, the validation of the tests regarding the constraints defined by the application are conducted. Section 6.2 presents the tests results of the Fuzzy Logic implementation, followed by a comparison between MATLAB tests and FPGA tests for the Kalman Filter, in Section 6.3. The third Section of this Chapter, Section 6.4, presents the results of the goal of this project, that is the implementation and working of the Kalman Filter with the Fuzzy Logic. And finally, a possible optimization is discussed in Section 6.5.

6.2 Fuzzy logic implementation validation

The fuzzy logic test implementation is only graphical. Indeed, the time needed for the FPGA to compute it is lower than the time between two frames (33ms), so the implementation can be considered successful. Graphical simulation is made using MATLAB to display a cursor and to control it.
The principle is quite simple: a MATLAB toolbox allows the user to display a cursor. The screen matrices depends of the resolution of the monitor used. The set of coordinates (0,0) corresponds to the extreme up right size of the monitor and the set of coordinates (1280,1024) (on a 18-inch monitor) corresponds to the extreme down left size. The group project sets the initial coordinates to the middle of the screen : (640,512). Indeed, when no activation is produced in any sensor, the position of the piercing (AU) is supposed to be in the middle of the MU device, that is why the initial position of the cursor in the screen has these values. Furthermore, if the initial position of the cursor was on one corner, the cursor can disappear when the algorithm will run because values outside the screen matrix cannot be displayed. Then the fuzzy logic (with also the function created by the HST research team) runs on the FPGA. The output of this function is a set of (x, y) coordinates which depends on the position of the AU in the MU as explained on the Chapter 4. Each coordinates (x, y) can have a value between 127 and -127. These two max and min values cannot be modified using the RS 232 protocol between the CU and the PC Unit. In fact, the data transmitted between those two are coded with 8 bits but with no sign bit. The values obtained in the CU are between -127 and 127 and the values received by the PC Unit are between 0 and 255. The addition of 127 to the value transmitted by the CU and the subtraction of this constant at the reception by the PC Unit permits to fix this problem.
The algorithm used to control the cursor with MATLAB always add the new coordinates received to the previous coordinates. This method is closer to the use of a joystick, to command a plane for example than the use of a mouse where the sensor area (the MU in the project case) should correspond to the screen matrix). However, the use of the joystick is very well suited with the control of the wheelchair which is one of the main goals of the TCS.

The main problem when the new coordinates are always added to the previous one is when one sensor is activated for too much time, the cursor will disappear from the screen matrix. For instance, when the sensor 2 (that is the sensor to go up) is activated the cursor will go up and disappear from the screen matrix. The group project defines a closed square matrix area. This closed square matrix area is the screen matrix. In this way, when the cursor goes out of the square matrix area (that means that the cursor goes out of the screen), the coordinates of the cursor are reassigned to the middle of the screen. A graphical movie (fuzzy_logic.avi) is present on the CD related to this thesis and shows the working of the fuzzy logic algorithm graphically.

The main problems highlighted in this video are the following:

- A long calibration is needed to obtain the maximum calibration gap than an activation can reach for each sensor. With this system if the activation gap of a sensor is wrong, too high for example, the activations related to this sensor will never be maximal, so the cursor cannot go directly in the screen area related to this sensor.
- If the baseline starts growing, the algorithm will work. But if the baseline starts decreasing then the algorithm will interpret it as small activation. In consequence the cursor will start to shift in one direction. The temperature variations are the main limitations of this algorithm.
- Noise present in the values received from the MU is very high and sometimes it can be interpreted as small activations. For instance when no activation is present, noise makes the cursor jump around the central position. This can be very annoying if the cursor is used to select data on the screen. The HST research group decides to put a threshold to filter the noise. No activations are considered if the value of the activation is lower than the threshold (practically 60 for a baseline around 200).

The main limitation of the threshold is that small activations are not detected because interpreted as noise. The sensors loose their sensitivity and on the screen the cursor seems “heavy” to control.

6.3 Comparison MATLAB and FPGA implementations for the Kalman filter

The test made at the scope (cf Paragraph 5.3.3.4) shows that the time between two frames sent by the FPGA is constant and equal to 33ms. The implementation of the Kalman filter limited at 8 sensors only on the mouth area is successful. However, there are some differences between the results obtained by MATLAB simulation and the results obtained by the implementation on the FPGA.

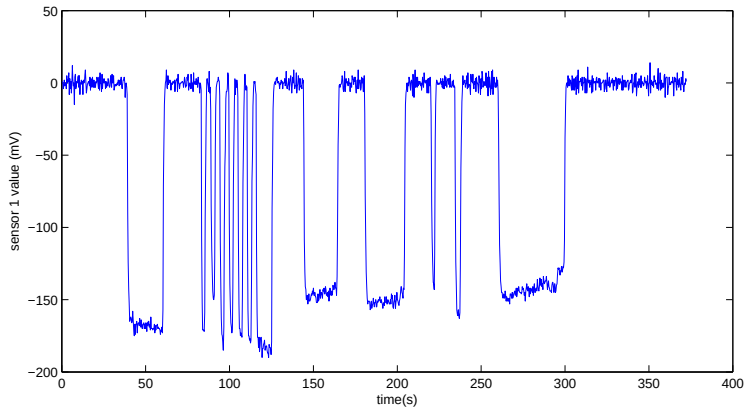


Figure 6.2: *The Kalman filter directly running on the FPGA and applied on sensor 1.*

The magnitude of the noise is increased in comparison with the MATLAB implementation whereas the same covariances matrices Q and R. This is probably caused by the add of a new device between the MU and the PC Unit, which add computational noise. The covariances matrices Q and R should be tuned to obtain a better solution. Furthermore, the sensibility of the sensor is lower, and sometimes a peak appears after an activation. The peak is due to the time response of the Kalman filter in the FPGA that is larger than in MATLAB (because the FPGA has a lower clock frequency than the PC and also because it is less suited to make large computations than the PC). In fact, the detection of an activation in the Kalman filter algorithm is made by the logical decision function based on two detectors: $detect$ and $detect\_fast$. The detector $detect$ is using a high pass Kalman filter. The high pass Kalman filter begins to oscillate when a variation in the baseline is produced. This oscillation permits to detect an activation and the baseline is computed in consequence as explained in the Kalman filter Section 2.4. However, if the activation is detected with a little delay due to the computation time of the Kalman high pass filter, the baseline created with the values before the activation will be inexact.

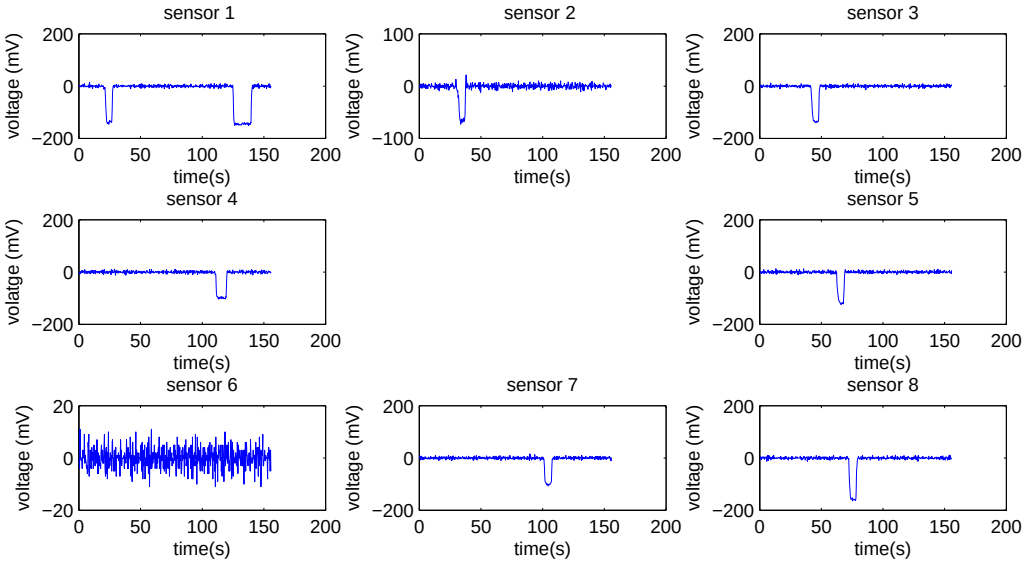


Figure 6.3: *All value of the sensors when the piercing is moved in the mouse area of the MU. The disposition of the sensor is the same than on the MU. The analysis of the time when the activations appear gives the path of the piercing in the MU. Such analysis can be a way to create a mouse cursor instead of a joystick. The piercing starts on sensor 1, sensor 2, sensor 3, ...*

The main advantages of the Kalman filter implemented on an FPGA are listed here below:

- The value of the noise (less than 10/200, as shown in Figure 6.3, sensor 6) is reduced and can be easily removed from the useful signal,
- No baseline is present. The system is free from the baseline shift variation due to temperature change.

On other side, the implementation of the Kalman filter suffers from negligible drawbacks:

- The system needs a lot of time to compute the first iterations and the first computation. Actually the matrices used in the Kalman filter have to be stored in the memory of the FPGA and after few runs of this algorithm on the FPGA, the matrices are almost constant so the system can be very fast. However this auto calibration needs time and can reach the minute depending on the parameters used in the algorithm,
- The activation of the sensors is sometimes inaccurate due to the small delay introduced by the FPGA to perform the computation. This small delay can degrade the sensibility of the sensors or causes deactivations.

However, the drawbacks introduced are minor and they can be corrected by tuning the Kalman filter (modifying the covariances matrices or the logical decision algorithm for instance).

6.4 Test with Fuzzy Logic and Kalman filter

The fuzzy logic was implemented correctly such as the Kalman filter limited to 8 sensors. The group follows the advices of the HST research group to implement the fuzzy logic without using directly the

raw data coming from the MU but the data filtered with the Kalman filter. Some code modifications are made in order to make it possible. The Kalman filter uses an infinite loop to read the data from the MU, so the call to other functions ($Fuzzy\_getXY$ for instance) with other loops slow down the computations. In consequence, the group decides to create only one function with the Kalman filter and the fuzzy logic. Fortunately, the computation time for both the algorithms is lower than 33 ms and these two algorithms can be implemented at once. The main drawback of this implementation is the auto calibration time (explained in the paragraph 6.2) that can reach several minutes. In the movie linked with this thesis (fuzzy_logic_with_kalman.avi), the moving of the cursor when the piercing (AU) is moving on the MU, is highlighted. The noise as the baseline are totally removed and the cursor keeps the current position when no activation is performed. The results are satisfying in spite of the apprehension at the beginning of the project (linked on the computation limits of the FPGA). The main limitation of this implementation is the limitation of the Kalman filter: the cursor is not so sensitive due to the activation detection. However, this problem can be fixed by tuning the Kalman filter and the logical decision function (used to detect the activations). This implementation is very well suited with the control of the wheelchair where the activations have to be stronger before being sent to the wheelchair.

6.5 Optimization: the hardware acceleration

FPGAs are components made to run HDL programs. The softcore processor NIOS II permits the execution of C codes but the interface between the processor and the FPGA is not optimized. Altera provides a tool, C2H, which permits to transform C-code to a Hardware Description Language (HDL). This transformation is called hardware acceleration. Hardware acceleration provides better results when the compiler mode of work is known by the designer in order to structure the C code to optimize translation into Hardware implementation. C2H associates one Hardware acceleration module to each C function. Each subfunction is accelerated on separated modules.

6.5.1 C2H introduction

C2H is acronym for C code to Hardware Acceleration Compiler. It is used to implement a C function directly in Hardware. C2H compiler also allows debugging of the C code before translating it into a hardware acceleration unit that will be related to the SOPC Builder architecture.

Performance can be increased in several orders of magnitude. Each C instruction is directly transcoded into a Hardware unit and facilitating in this way the debugging of the translation.

The following acceleration features are implemented:

- Parallel execution of unrelated instructions,
- Pipelining of code inside loops and memory access.

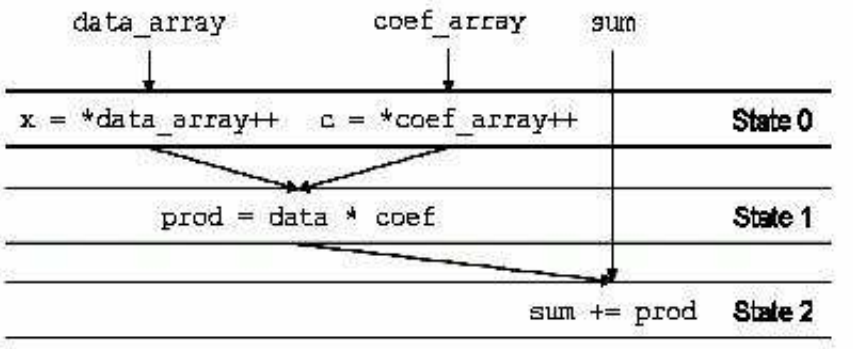
C2H is interfaced with SOPC to integrate generated hardware module into the global architecture and with Quartus to recompile the hardware design.

The following procedures are used to change a C program into an Hardware Acceleration Module (HAM):

- Arithmetic and Logical operators are directly translated into their equivalent in HAM e.g. "$a++$" C instruction is converted into 32 bit up counter, "$+=$" into 32 bits adder,
- Assignment is translated into registration of a variable e.g. "$x = a \times b$" is translated into 64 bits multiplier and result is stored in Register (in one clock cycle). However, some operations are unregistered such as Shifting by a constant number of bits and Inversion,

- Iterations are pipelined meaning that the next instruction is loaded before the execution of the current one is completed. Execution is split into states (related to each clock cycle),

  Figure 6.4 is the state diagram and associated pipelining:



Time	Iteration 0	Iteration 1	Iteration 2
0	(State 0) x = *data_array++ c = *coef_array++		
1	(State 1) prod = c * x	(State 0) x = *data_array++ c = *coef_array++	
2	(State 2) sum += prod	(State 1) prod = c * x	(State 0) x = *data_array++ c = *coef_array++
3		(State 2) sum += prod	(State 1) prod = c * x
4			(State 2) sum += prod

Figure 6.4: *HAM for a loop. Modified from [55]*

States 0 and 1 of iteration 1 are pre-fetched and executed before iteration 0 is completed and so on for other iterations. However, if there are dependencies between iterations, the pipelining is delayed to wait for the result of the current iteration before pre-processing of the next iteration (called Loop-Carried dependencies).

- Conditional execution ("$if \ldots then \ldots else \ldots endif$", "$switch$") are implemented with a multiplexer controlled by the condition,

- Memory access is made through Avalon Read/Write Master to make the memory access faster,

6.5.2 Possible solution for all the sensors

The group project wants to filter all the 24 sensors with the Kalman filter and also runs the fuzzy logic on the FPGA. However, the computation speed can be not sufficient for all the sensors. Hardware acceleration can be a possible way to solve this problem, although hardware acceleration can be only applied on function.

Hardware acceleration can be hardly applied to the Kalman filter. Indeed, the Kalman filter is made directly in the infinite loop where the data input reading is performed. The modification of this code into a function can be done into two different ways:

- The creation and the definition of all the matrices is done inside the function which means that for all the iterations they are defined and created again. The memory space and the computation units can be substandard with this solution,
- A Kalman filter function can be defined with only the computational part of the Kalman filter. In this case numerous parameters have to be called and returned. The Kalman filter is recursive, so for numerous variables, current and previous values have to be saved. The memory space would not be enough with this solution.

The creation of a function with all the Kalman filter code is not advised but small computation part can be defined as function in order to be accelerated in Hardware. Furthermore, the fuzzy logic algorithm can be accelerated in hardware which can make the system faster.

The group project studied and tried to implement this optimization method but several issues occur not allowing any noticeable results. Meanwhile, the project deadline approached.

6.6 Conclusion about the tests and validation

The tests of the implementation of the fuzzy logic correspond to check if moving the AU on the 8 sensors forming the joystick on the palatal plate makes the computer mouse cursor move.
The tests of the Kalman filter are compared to the simulation made on MATLAB. The FPGA implementation introduces a higher voltage level of noise due to the add of the radio PCB board, which is the microcontroller prototype design by the HST research team. But this level is only 10mv out of 200mv. This value of noise cannot be considered as an activation by the CU as an activation value is ranged between 100mv and 200mv (from the weakest to the strongest activations). The time response of the Kalman filter FPGA implementation is larger hence less sensibility of the sensors. The system needs a lot of time to compute the first iteration and the first computation for the auto calibration.
The tests of the implementation of Kalman filter followed by the Fuzzy logic is performed. For the 8 sensors forming the palatal plate, the results are very satisfying. The mouse cursor moves in smooth and accurate way while moving the AU on the palatal plate. However, the sensors are less sensitive and the user needs to apply strong pressure with the AU.

Chapter 7

Conclusion & Perspectives

7.1 Ending discussion

For disabled people affected by mobility disabilities, the control of their environment is a constant fight. Simple moves as opening a door, moving their wheelchair or switching the light are impossible without the help of a third person. Access and use of common modern tools, e.g. computer, mobile phone also requires help. New technologies permits to take advantage of every movement disabled people can do. Control systems are now available, based on eyes, jaws or head movements detection. These systems add complex architectures visible by all and increase the feeling of difference. Some control systems based on tongue movement have been declared preferred by disabled people because they are hidden in the mouth even if their accuracy is less efficient than other control systems. But this kind of systems does not permit a usage during eating or drinking. A research team from the Department of Health Science & Technology (HST) at Aalborg University (AAU), by using Faraday´s law of induction for a coil, has developed a tongue control system able to be used during eating and drinking. A ferromagnetic material pierced on the tip of the tongue, the Activation Unit (AU), changes the properties of the coils, implemented in a palatal plate, the Mouthpiece Unit (MU), by approaching or moving away from them. This change of properties of the coil is referred to as an activation of a sensor and is sent by wireless to a Central Unit (CU) which gather all the activations, process them and send the result to the right external device to be control, e.g. a computer mouse cursor. The first prototype is able to control only the mouse cursor of a computer and some perturbations do not make the system accurate enough to be used by disabled people.

To enhance the TCS, an implementation of the existing algorithms of the CU on a more powerful, to improve the data processing, to add new external devices interfaces, platform is requested. The goal of this 10^{th} Semester ASPI project is to design a new CU platform and to implement different algorithms of data processing and external devices control interfaces on this new platform which must fulfil the 3 sub-goals defined in Chapter 1, Section 1.2. The new CU platform must :

1. process the raw signal from the MU to produce a better signal quality to enhance the accuracy of the system by removing the noise and the baseline wandering

2. choose a flexible and extendible platform in order to implement all the algorithms existing and the algorithms created. Furthermore this platform needs different inputs and outputs in order to plug useful devices for future applications.

3. implement all the algorithms and create output signals that can be interpreted by a PC Unit.

First of all, MATLAB simulations were performed to understand carefully the input data and the various problems of the prototype. Then the creation of a filter to remove noise and the baseline was achieved. This filter totally created by the group uses a Kalman filter to perform the noise and the baseline wandering suppression. First a low pass Kalman filter was created to remove the noise and then the Kalman filter was upgraded to remove the baseline wandering. Indeed the baseline definition needed to be defined before removing it. The group project made a high pass filter and a logical decision function to define recursively the baseline.**The objectives were successfully achieved and the direct translation of this algorithm from MATLAB to C code was done.**
The choice of the platform was the next step. Two kind of platforms leads the industry nowadays: FPGAs and DSPs. FPGAs are programmable architectures and can be used in different applications with high parallelism and a low frequency requirements. DSPs are fixed architecture with a clock frequency in general bigger than FPGAs. The computation performances of a DSPs are superior of the ones in a FPGA but connectivity with external devices is limited. DSPs are less expensive than FPGAs, easier to program (directly programmed with C code) and consume less power. The limited platforms available obligate the group to choose a FPGA. The FPGA used for the study and the implementation of the algorithms, called Altera DE2, has a microprocessor (NIOS II) on it. In such a way, code C can be directly implemented and complex computations can be done.

The FPGA architecture was totally defined using SOPC Builder interface to make easier the add of the different modules and interfaces but also to connect them together. The presence of the software processor NIOS II allows the user to directly implement high level code like C code (using the software NIOS II IDE). Obviously, more restriction are existing in comparison with DSP compilers.
HST research group has already created an algorithm (named $Fuzzy\_getXY$) to control a cursor using 8 sensors in the MU. Using the position of the AU in the MU, two coordinates are produced and sent to the PC Unit. Fuzzy logic is used to allocate at each sensor an activation value between 0 and 1. Normal logic allocates only boolean values (0 or 1) to each sensor thus only 8 positions on the screen matrix would be reach, that is not the case using the fuzzy logic were intermediate positions between two sensors are available.
This algorithm was implemented in an 8 bit microcontroller using an adapted C code. The use of the fuzzy logic algorithm in the new board was not in any way problematic. In fact, C code for microcontroller is very closed to C code for FPGA. Results were concluding: the cursor can be controlled by the AU moving inside the MU. However, the presence of the noise, the baseline wandering problem and the constant re calibration of the cursor, make this solution limited.
The Kalman filter algorithm designed and developed by the group, needed to be applied before running the fuzzy logic algorithm. The implementation of the Kalman filter was dissimilar from the MATLAB implementation. Deep modifications have been applied such as the diminution of matrices size. The parallelisation of the solution was achieved in order to obtain squares matrices of dimension 2. Moreover, the algorithms used to compute multiplication or matrices inversion have a very high complexity which means that they are using numerous board resources that slow down the algorithm running. **The group manages to run the Kalman filter algorithm, trying to avoid complex computations, using parallelism and reducing the number of filtered sensors to the 8 present in the MU.**
In order to compare the solution with and without Kalman filter on a physical application, fuzzy logic algorithm was applied after running the Kalman filter to command the mouse cursor. Although a long self calibration period, the cursor can be controlled with easiness. The cursor is not jumping around as it does without Kalman filter, which means that the noise reduction is well achieved. Furthermore the baseline is totally out: the cursor is not shifting alone anymore. However, the Kalman filter needs to be tuned (changing the covariance parameters and the threshold in the logical decision algorithm) to reach more sensitivity in the cursor control.

7.2 Perspectives

7.2.1 Short term perspectives

Because of time restrictions, the following objectives have not been fulfilled:

1. The implementation of the Kalman filter is made for only 8 sensors. In the final solution all the 24 sensors should be corrected with the Kalman filter. However, this solution can be problematic because of the limited size memory and computational resources of the FPGA. Hardware acceleration tool provided by Altera can be useful to manage the implementation of the complete Kalman filter. Furthermore the choice of a new platform can be investigated, such as DSP in order to perform computations with higher complexity and less parallelism (for instance speech processing). This choice cannot be guided only by the willing to implement the entire Kalman filter but if some other functionalities are added (such as Keyboard control,...) requiring sizeable computational resources, the change of the architecture can be advised.

2. The auto calibration time has to be reduced. When the Kalman filter and the fuzzy logic are both running, the auto calibration time of the filter is too long and cannot suit with real life applications.

3. The flash memory has to be programmed in order to store all the hardware architecture and the C programs. In such a way no reprogrammation of the board has to be made each time the board is switched on. The programmation of the flash memory is absolutely required for the final product.

4. The Kalman filter applied on only 8 sensors has to be tuned in order to control the cursor in an smoother way and with the maximum precision.

7.2.2 Long term perspectives

The creation and the addition of other algorithms for other applications needs time and it is not suitable with the project time period. However the HST research group can create other applications such as the control of a keyboard, the control of the wheelchair, TV channels selection, emergency signal, phone interface,...
This master thesis is aimed to show a possible way of algorithm implementation on a different board. The board used is a middle solution between the two main existing platforms. In fact, the board used is a FPGA with a software microprocessor which can be programmed like a DSP. In such a way, the advantages of the FPGAs such as multi existing input and output interfaces are present. Because of the software microprocessor, code C can be easily implemented in it and a real time gain is made. This kind of platform is a recommended trade-off for hardware beginners. The members of HST research group will have to choose the right platform depending on the other applications between an easy programmation and better performances, between high computational performances and multi interfaces,... In any way the board selection cannot be made regardless of the practical applications.

Bibliography

[1] http://www.who.int/classifications/icf/en/ (last visit the 08/03/09)

[2] http://v1.dpi.org/lang-en/resources/topics_detail?page=74 (last visit the 08/03/09)

[3] http://www.disabled-world.com/artman/publish/article_0082.shtml (last visit the 08/03/09)

[4] http://en.wikipedia.org/wiki/Mouth (visited on the 08/03/09)

[5] "'Comparison of Computer Interface Devices for Persons with Severe Physical Disabilities"', Lau C. and O´Leary S., The American Journal of Occupational Theraphy, vol. 47, pp.1022-1030, 1993

[6] "'Palatal tongue controller"', Clayton C, Platts RGS, Steinberg M, and Hennequin JR, J. of Micro-computer Applications, 15, pp. 9-12, 1992

[7] "'Apparatus for controlling peripheral devices through tongue movement, and method of processing control signals"', Buchhold N, United States Patent no. 5,460,186, 1995.

[8] Patent: WO9307726, New Abilities Systems Inc., 1993.

[9] "'A Tongue Based Control for Disabled People"', Lotte N.S. Andreasen Struijk, Center for Sensory Motor Interaction, Aalborg University, DK-9220 Aalborg, Denmark

[10] http://hyperphysics.phy-astr.gsu.edu/hbase/electric/farlaw2.html#c1 (last visit the 09/03/2009)

[11] http://en.wikipedia.org/wiki/Inductor (last visit the 23/03/09)

[12] "'Requirement Specifications"', Morten Enemark Lund

[13] nRF24L01+ Single Chip 2.4GHz Transceiver Product Specification v1.0 http://www.nordicsemi.no/files/Product/data_sheet/nRF24L01P_Product_Specification_1_0.pdf (last visit the 02/06/09)

[14] ASPI Introduction slides, AAU, 2009.

[15] "'Digital Signal Processing amd spectral Analysis"', Thierry Pitarque, Polytechnique Nice

[16] "'The Scientist and Engineer's Guide to Digital Signal Processing"', chapter 15, Steven W. Smith California Technical Publishing

[17] "'On the Theory of Filter Amplifiers"', Wireless Engineer (also called Experimental Wireless and the Wireless Engineer), vol. 7, 1930, pp. 536-541

[18] "'The Scientist and Engineer's Guide to Digital Signal Processing"', chapter 20, Steven W. Smith California Technical Publishing

[19] "'A Bessel Filter Crossover,and Its Relation to Others"', RaneNote 147© 1998, 2002, 2006 Rane Corporation, Ray Miller

[20] http://www.analog.com/static/imported-files/tech_docs/dsp_book_Ch19.pdf (last visit the 04/04/09)

[21] http://sepwww.stanford.edu/sep/prof/pvi/spec/paper_html/node18.html (last visit the 04/04/09)

[22] http://www.technick.net/public/code/cp_dpage.php?aiocp_dp=guide_edft_013 (last visit the 04/04/09)

[23] http://sepwww.stanford.edu/sep/prof/pvi/spec/paper_html/node19.html#SECTION00142000000000000000 (last visit the 04/04/09)

[24] "Neural control of computer cursor elocity by decoding motor cortical spiking activity in humans with tetraplegia" Sung-Phil Kim, John D Simeral, Leigh RHochberg, John P Donoghue and Michael J Black, Journal of Neural Engineering 2008

[25] "'Kalman filter theory and practice"', M.S.Grewal, A.P. Andrews

[26] "'Electronic and Loop-controlled systems"', Sylvie Icart, Polytechnique Nice

[27] "'An introduction to the Kalman filter"', G. Welch and G. Bishop

[28] "'Stochastic Models, Estimation, and Control"', vol. 1, by Peter S. Maybeck

[29] http://focus.ti.com/lit/an/sloa024b/sloa024b.pdf (last visit the 19/04/09)

[30] Cauer, W, "'Die Verwirklichung der Wechselstromwiderst ande vorgeschriebener Frequenzabh angigkeit"', Archiv für Elektrotechnik, 17, pp355 - 388, 1926.

[31] Butterworth documentation on MATLAB -doc butter

[32] http://www.mathworks.com/access/helpdesk/help/toolbox/fuzzy/index.html (last visit the 05/05/09)

[33] http://en.wiktionary.org/wiki/fuzzy (last visit the 05/05/09)

[34] http://www.xilinx.com/ (last visit the 25/04/09)

[35] http://www.altera.com/ (last visit the 25/04/09)

[36] http://www.atmel.com/ (last visit the 25/04/09)

[37] http://www.latticesemi.com/ (last visit the 25/04/09)

[38] Les circuits FPGA - Concepts de base, architecture et applications by François Verdier - http://www.etis.ensea.fr/ verdier/cours/cours_Xilinx_2006.pdf (last visit the 25/04/09)

[39] Detailled DSP Hardware (aDSP2199X) architecture - http://files.iai.heig-vd.ch/Enseignement/Supports%20de%20cours/DSP%20architecture%20et%20applications/Analog%20Devices%20Visua 2199x%20Hardware%20Reference.pdf (last visit the 02/06/09)

[40] http://www.adaptyv.com/doc/xilink/Xilink-FPGAsandDSP.pdf (last visit the 02/06/09)

[41] http://www.vlsi.ee.upatras.gr/ mgalanis/pubs/DSP2002.pdf (last visit the 02/06/09)

[42] NIOS II Architecture overview. http://www.altera.com/literature/hb/nios2/n2cpu_nii51002.pdf (last visit the 02/06/09)

[43] Altera Product catalogue. http://www.altera.com/literature/sg/product-catalog.pdf (last visit the 02/06/09)

[44] Joakim Ögren. http://www.hardwarebook.info/Serial_(PC_9) (last visit the 02/06/09)

[45] http://www.wisegeek.com/what-is-sdram.htm (last visit the 02/06/09)

[46] William F. Egan, Frequency Synthesis by Phase Lock (2nd ed.), John Wiley and Sons, 2000 (provides useful Matlab scripts for simulation)

[47] http://www.ict.kth.se/courses/IL2207/0810/Labs/Lab1/tut_DE2_sdram_vhdl.pdf (last visit the 02/06/09)

[48] STORAGEsearch. http://www.storagesearch.com/bitmicro-art1.html. (last visit the 02/06/09)

[49] http://www.altera.com/products/devices/serialcfg/features/scg-adv_features.html#isp (last visit the 02/06/09)

[50] http://www.altera.com/literature/ug/ug_nios2_flash_programmer.pdf (last visit the 02/06/09)

[51] http://www.altera.com/literature/lit-qts.jsp (last visit the 02/06/09)

[52] NIOS II Software Development Kit. http://www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf (last visit the 02/06/09)

[53] Coppersmith, D., Winograd S., Matrix multiplication via arithmetic progressions, J. Symbolic Comput. 9, p. 251 - 280, 1990.

[54] Cormen, Thomas H.; Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford (2001) [1990]. "'Introduction to Algorithms"' (2nd ed.). MIT Press and McGraw-Hill. pages. 755 - 760. ISBN 0-262-03293-7.

[55] NIOS II C2H Compiler User Guide. http://www.altera.com/literature/ug/ug_nios2_c2h_compiler.pdf (last visit the 02/06/09)

List of Figures

List of Tables