


Technische Universität Berlin
Faculty IV - Electrical Engineering and Computer Science
Department of Energy and Automation Technology
Electrical Drives

Master's Thesis

DESIGN AND PROGRAMMING OF THE SOFTWARE TO CONTROL THE SENSOR AND SENSORLESS BLDCM OF THE ELECTRICAL PROPULSION SYSTEM FOR THE SOLAR AIRCRAFT

presented by

Pablo Armero Almazán
Matr.-Nr.: 362142

Date of submission: **August 28, 2014**

Supervisor: **Prof. Dr.-Ing. Uwe Schäfer**
Tutor: **Rong Dong**

Declaration

I hereby declare in lieu of an oath that I have produced this work by myself. All used sources are listed in the bibliography and content taken directly or indirectly from other sources is marked as such. This work has not been submitted to any other board of examiners and has not yet been published. I am fully aware of the legal consequences of making a false declaration.

Eidesstattliche Erklärung

Ich versichere an Eides statt, dass ich die vorliegende Arbeit selbstständig angefertigt habe. Alle benutzten Quellen sind im Literaturverzeichnis aufgeführt und die wörtlich oder inhaltlich entnommenen Stellen als solche kentlich gemacht. Die Arbeit wurde bisher keiner anderen Prüfbehörde vorgelegt und noch nicht veröffentlicht. Ich bin mir bewusst, dass eine unwahre Erklärung rechtliche Folgen hat.

Place/Date

Signature

There is no lack of time, there is lack of interest.
Because when one really wants something,
dawn becomes day, Tuesday becomes Saturday
and a moment becomes an opportunity.

Acknowledgements

First and foremost, I have to thank my parents, without whose sacrifices and efforts I would not be able to reach this goal. I only hope they are proud of what their sons have achieved, their immeasurable work finish here, but my journey has just started. I also need to thank my brother, who I took as an example and encouraged me to improve day by day, and the rest of my family, who have always been there to support me.

I would like to express my gratitude to the whole team of CITCEA-UPC, but namely to Samuel Galceran, who have trusted me and helped me in many ways, academically and professionally.

A very special thanks goes out to the staff of the Electrical Drives Institute from the Technische Universität Berlin, for making my stay in Berlin easier and help me with all I needed.

Last but not least, I must acknowledge my girlfriend, for being there to support and encourage me at any moment and for bearing me when I am insufferable.

In conclusion, I recognize that this whole path that now comes to an end would not have been possible without all this surrounding, which is as important as sometimes unrecognised.

Abstract

A Brushless Direct Current (BLDC) machine and converter are applied for the electric propulsion system of the solar aircraft. BLDC motors are electronically commutated instead of mechanically, as Direct Current (DC) motors do, so the position of the rotor must be known to do the commutation at the right moment.

The control of BLDC motors can be done in sensor or sensorless mode. being the use of Hall-effect sensors the most widespread use inside the sensored control. The use of the hall sensors divide a complete revolution of the motor in 6 sectors, where in each sector only two phases are conducting (block commutation, trapezoidal control).

However, considering the complexity of the high altitude, the electrical propulsion system cannot work properly if the hardware of the Hall sensors is broken. Therefore a sensorless control is necessary.

The BLDC motor provides an attractive candidate for sensorless operation because the nature of its excitation inherently offers a low-cost way to extract rotor position information from motor-terminal voltages. The implemented sensorless technique consists of sampling the non-fed phase voltage and comparing it to half of the DC-link voltage to detect the zero crossing point.

The most important part of the sensorless control is the sample of the non-fed phase, so the software must synchronise the sample with the PWM signal and apply the necessary delays in order to avoid commutation spikes and execute a correct BEMF measure.

The software must be able to control the motor in both sensored or sensorless mode and to change from one to the other if an error occurs, so the code has to be compatible between the two modes.

Contents

Acronyms

μC	Microcontroller
ADC	Analog-to-Digital Converter
BEMF	Back Electro-Motive Force
BLDC	Brushless Direct Current
CAN	Controller Area Network
CCU6	Capture/Compare Unit 6
CPU	Central Processing Unit
DAVE™	Digital Application Virtual Engineer
DC	Direct Current
DSP	Digital Signal Processing
EMF	Electro-Motive Force
FOC	Field Oriented Control
GPIO	General Purpose Input/Output
GPT	General Purpose Timer
HW	Hardware
IGBT	Insulated Gate Bipolar Transistor
IGCT	Integrated Gate Commutated Thyristor
ITC	Interrupt Controller
JTAG	Join Test Action Group
LED	Light-Emitting Diode

MAC	Multiply-Accumulate unit
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor
PEC	Peripheral Event Controller
PI	Proportional-Integral
PMSM	Permanent Magnet Synchronous Machine
PWM	Pulse-Width Modulation
RPM	Revolutions Per Minute
SW	Software
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus
ZCP	Zero-Crossing Point

List of Tables

List of Figures

List of Code Snippets

1. Introduction

1.1. Motivation

Electrical mobility consciousness has increased steadily over the past few years. Seems that the use of electricity as power source is gaining credibility and is settling on as the future substitute for fossil fuels for mobility, in favour of a cleaner and better world.

Therefore, motion control plays a leading role in the future development of this technology, due to the needs of an efficient drive with the best possible performance provided by a cost-effective solution, to make the most of the available energy. But electrical mobility is not only related to ground transportation, but all kinds of mobility, which have different issues and challenges.

1.2. Objective

The aim of this project is to confront the problems that an electrical drive can come across in an aircraft electrically propelled. In this case, the possibility of a failure of the position sensors is taken into account, and this thesis is intended to find a suitable solution for the BLDC control, whether the position sensors are functional or they break down. The implementation of the project encompasses the pin selection for the hardware and the software design for hall sensor mode or sensorless mode, as well as the detailed explanation for further test and modification by future contributors to the project.

1.3. BLDC Motor

1.3.1. History

The BLDC motor is developed on the basis of brushed DC motors. The actual electrical machine theory was established when Faraday discovered the electromagnetism induction phenomenon in 1831. The first DC motor was made in the 1840s. Confined by the development of power electronic devices and permanent magnet materials, BLDC motor was designed successfully more than one century later. In 1915, Langmuir invented the mercury rectifier to control grid electrode and made the DC/AC converter. Contrasting the disadvantages of traditional motors, in the 1930s some scholar started developing brushless motors in which electronic commutation was implemented, which made preparations for the BLDC motor. However, at that time, power electronic devices were still in the early stage of development, scholars could not find and appropriate commutation device.

In 1955, Harrison and Rye made the first patent claim for a thyristor commutator circuit to take the place of mechanical commutation equipment. This is exactly the rudiment of the BLDC motor. The principles of operation are as follows, when the rotor rotates, periodic Electro-Motive Force (EMF) is induced in the signal winding, which leads to the conduction of related thyristors. Hence, power windings feed by turns to achieve commutation. However, the problems are, first, when the rotor stops rotating, induced EMF cannot be produced in the signal windings and the thyristor is not biased, so the power winding cannot feed the current and this type of brushless motor has no starting torque. Furthermore, power consumption is large because the gradient of the electric potential's sloping part is small. To overcome these problems, researchers introduced the commutators with centrifugal plant or put an accessory steel magnet to ensure the motor started reliably. But the former solution is more complex, while the latter needs an additional starting pulse. After that, by numerous experiments and practices, the electronic commutation brushless motor was developed with the outcome of Hall elements in 1962, which inaugurated a new era in productionization of BLDC motors. In the 1970s, a magnet sensing diode, whose sensitivity is almost thousands of times greater than that of the Hall element, was used successfully for the control of BLDC motors. Later, as the electrical and electronics industry has been developing a large number of high-performance power semiconductors and permanent magnet materials, the solid ground for widespread use of BLDC motors is finally established.

In 1978, the Indramat branch of Mannesmann Corporation of the Federal Republic of Germany officially launched the MAC brushless DC motor and its drive system on Trade Shows in Hanover, which indicates that the BLDC motor had entered into the practical stage. Since them, worldwide further research has proceeded. Trapezoid-wave/square-wave and sine-wave BLDC motors were developed successively. The sine-wave brushless DC motor is the so-called Permanent Magnet Synchronous Machine (PMSM). With the development of permanent magnet materials, microelectronics, power electronics, detection techniques, automation and control technology, especially the power-switched devices like Insulated Gate Bipolar Transistor (IGBT), Integrated Gate Commutated Thyristor (IGCT) and so on, the BLDC motors in which electronic commutation is used are growing towars the intelligent, high-frequency and integrated directions.

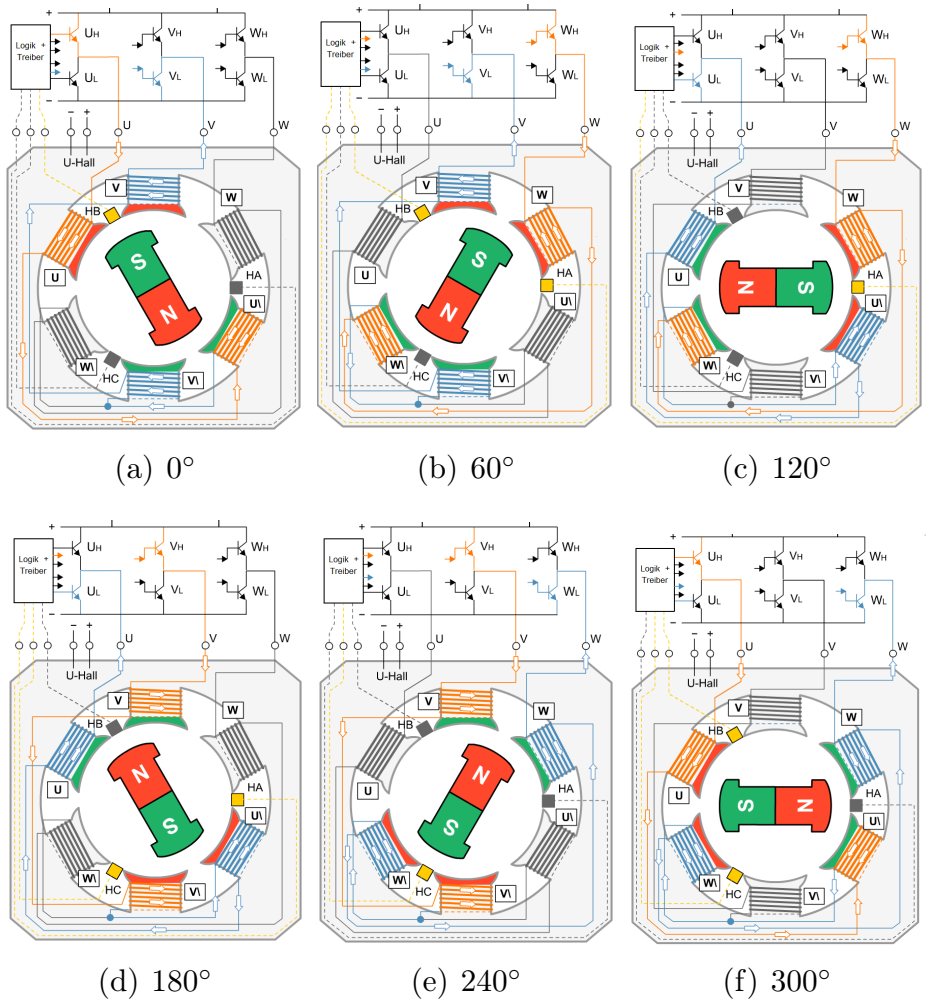
In the late 1990s, computer techniques and control theories developed rapidly. Microprocessors such as microcontroller units (CPU), digital signal processors (DSP), field programmable gate arrays (FPGA), complex programmable logic devices (CPLD) made unprecedented development while a qualitative leap was taken in instruction speed and storage space, which further promoted the evolution of BLDC motor. Moreover, a series of control strategies and methods, such as sliding-mode variable structure control and so on, are constantly used in BLDC motor drive systems. These methods can improve the performance on torque-ripple minimization, dynamic and steady-state speed response and system antidisturbance ability to some extent, as well as enlarge the application range and enrich the control theory [1].

1.3.2. Operation of the BLDCM

The BLDC motor unlike a DC motor, does not use brushes for commutation, instead, they are electronically commutated. They have many advantages over DC motors, some of them are [2]:

- High dynamic response.
- High efficiency.
- Long operating life.
- Lower operation noise.
- Higher speed range.

BLDC motors are synchronous motors, which means that the magnetic field generated by the stator and the magnetic field generated by the rotor rotate at the same frequency, so they do not experience *slip* as the induction motors. To rotate the BLDC motor, the stator windings have to be energized in a sequence, based on the rotor position. A full rotation period in hall sensor mode is shown in figure 1.1.



(a) 0° (b) 60° (c) 120°

(d) 180° (e) 240° (f) 300°

Figure 1.1.: Full rotation of a BLDC motor [3]

There are many ways to control a BLDC motor, each one with its advantages and disadvantages, the most used are:

- Field Oriented Control (FOC) (Sinusoidal)
- Hall sensor control (Trapezoidal)
- Sensorless control (Trapezoidal)

Field Oriented Control

FOC is a method which implies the measurement of motor currents and transform them into a rotating coordinate system synchronised with the rotor position, thus, to achieve that control the rotor position has to be directly measured or estimated and the Hardware (HW) peripherals used for the task must be suitable for an intensive computation compared to scalar controls, which means more expensive and a more complicated control. On the other side, it provides better dynamic responses and often more silent operations [4].

Hall effect sensors

These kinds of devices are based on Hall-effect theory, which states that if an electric current-carrying conductor is kept in a magnetic field, the magnetic field exerts a transverse force on the moving charge carriers that tends to push them to one side of the conductor. A build-up of charge at the sides of the conductors will balance this magnetic influence producing a measurable voltage between the two sides of the conductor. The presence of this measurable transverse voltage is called the Hall-effect because it was discovered by Edwin Hall in 1879.

Most BLDC motors have three Hall sensors inside the stator on the non-driving end of the motor. Whenever the rotor magnetic poles pass near the Hall sensors they give a high or low signal indicating the N or S pole is passing near the sensors as shown in figure 1.2 [5]. Based on the combination of these three Hall sensor signals, the exact sequence of commutation can be determined.

Embedding the Hall sensors into the stator is a complex process because any misalignment in these Hall sensors with respect to the rotor magnets will generate an error in determination of the rotor position. To simplify the

process of mounting the Hall sensors onto the stator some motors may have the Hall sensor magnets on the rotor, in addition to the main rotor magnets. Therefore, whenever the rotor rotates the Hall sensor magnets give the same effect as the main magnets. The Hall sensors are normally mounted on a printed circuit board and fixed to the enclosure cap on the non-driving end. This enables users to adjust the complete assembly of Hall sensors to align with the rotor magnets in order to achieve the best performance [6].

A BLDC motor is driven by voltage strokes coupled with the rotor position. These strokes must be properly applied to the active phases of the three-phase winding system so that the angle between the stator flux and the rotor flux is kept close to 90º to get the maximum generated torque. Therefore, the controller needs some means of determining the rotor's orientation/position, the Hall-effect sensors.

The process of switching the current to flow through only two phases for every 60 electrical degree rotation of the rotor is called electronic commutation. The motor is supplied from a three-phase inverter, and the switching actions can be simply triggered by the use of signals from position sensors that are mounted at appropriate points around the stator.

Such a drive usually also has a current loop to regulate the stator current, and an outer speed loop for speed control. The speed of the motor can be controlled if the voltage across the motor is changed, which can be achieved easily varying the duty cycle of the PWM signal used to control the six switches of the three-phase bridge. Only two inverter switches, one in the upper inverter bank and one in the lower inverter bank, are conducting at any instant. These discrete switching events ensure that the sequence of conducting pairs of stator terminals is maintained [7].

One of the hall sensors changes the state every 60 electrical degrees of rotation. Given this, it takes six steps to complete an electrical cycle. However, one electrical cycle may not correspond to a complete mechanical revolution of the rotor. The number of electrical cycles to be repeated to complete a mechanical rotation is determined by the rotor pole pairs. For each rotor pole pair, one electrical cycle is completed. The number of electrical cycles/rotations equals the rotor pole pairs [2]. This sequence of conducting pairs is essential to the production of a constant output torque.

In summary, permanent magnet motor drives require a rotor position sensor to properly perform phase commutation, but there are several drawbacks when such types of position sensors are used. The main drawbacks are the increased cost and size of the motor, and a special arrangement needs to be

made for mounting the sensors. Further, hall sensors are temperature sensitive and hence the operation of the motor is limited, which could reduce the system reliability because of the extra components and wiring [8]. To reduce cost and improve reliability such position sensors may be eliminated. To this end, many sensorless schemes have been reported for position (and speed) control of BLDC motors [9].

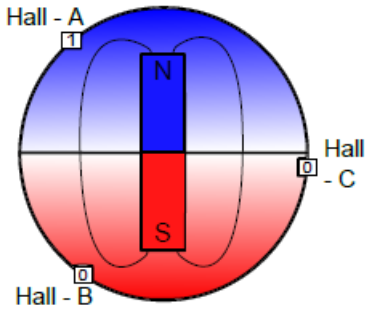


Figure 1.2.: Hall signal outputs depending on the magnet position

One of the advantages of this control is that, with only two phases conducting the motor can rotate and with the hall sensor signals the position and speed can be calculated, so the Microcontroller (μC) does not need a large computing capability as the ones used for FOC, reducing the final cost. If sensors are used for FOC, normally they are more expensive than hall effect sensors, but also more precise, such as incremental encoders or resolvers. The ease of control and the good performance/cost ratio makes the use of hall effect sensors one of the most used control techniques for BLDC motors.

Sensorless control

Problems with the cost and reliability of rotor position sensors have motivated research in the area of position sensorless BLDC machine drives. The sensorless control does not use any position sensor, it is based on the measurement of the non-conducting phase of the motor and the commutation time is calculated based on the Back Electro-Motive Force (BEMF). The BEMF in a BLDC motor is trapezoidal, as seen in figure 1.3, whereas in PMSM is sinusoidal. A BLDC motor can be driven with sinusoidal currents and a PMSM can be driven with direct currents, but for better performance, PMSM should be excited by sinusoidal currents and BLDC motors by direct currents.

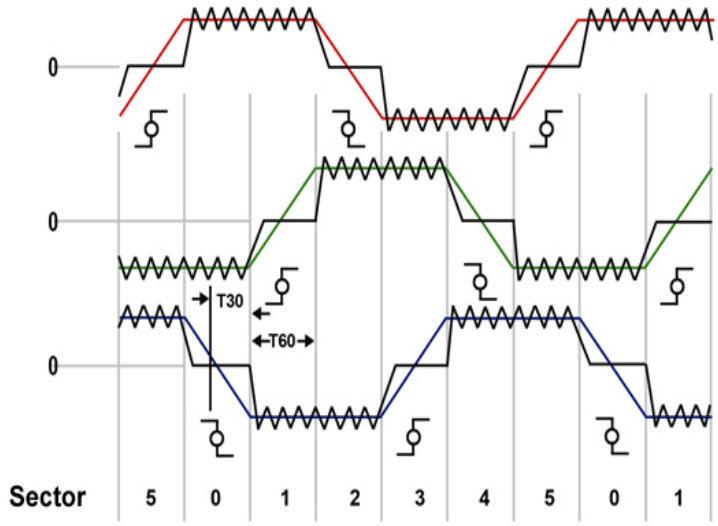


Figure 1.3.: Variation of the BEMF in the phases of a BLDC motor [12]

The BLDC motor provides an attractive candidate for sensorless operation because the nature of its excitation inherently offers a low-cost way to extract rotor position information from motor-terminal voltages. In the excitation of a three-phase BLDC motor, except for the phase-commutation periods, only two of the three phase windings are conducting at a time and the no conducting phase carries the back-EMF. There are many categories of sensorless control strategies [9]; however, the most popular category is based on back electromotive forces or BEMFs [10]. Sensing back-EMF of unused phase is the most cost efficient method to obtain the commutation sequence in star wound motors. Since back-EMF is zero at standstill and proportional to speed, the measured terminal voltage that has large signal-to-noise ratio cannot detect zero crossing at low speeds. That is the reason why in all BEMF-based sensorless methods the low-speed performance is limited, and an open-loop starting strategy is required [11].

BEMF sensing methods for BLDC motors are split in two categories: direct and indirect BEMF detection [13].

Direct BEMF detection methods The BEMF of floating phase is sensed and its zero crossing is detected by comparing it with neutral point voltage. This scheme suffers from high common node voltage and high frequency noise due to the Pulse-Width Modulation (PWM) drive, so it requires low pass filters and voltage dividers. This method includes:

- BEMF Zero-Crossing Point (ZCP) or Terminal Voltage Sensing
- PWM strategies

Indirect BEMF detection methods Because filtering introduces commutation delay at high speeds and attenuation causes reduction in signal sensitivity at low speeds, the range is narrowed in direct BEMF detection methods. In order to reduce switching noise, the indirect BEMF detection methods:

- BEMF Integration
- Third harmonic Voltage Integration
- Free-wheeling Diode Conduction or Terminal Current Sensing

The method used in the scope of this project is sampling the non-fed phase and compare this value to half of the DC-link voltage. The BEMF sampling is done during PWM ON statm allowing up to a maximum duty cycle (100%). The use of high frequency sampling without analog filtering and with the possibility of digital filtering gives more flexibility to this method.

The BEMF can be calculated as in equation 1.1:

$$E = 2NlrB\omega \tag{1.1}$$

where N is the number of winding turns per phase, l is the length of the rotor, r is the internal radius of the rotor, B is the rotor magnet flux density and ω is the motor's angular velocity.

The main objective in this sensorless control is to detect the ZCP. The ZCP is where the BEMF value is null, which occurs 30º after the commutation, thus, at half time between two commutations.

In figure 1.4 is depicted the motor terminal model, where L_x is the phase inductance, R is the phase resistance, E is the BEMF, V_n and V_x are, respectively, the star connection and phase voltages referenced to ground [14].

Then, the equation of the electrical model is as in equation 1.2:

$$V_x = RI_x + L\frac{dI_x}{dt} + E_x + V_n \tag{1.2}$$

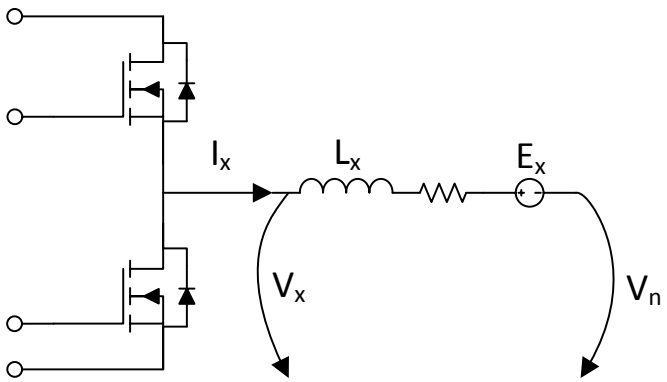


Figure 1.4.: Electrical model of a stator terminal

Considering C as the non-fed phase, which means $I_c = 0$, the equation for the three phases can be rewritten as:

$$V_a = RI_a + L\frac{dI_a}{dt} + E_a + V_n$$
$$V_b = RI_b + L\frac{dI_b}{dt} + E_b + V_n$$
$$V_c = E_c + V_n$$

As phase C is not conducting, the current flows only through phase A and B. Therefore:

$$I_a = -I_b$$

Including this into the above equations and adding the three terminal voltage equations, the result is:

$$V_a + V_b + V_c = E_a + E_b + E_c + 3V_n$$

As was depicted in figure 1.3, at the ZCP the sum of the three BEMF is null, then the above equation can be rewritten as:

$$V_a + V_b + V_c = +3V_n \tag{1.3}$$

At ZCP the BEMF of phase C is also 0, so its phase equation is equal to:

$$V_c = V_n \tag{1.4}$$

Inserting equation 1.4 in equation 1.3 and assuming that high-side Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET) of phase A and low-side MOSFET of phase B are closed, the final equation is:

$$V_{DC} + 0 + V_c = +3V_c$$

where V_c at the ZCP is finally:

$$V_c = \frac{V_{DC}}{2} \tag{1.5}$$

So, as seen in equation 1.5, the Software (SW) in the implemented sensorless control will detect the ZCP when the phase voltage is half of the DC link and proceed to calculate the time for the next commutation period. This process will be explained in detail in section 3.4.

2. Hardware and software tools

2.1. XE167F Easy Kit

The hardware used in this thesis is the Infineon Thechnologies' Easy Kit XE167F depicted in figure 2.1. This kit includes a μC from the XE166 family, the XE167F-96F66F and many peripherals such as an on board Universal Serial Bus (USB) to Join Test Action Group (JTAG)/Universal Asynchronous Receiver-Transmitter (UART) interface, a potentiometer, a push-button, Light-Emitting Diodes (LEDs) and the documentation and examples needed for a quick start [15].

Figure 2.1.: XE167 Easy Kit

2.1.1. XE167F-96F66F

The μC included in this Easy Kit is the XE167F-96F66F. This μC is a 144-pin device of the 16-bit architecture family XE166, which uses the C166v2 core. In figure 2.2 a classification of the family products is shown.

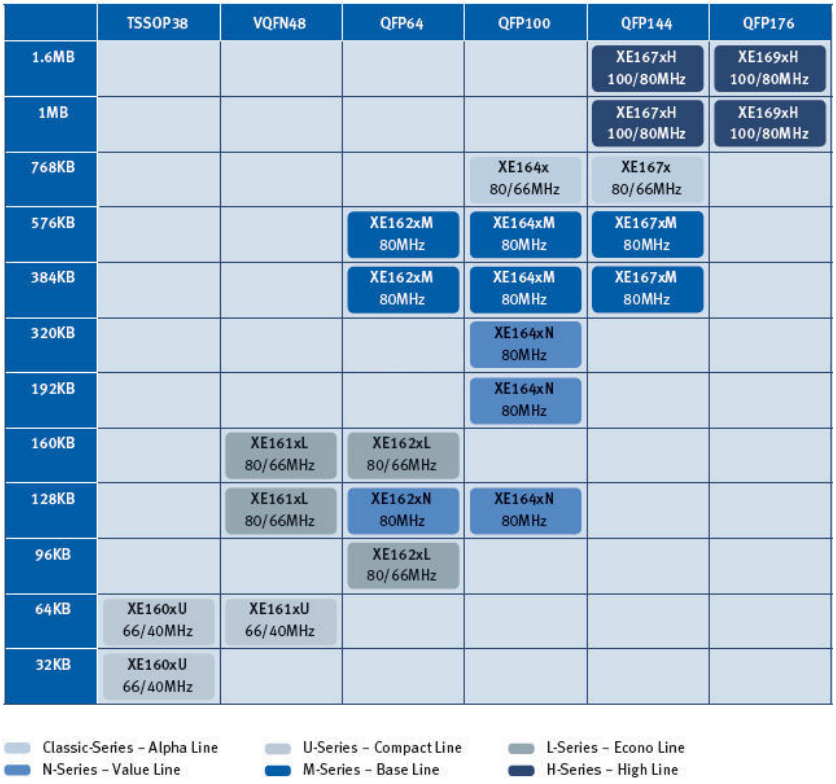


Figure 2.2.: Infineon Technologies' XE166 family classification

Some of the main features of this μC are [16]:

- Maximum clock frequency of 66 MHz
- One-cycle multiply-and-accumulate instructions with the Multiply-Accumulate unit (MAC)
- 64 kByte SRAM
- 768 kByte FLASH
- Two Synchronizable Analog-to-Digital Converter (ADC) with up to 24 channels, 10-bit resolution, conversion time below 1 μs, optional data preprocessing (data reduction, range check)

- Up to 4 Capture/Compare Unit 6 (CCU6)
- 5 General Purpose Timers (GPTs)
- C programmable
- 5 Controller Area Network (CAN) nodes
- 6 Serial channels
- Up to 118 General Purpose Input/Output (GPIO) lines
- Single power supply from 3.0 V to 5.5 V

2.1.2. The Easy Kit Board

The XE167F Easy Kit board includes all the necessary peripherals to perform a complete motion control chain. It includes two connectors BU101 and BU102 specially designed to act as the interface to a power board for motion control [15]. The Easy Kit board is shown in figure 2.3.

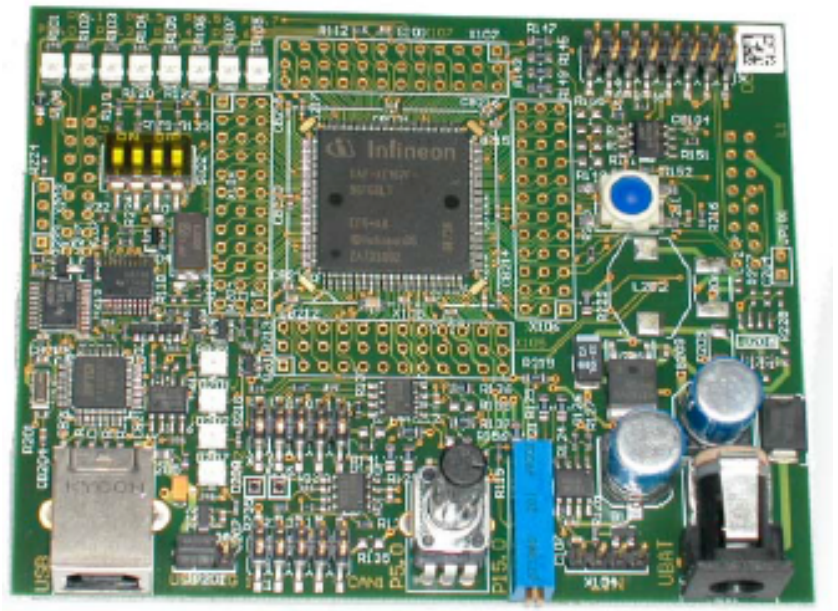


Figure 2.3.: XE167 Easy Kit board

To develop the software, it is necessary to first select the needed pins. The needs of the control are:

- 6 PWM signals
  - 3 for high-side MOSFETs
  - 3 for low-side MOSFETs
- 7 ADC channels
  - 3 for phase voltages
  - 1 for DC link voltage
  - 1 for DC link current
  - 2 for phase currents (for future FOC implementation)
- 3 pins for protection
  - 1 for overvoltage
  - 1 for overtemperature
  - 1 for a driver error latch
- 3 inputs for hall sensors

To select the most suitable pins of the μC, the before mentioned connectors BU101 and BU102 are used. The final pin selection, with their function and connector of the board is detailed in the table 2.1.

Purpose	**Pin**	**μC function**	**Connector**
PWM outputs	CC60 (10.0)	Low-side 1	BU101
	COUT60 (10.3)	High-side 1	
	CC61 (10.1)	Low-side 1	
	COUT61 (10.4)	High-side 2	
	CC62 (10.2)	Low-side 3	
	COUT62 (10.5)	High-side 3	
Driver error output	CTRAP (10.6)	_ERR	
Hall sensors	CCU60_CCPOS0 (10.7)	Hall 1	BU102
	CCU60_CCPOS1 (10.8)	Hall 2	
	CCU60_CCPOS2 (10.9)	Hall 3	
Currents	ADC0_CH3 (5.3)	I_{DC}	
	ADC0_CH8 (5.8)	I_a	
	ADC0_CH13 (5.13)	I_b	
Voltages	ADC1_CH1 (15.1)	V_a	X105A
	ADC1_CH4 (15.4)	V_b	
	ADC1_CH7 (15.7)	V_c	
	ADC1_CH2 (15.2)	V_{DC}	X105B
Overvoltage	GPIO (5.14)	Port input	X106A
Overtemperature	GPIO (5.11)	Port input	

Table 2.1.: Pin selection

Once the pins have been selected, the next step is to configure the peripherals of the μC to use this pins with their corresponding function.

2.2. DAvE™ version 2

Digital Application Virtual Engineer (DAVE™) is a free application with which the user can start developing the code for the μC. It provides a visual interface that makes it easier to configure the different peripherals and initialise the desired application, generating the needed C-code with the main driver functions and interrupts. The main screen of the XE167F-96F66F in the DAVE™ environment is depicted in figure 2.4. In this project, the second version of DAVE™ generation tools is used, because it is the one compatible with the XE166 family [17].

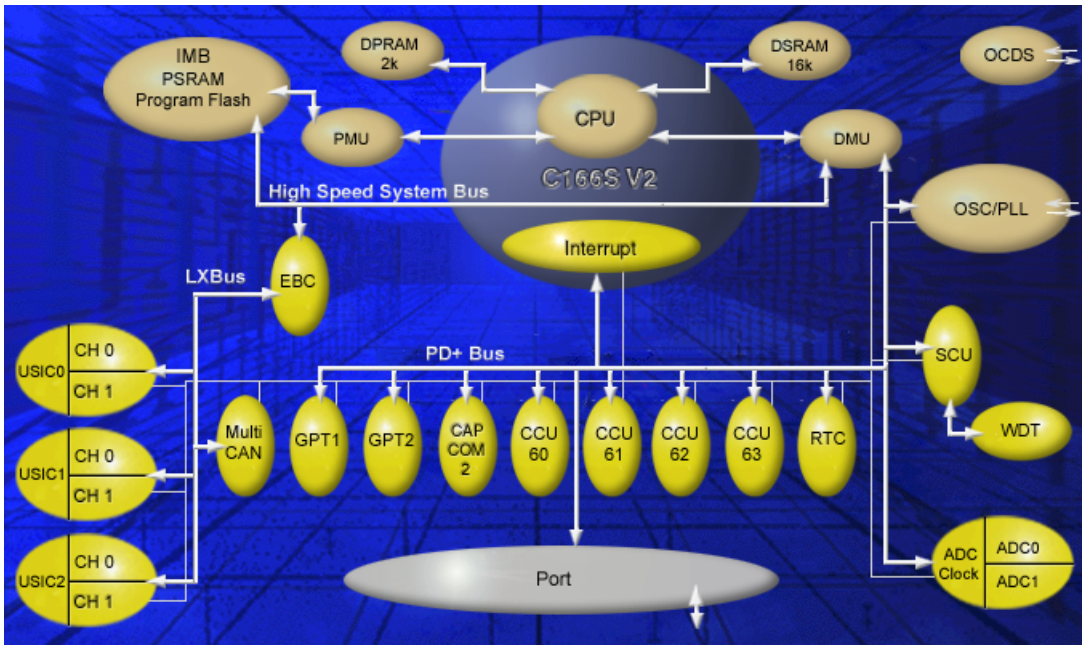


Figure 2.4.: DAVE main screen for XE167F

This programm will generate the main function and the peripheral initialisation as well as the interrupt functions, so there is only the need to create the application specific code inside the generated functions. For that purpose, the generated files include gaps where the user can introduce its own code, so that, if the DAVE™ project is modified, this code will stay intact. This gaps are implemented as in listing 2.1:

```
while(1) // Enter the endless loop.
 {
  // USER CODE BEGIN (Main,4)
  TRAP_RESET(); // Reset the software trap flag if requested by
     the user.
  MOTOR_OPERATION(); // Start or Stop the Motor as per the present
     conditions.
  RampUp(); // Ramp up(or down) the motor to the new speed.
  if(P5_IN & 0x4800) // If overvoltage or overtempreature
     detected...
  {
  status_overvoltemp = 1; // ... set the overtemperature and
     overvoltage flag.
  Motor_stop(); // ... stop the motor.
  }
  // USER CODE END
 }
```

Listing 2.1: User's code gap example

Once the peripherals are configured with the DAVE™ environment, the application code must be added and edited in the C compiler. Infineon products can be programmed in many development environments, but the most common used are Tasking and Keil.

2.3. Keil

To develop all the code and compile the software, the latest version of the Keil compiler (Keil μVision3) is used. This software includes the necessary files to program the μC and can directly import a DAVE™ project.

This is the last step of the tool chain, where finally the code is developed and loaded into the μC. The programming environment is depicted in figure 2.5.

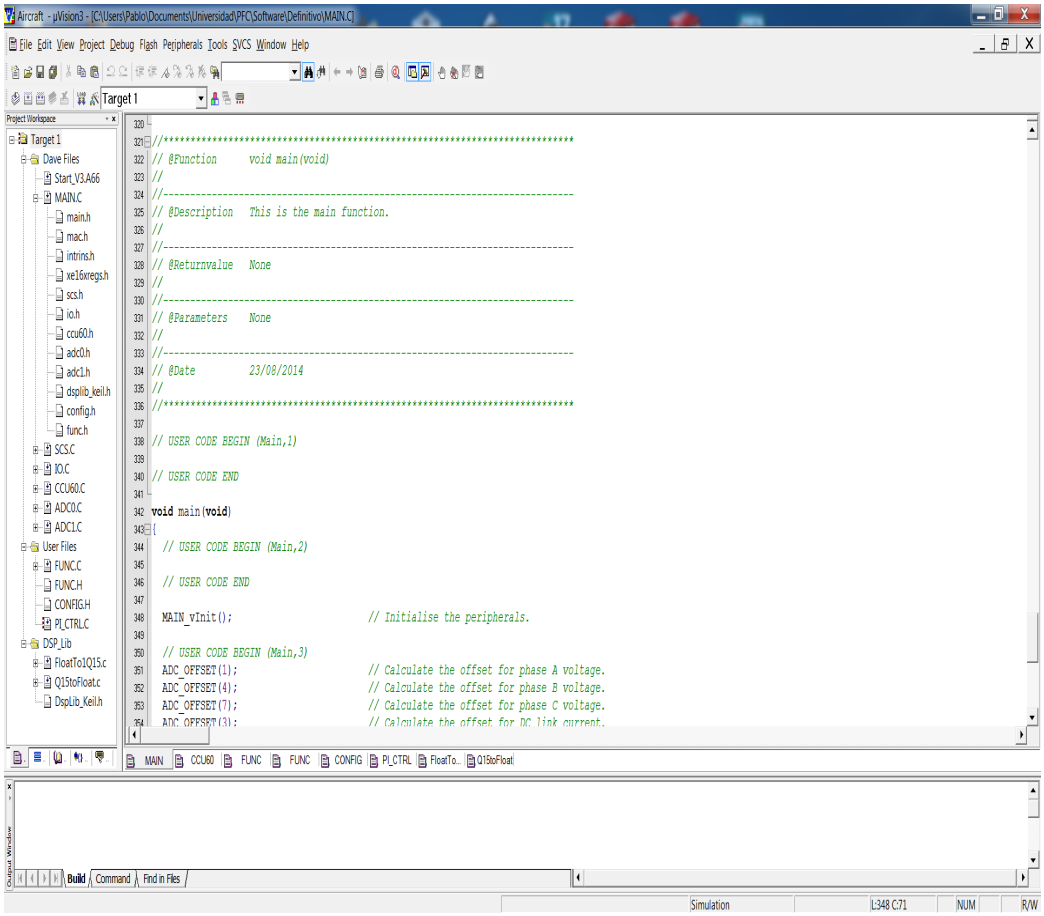


Figure 2.5.: Keil programming interface

3. Implementation

In this chapter are explained the main parts of the developed code, such as the use of the CCU6, the speed and current controllers, the two control modes, protection routines and the transition between control modes.

The most important peripherals used to develop the software are the ADCs and the CCU6, the ones that receive the signals to decide the commutation time and allow the protection interrupt routines and fast output responses.

3.1. The Capture/Compare Unit 6

The CCU6 is a high-resolution 16-bit capture and compare unit, specially designed for AC drive control. It supports block commutation control and control mechanisms for multi-phase machines. One of the main important features of this unit is the possibility to operate in hall sensor mode, directly configured with DAVE™ , which will configure the needed registers to capture the hall sensor signals and synchronise the commutation [18, 19].

The main blocks inside the CCU6 are the two timers (the T12 and the T13) and the Capture/Compare registers (CC6x).

The timer T12 is the one which executes the commutation of the outputs and is also used to measure the speed of the motor. In both control methods it operates as a timer, but the configuration of the compare registers is different between the hall sensor and sensorless method.

The CCU6 module can also enable or disable the modulation of the outputs, as well as choosing which timer can do the modulation and select which events generate an interrupt to have a full control of the process.

See in figure 3.1 how the inputs are configured in the DAVE™ environment.

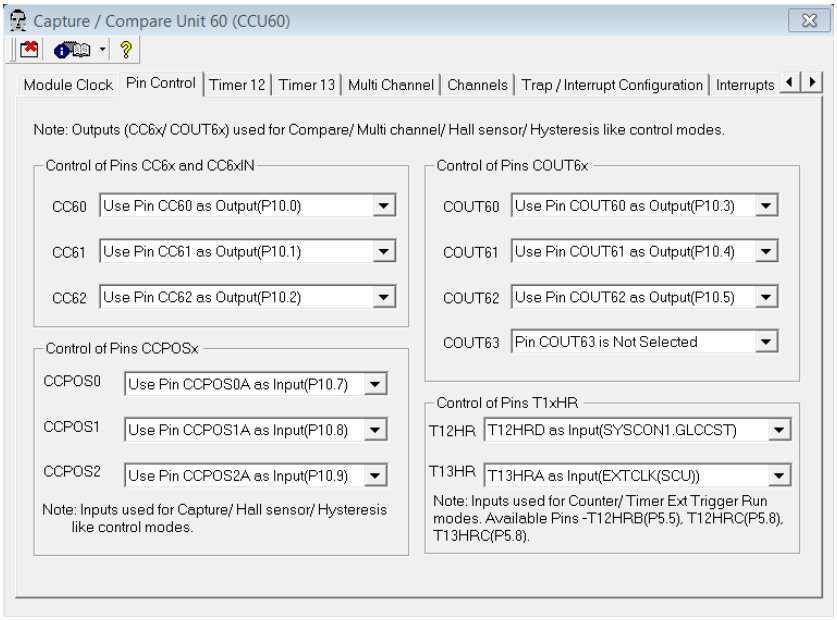


Figure 3.1.: Configuration of CCU6 channels

3.1.1. PWM generation

The control method in both hall sensor mode and sensorless mode is with unipolar PWM. This means that only the high-side MOSFET is modulated when it is enabled, whereas the low-side MOSFET stays closed the whole period. Unipolar switching mode reduces electromagnetic noise, DC bus ripple and commutation losses because there is less switching [20]. As most of the time the motor will run with hall sensors active, this method will be implemented as the primary control, while the sensorless control will only take part if the hall sensors fail.

The modulation is done with the T13, and the channels which are allowed to be modulated by the T13 are configured in the register *CCU60_MODCTR* shown in figure 3.2.

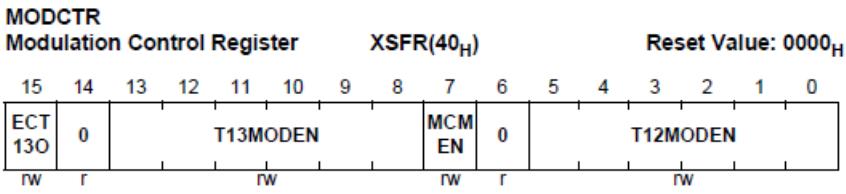


Figure 3.2.: Configuration of the CCU60_MODCTR register

The even bits of the T13MODEN correspond to the outputs CC6x, while the odd bits to the COUT6x. So the final configuration for the register is as in listing 3.1.

```
CCU60_MODCTR = 0x2A80; // load CCU60 modulation control register
```

Listing 3.1: Unipolar PWM configuration

The high-side MOSFET will be closed when the T13 value is over the compare register CCU60_CC63R, as shown in figure 3.3.

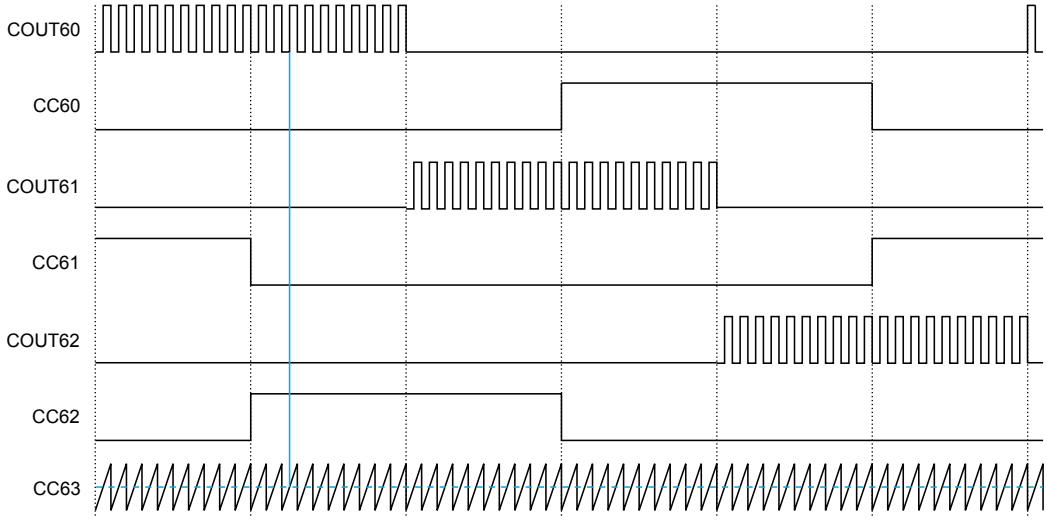


Figure 3.3.: Modulation of the output with T13

3.2. Speed and current controller

The speed and current controller are both implemented using the common used Proportional-Integral (PI) controller, but in a cascade control system. A cascade control is characterized by a master control loop (the outer and slower one) and a faster, slave, inner loop. The cascade control inner loop compensates for specific disturbances at source and largely prevents them from affecting the process being controlled, improving considerably the dynamic response. To implement a cascade control, the inner loop must have a faster response than the outer loop, at least 3-5 times faster [21]. This is permits that, in an electric motor, the speed can be the outer or primary loop and the current can be the inner or secondary loop, due to the fact that the

electrical time constant is normally much more smaller than the mechanical time constant. A block diagram of a cascade loop is depicted in figure 3.4.

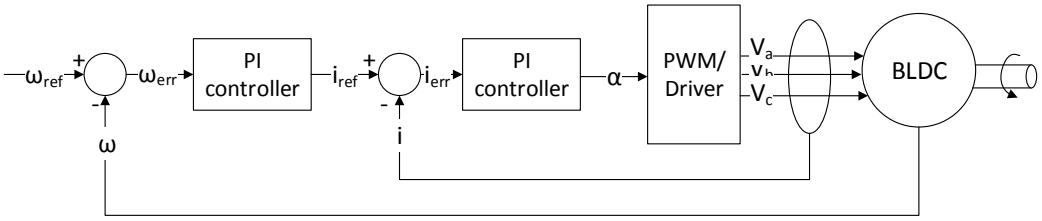


Figure 3.4.: Cascade control block diagram

The speed PI controller will execute every 1 ms, whereas the current PI every 50 μs, synchronised with the T13 period match.

3.2.1. MAC unit

The cascade control implementation in software uses the MAC unit of the μC. The MAC unit enhances strongly the DSP performance and 32 bit data processing capability. It is a built-in unit in the CPU core, which means that can be considered as an additional arithmetic unit dedicated for Digital Signal Processing (DSP) and 32 bit calculation. In the execution stage of the instruction processing pipeline, the instruction will be put into the ALU or MAC unit to be executed according to which instruction set (normal or MAC) the code comes from. If the instruction comes from normal instruction set, the code execution will be done in the ALU unit. If the instruction is from the MAC instruction set with CoXXX prefix, this instruction will be processed in the MAC unit [22].

Infineon and Keil provides a library with many mathematical functions which use this unit to reduce the calculation time in complex routines. In this project some of these functions are used, i.e. the *pi_controller64* function. This function calculates the output of a PI controller whose inputs and output are in 1Q15. The calculation is done in only 42 clock cycles, which means only 636,4 ns.

Listing 3.2 shows the code where the PI is implemented. See that is totally implemented in assembler and the commands used are referenced to the MAC unit.

```
int pi_controller64(long *pi_parameter,int reference,int actual)
{
long a;
#pragma asm

    mov     R12,MCW     ;Save MCW register
    mov     MCW,#1536   ;Set saturation and shift left
    mov     R11,ZEROS   ;Load zero in R11
    CoLOAD  R11,R9      ;Load Accumulator (High) with R9 (reference)
    CoSUB   R11,R10     ;error = reference - actual
    CoSTORE R9 ,MAS     ;Load error in R9
    mov     R3,[R8+]    ;
    CoLOAD  R3,[R8+]    ;Load yn (integral buffer) in accumulator
    mov     R1,R8       ;Save parameters addres in R1
    mov     R5,[R1+]    ;Load Kp (proportional Constant) in R5
    mov     R6,[R1+]    ;Load Ki = T0/Ti (integral Constant) in R6
    CoMAC   R6,R9       ;yn = Ki * error + yn
    mov     R6,[R1+]    ;Load ymax (limit value max)
    mov     R7,[R1+]    ;Load ymin (limit value min)
    CoMIN   R11,R6      ;Limit max yn
    CoMAX   R11,R7      ;Limit min yn
    CoSTORE R4,MAH      ;Store yn-high in R4
    CoSTORE R3,MAL      ;Store yn-low in R3
    mov     [-R8],R4    ;Store R4 in integral buffer(High)
    mov     [-R8],R3    ;Store R3 in integral buffer(Low)
    CoMUL   R5,R9       ;Kp * error
    CoSHL   #6          ;64 * Kp * error
    CoADD   R3,R4       ;y = yn + (64 * Kp * error)
    CoMIN   R11,R6      ;Limit max y
    CoMAX   R11,R7      ;Limit min y
    CoSTORE R4,MAS      ;Store y-high in R4 (return register)
    mov     MCW,R12     ;Restore MCW register
#pragma endasm
a = reference;
a = actual;
a = *pi_parameter;
}
```

Listing 3.2: PI implementation using MAC unit

3.2.2. Speed calculation

Speed reference calculation

In both control methods, the speed reference is not set directly, but constantly increased or decreased as a ramp. This makes that the controller follows the new vale avoiding oscillations and great current spikes due to big accelerations or decelerations. The *RampUp()* and *Speed_ref_ramp()* functions are the ones that compute this constant approach to the end reference value.

The function *RampUp()* will increment the counter every T13 period match, i.e. every 50 μs, and when the counter value is equal to the predefined time step, it will call the *Speed_ref_ramp()* and reset the counter. Inside the *Speed_ref_ramp()*, the speed reference used in the PI is incremented or decremented if it is not equal to the end reference (the final value to achieve).

The *RampUp* function also recalculates the time step necessary if the end reference is modified, in order to reach the end value in the predefined time stored in the variable *rampup_time*.

Motor speed calculation

The measured speed of the motor is calculated from different values of the timer T12. The function *Speed_calc* receives as input parameters a T12 sample corresponding to the number of counts of the timer between commutations (in hall sensor mode) or between ZCP (in sensorless mode), a counter value and a limit value, which will define if there are enough samples to start averaging the speed.

The main speed calculation is based on the use of a speed buffer which can accumulate up to 6*Pole Pairs samples. That is because every 6 sector a complete electrical revolution is done, the pole pairs indicate the number of electrical revolutions per mechanical revolution. Therefore, the speed buffer contains the amount of counts per mechanical revolution, which only needs to be converted to Revolutions Per Minute (RPM). To do that, the equation 3.1 is used.

$$n = \frac{60}{speed\_buff * T12_{resolution}} \tag{3.1}$$

However at the beginning there are not enough samples to implement the average of the speed, so only the sector time is used, and the equation to convert it to RPM is slightly different:

$$n = \frac{60}{sector\_time * 6 * Pp * T12_{resolution}}$$

3.3. Hall sensor control

The main control of the motor will be with the hall sensors active. The hall sensor signals are connected to pins 10.7, 10.8 and 10.9 as shown in table 2.1. In this mode, the timer T12 is configured as a timer, the registers CCU60_CC61 and CCU60_CC62 as compare registers, and the CCU60_CC60 as a capture register, which will read the hall sensors input. In table 3.1, the different registers and its function are listed.

The timer T12 clock frequency will be 1/256 the Central Processing Unit (CPU) clock frequency, whereas the T13 will have the same clock as the CPU. This makes the period of T13 to be short enough to modulate the output signals between commutation (f_{pwm} = 25 kHz), while the timer T12 can be used to measure the speed without overflowing too fast.

Register	**Configuration**	**Function**
CCU60_CC60	Capture mode	Detects an edge in any hall sensor signal
CCU60_CC61	Compare mode	Requests the new commutation if hall pattern is correct after a short time to avoid noise
CCU60_CC62	Compare mode	Will trigger the speed calculation after the commutation

Table 3.1.: CCU6 register function in hall sensor mode

A drawing explaining how it works is shown in figure 3.5. When the CCU60_CC60 detects an edge in the hall sensors inputs, it compares the read pattern with the actual and the expected pattern, and, if it matches the expected pattern, it will store the T12 value and reset the timer.

After a short time to avoid noise in the sensor signals, the commutation is requested when CCU60_CC61 compare register is matched and is synchronized at T13 one match. The speed calculation is done when T12 equals CCU60_CC62 compare register [23].

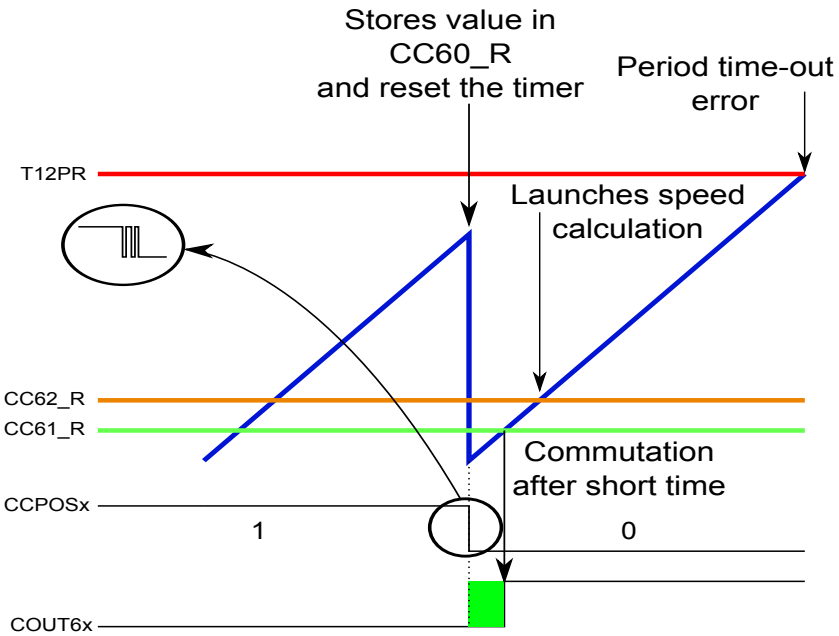


Figure 3.5.: T12 operation in Hall sensor mode

In the interrupt serviced by a CCU60_CC62R compare match, the speed is calculated based on the T12 counter between correct hall events and the next commutation pattern is loaded. The commutation pattern of the motor is listed in table 3.2.

Hall input	001	011	010	110	100	101
Clockwise	C+B-	A-C+	A-B+	C-B+	A+C-	A+B-
Counterclockwise	C-B+	A+C-	A+B-	C+B-	A-C+	A-B+

Table 3.2.: Hall sensor pattern and corresponding output

This values are stored in the variable *SP_Tab_r_l*. Each element of this variable consists of the current hall signal, the expected hall signal and the corresponding output. The order of the table is selected so, for an index equivalent at the actual hall sensor state, it returns the next commutation pattern. So a load of the next pattern is done as in listing 3.3.

```
CCU60_MCMOUTS = SP_Tab_r_l[((CCU60_MCMOUT & 0x3800) >> 11) +
    direction];   // ... load the next commutation pattern.
```

Listing 3.3: Load of the next commutation pattern

The variable *direction* provides an offset to choose between the clockwise or counterclokwise sequence.

3.4. Sensorless control

For the sensorless control we also use the T13 and T12 of the CCU6 with the same configuration. But the registers are not configured as in hall sensor mode, therefore, they are all in compare mode. Despite this change, the CCU60_CC61 and CCU60_CC62 have the same function as in hall sensor mode, in order to make the two control methods as much code-compatible as possible [24].

The difference in this method is that the speed is not calculated between commutations, but between ZCPs. Also the timer T12 is restarted after detecting a ZCP. As was deduced in equation 1.5, the ZCP is reached when the non-fed phase voltage equals half of the DC link voltage. When this it is detected, the software will recalculate the next commutation time and store it in CC60_CC61R, to request the commutation at the required moment. The CC60_CC62R, which triggers the interrupt for the speed calculation, has a value a little bit higher than CC60_CC61R, the same as in hall sensor mode. The T12PR is still used to trigger a time-out error interrupt if needed.

To detect the ZCP, many measures must be done during each sector. These measures can not be done randomly, because the measured value may not be always appropriate. To evaluate the BEMF properly, the voltage measurement must be done while the two phases are conducting, that is, after the compare match of T13. The performance in sensorless mode is illustrated in figure 3.6.

It is also important, that the BEMF measurement do not start just after the commutation, since the commutation spikes could lead to a false ZCP, this is achieved by the delay between CC60_CC61R and CC60_CC62R, making CC60_CC62R set the start of a slope 0 (rising) or 1 (falling). The code in 3.4 shows how inside the interrupt the corresponding slope, based on the previous ZCP detection, is set. More detailed explanation in appendix B.

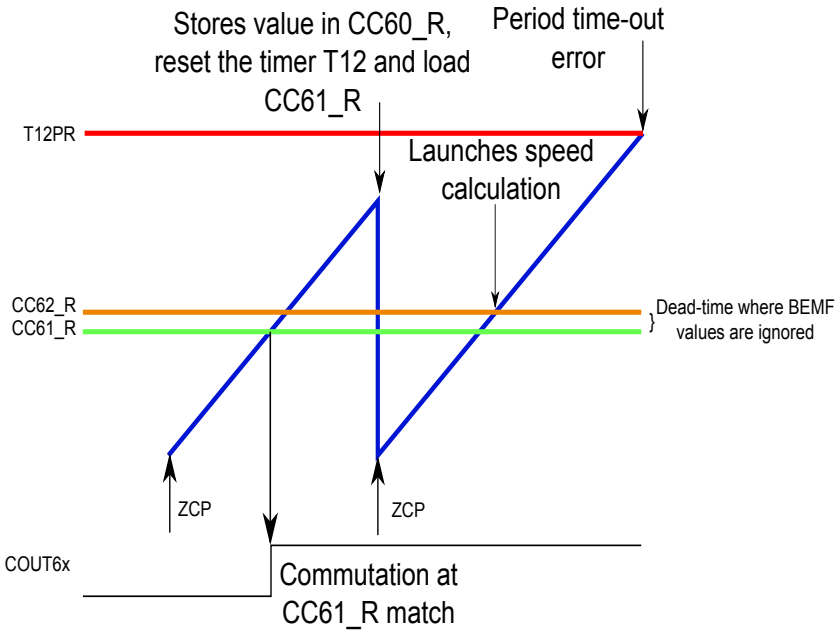


Figure 3.6.: T13 operation in Sensorless mode

```
if (slope == 2)       // If a falling edge has been detected...
{
   slope = 0;         // ... we need to look for rising edge.
}
if (slope == 3)       // If a rising edge has been detected...
{
   slope = 1;         // ... we need to look for falling edge.
}
```

Listing 3.4: Dead-time for BEMF detection set by CC60_CC62R

3.5. ADC measurements

As explained in section 3.4, the exact time of the measurements is really important for obtaining a reliable BEMF measure. Therefore, the measurements are synchronised with the T13 period or compare match depending on the signal to be measured. However, the measurement is not done at the exact moment of the match, since the sample will include the commutation spikes, but a short time after. Based on the ADC used by each measure and

the time requirements, the table 3.3 shows when each channel is converted. The figure 3.7 show the ADC conversions synchronised with the T13 during the switch pattern A+ B-, that is, measuring BEMF of phase C.

ADC Channel	Measurement moment
ADC0_CH3	T13 Compare match
ADC1_CH1	T13 Compare match (if non-fed)
ADC1_CH2	T13 Period match
ADC1_CH4	T13 Compare match (if non-fed)
ADC1_CH7	T13 Compare match (if non-fed)

Table 3.3.: ADC channel measurement moment

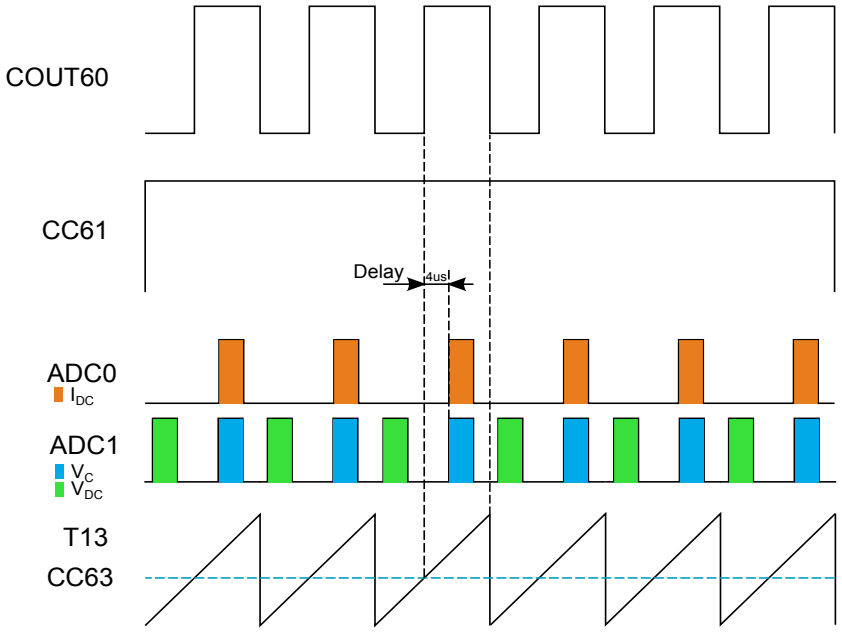


Figure 3.7.: Measurements synchronised with T13

As can be seen in figure 3.7, although the conversions of the phase voltages samples are synchronised with the T13 compare match, there is a delay between the compare event and the start of the conversion. This delay is done in order to avoid the noise detection caused by the commutation of the MOSFETs, as shown in figure 3.8, and to take into account the delay introduced by the optocoupler used in the hardware to isolate the ADC from the high voltage circuit.

This delay introduced by the optocoupler has been measured and is equal to 4μs, so it is necessary to add a delay by software, to select the appropri-

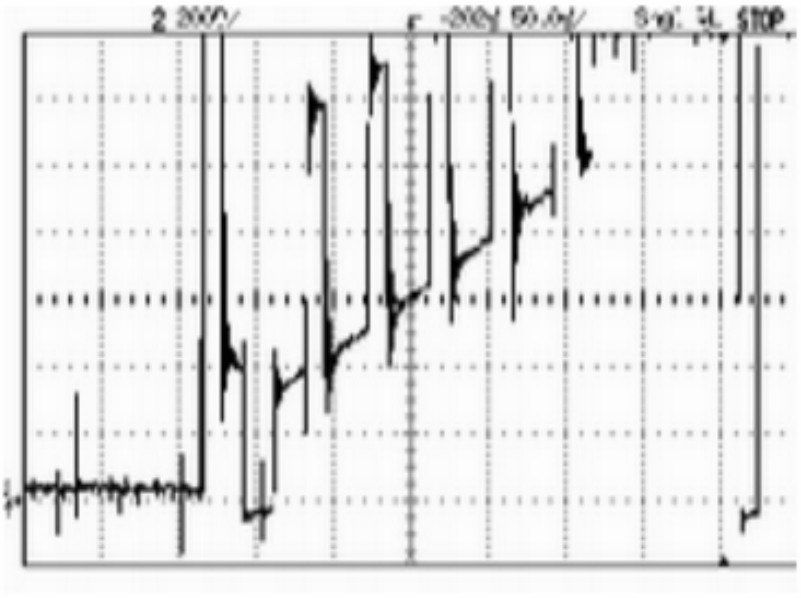


Figure 3.8.: Noise generated by the modulation of the conducting phases

ate time, the number of cycles used to service the interrupt and start the conversion must be taken into account:

- Cycles to service the interrupt = 7-11
- Cycles of the arbitration round of the ADC = 4
- Cycles to the conversion request = 6 cycles

The 4μs delay is equivalent to 264 clock cycles, which means that the delay function must last:

$$Forced\ delay = 264 - 11 - 6 - 4 = 243 cycles$$

To implement this delay this the function *_nop_()* is used, which is equal to 1 cycle delay. Taking into account the optimization level and the loop check, the final loop implemented is in the listing 3.5.

```
for(x=0; x<200; x++) {
 _nop_();                                                    // No
    operation function delay
}
```

Listing 3.5: Forced delay for a correct BEMF measurement

3.6. Protections

The protection routines are the most important, because they prevent the motor from being damaged. There are many different errors that can cause permanent damage to the motor, so the software must identify and prevent the damage by stopping the motor if any error signal is detected and register which fault caused the motor to stop. In table 3.4 are listed the errors and how are they handled by the software.

Error	**Handled by**
Overcurrent	ADC0_SRN0INT (Over specified limit)
Overvoltage	Main loop
Overtemperature	Main loop
Driver error	CCU60_IS_TRPF (TRAP)
Time-out	CCU60_IS_T12PM (T12 Period match)
Wrong Hall Event	CCU60_IS_WHE

Table 3.4.: Error signals

Most of the error signals will stop the motor in case they are latched, with the exception of the overcurrent and wrong hall event erros. The overcurrent interrupt will check the value of the measured current and, if its inside a specified limit, it will not stop the motor until a predefined amount of time is exceeded, avoiding a motor stop only if momentary current peaks occur.

The wrong hall event error occurs when the hall sensor signals fail, in this case, before stopping the motor, the software will try to change to sensorless control if possible. This error is the purpose of the thesis and is treated into detail in section 3.8.

3.7. Interrupts

The architecture of the XE167F supports several mechanisms for fast and flexible response to service requests from internal or external sources:

- Interrups generated by the Interrupt Controller (ITC)
- DMA transfers issued by the Peripheral Event Controller (PEC)
- Traps caused by the TRAP instruction or system specific states

The normal interrupt processing has a total of 96 separate interrupt nodes assignable to 16 priority levels subdivided in 8 groups, which allows the user to specify in which order multiple interrupt requests must be handled. During a normal interrupt process, the CPU temporarily suspends the current program execution and branches to an interrupt service routine to service the requested flag [25].

In the application, there are many interrupts that can be active at the same time. Therefore, the protection interrupts will be prioritised before the other requests. In figure 3.9 is depicted the priority of the software interrupts. The higher the interrupt level, the higher priority, and if two interrupts have the same priority level, the CPU will service first that with a higher group level.

	Group 0	Group 1	Group 2	Group 3
Level 15				
Level 14				
Level 13				
Level 12				
Level 11				
Level 10				
Level 9				
Level 8	CCU60 I1 INT			
Level 7	ADC INT 0			
Level 6			CCU60 I3 INT	
Level 5		CCU60 I2 INT		
Level 4				
Level 3				
Level 2				
Level 1				

Figure 3.9.: Interrupt priority

The selection of the interrupts and their routines is one of the most important parts of the application, due to the fact that being two different control methods, both need to work with the same interrupts and with as much code-sharing as possible to allow a fast transition from one control to the other and reduce the size of the final application.

3.8. Transition from Hall to Sensorless mode

The main objective of the thesis is developing a software capable of switching from a hall sensor control to a sensorless control if the hall sensors break stop working properly. To do this transition it is necessary to compare both control methods and their code to unify as much as possible the functions, interrupts and variables. Therefore, during the hall sensor control mode, the BEMF of the non-fed phase is measured, so that if the sensors fail, the control method can be switched. This entails that the size of the application is larger, as well as the CPU load, but on the other hand it can prevent the motor from stopping in case of error event.

As the failure of the sensors cannot be predicted, the software has to do the transition independently of when they occur. To achieve that, is necessary to evaluate the different situations when the hall sensors can stop working and proceed based on the moment of breaking.

The different situations when the hall sensors can fail are depicted in figure B.12.

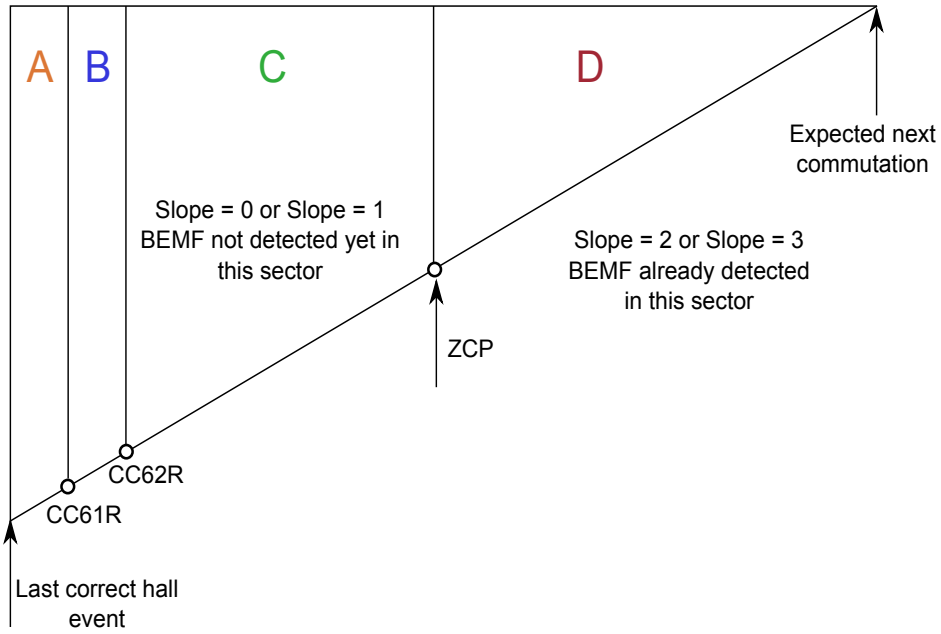


Figure 3.10.: Hall sensors fail situation

3.8.1. Situation A

This situation can occur if, after a correct hall input, the sensors stop working without having been done the commutation. If that happens, the software has to force the commutation, load the next pattern and reconfigure the timers and registers for the sensorless mode.

The reconfiguration of the registers is the same in all the possible situations, since it disables the hall sensor mode and enables all the registers as in compare mode. It also disables the wrong hall event interrupt.

To restart the timer, as the BEMF has not been detected yet, the timer T12 must be loaded with half of the sector time, which corresponds to the actual value of *zc*, since instead of measuring time between ZCP, as the timer is restarted every commutation, only stores half of the time between ZCP. Therefore, the timer T12 is restarted with the variable *zc* plus the actual T12 value.

3.8.2. Situation B

In situation B, the only difference is that the commutation has already been done, so there is no need to do it. The transition is done by reconfiguring the registers, loading the next pattern and restarting the timer T12 as in subsection 3.8.1. Neither in situation A nor in situation B the compare registers CCU60_61R and CCU60_62R need to be modified, given that as the BEMF has not been detected yet, the sensorless software control will change them when it is detected.

3.8.3. Situation C

This stretch is the last stretch where the BEMF has not been detected. Unlike the previous situations, the commutation and load of the next pattern has already been done, so the routine only have to reconfigure the registers and reload the timer T12 as in subsection 3.8.1.

3.8.4. Situation D

The last stretch corresponds to the region where the BEMF has already been detected, so in this case the compare registers must be reloaded to do the next commutation and the timer is reset to a different value than in the previous situation, due to the fact that now it will directly measure the time between ZCP. In this case, the timer T12 is reset with the actual value of T12 minus the actual *zc* value.

3.8.5. Software implementation of the transition

The implementation of the above situations is accomplished inside the wrong hall event interrupt routine. The listing 3.6 shows the necessary code to commplete a transition from hall sensor mode to sensorless mode.

```
if(CCU60_IS & 0x2000) // If CCU60_IS_WHE...
{  // ... wrong hall event detection.
// USER CODE BEGIN (NodeI1,18)
   status_hs_error=1; // ... set the hall sensor error status flag.
   if(status_bemf) // If bemf detection is possible...
   {
      bemf_aux = CCU60_T12; // ... save the actual T12 value.
     if(slope == 0 || slope == 1) // If bemf not detected yet in
        this sector...
      {
         CCU60_T12MSEL = 0x0111; // ... change hall sensor mode to
            sensorless mode.
         CCU60_IEN     &= 0x97FF; // ... disable wrong hall event or
            idle interrupts.
         CCU60_TCTR4   = 0x0045;  // ... T12 reset and stop.
         CCU60_T12     = zc_time + bemf_aux;    // ... set the
            counter time to the half of the sector time (actual
            zc_time).
         CCU60_T12PR   = (zc_time << 1) + zc_time; // ... set the
            time out limit to 1.5 of the sector time.
         CCU60_TCTR4   = 0x0040;  // ... shadow transfer request for
            T12.
       CCU60_TCTR4  = 0x0002; // ... start T12.
         if(bemf_aux < CCU60_CC61R) // If the commutation was not
            done yet...
         {
          CCU60_MCMOUTS |= 0x0080;  // ... force commutation.
```

```
        }
        if(bemf_aux < CCU60_CC62R) // If the update of the pattern
            was not done yet...
        {
         CCU60_MCMOUTS = CCU60_MCMOUTS = SP_Tab_r_l[((CCU60_MCMOUT
             & 0x3800) >> 11) + direction]; // ... load next
             commutation pattern.
        }
      }
      else    // If bemf has already been detected in this sector...
      {
         CCU60_T12MSEL = 0x0111;    // ... change hall sensor mode
            to sensorless mode.
         CCU60_IEN     &= 0x97FF;   // ... disable wrong hall event
            or idle interrupts.
         CCU60_TCTR4   = 0x0045;    // ... T12 reset and stop.
         CCU60_T12     = bemf_aux - zc_time;   // ... set the
            counter time to the difference between the actual value
            and half the sector time.
         CCU60_T12PR = (zc_time << 1) + zc_time;  // ... set the
            time out limit to 1.5 the sector time.
         CCU60_CC60SR  = (zc_time << 1);  // ... set the CC60R for
            speed measurement with the time between zero crossing
            points.
         CCU60_CC61SR  = zc_time - 5;  // ... initialize the CCU61
            register with half of the time between zero crossings,
         // so that it commutates within the required time (with an
            advance).
        CCU60_CC62SR   = CCU60_CC61SR + 10; // ... initialize CCU62
           some time after to calculate the speed.
         CCU60_TCTR4 = 0x0040; // ... shadow transfer request for
            T12.
         CCU60_TCTR4   = 0x0002;  // ... start T12.
      }
}
else    // If bemf detection is not possible...
{
Motor_stop(); // ... stop the motor.
}
// USER CODE END
CCU60_ISR |= 0x2000; // ... clear flag CCU60_IS_WHE.
}
```

Listing 3.6: Transition from hall sensor control to sensorless control

3.9. Software design

This section is intended to explain the basics of the software procedure, but for a detailed explanation and understanding of the whole process it is recommended to see appendix A and appendix B attached at the end of the thesis.

The control process includes the following steps:

1. System clock initialisation and configuration
2. Peripheral initialisation
    a) Input/Output ports
    b) CCU60 Unit
    c) ADCs
3. Register the offset error in the ADC measures
4. Initialise the start variables and the PIs variables
5. Enter the endless loop
6. Reset the trap flag to enable the motor start
7. Start the motor if there is no error or stop it if there is
    a) Set the start and end speed reference
    b) Initialise the control variables
    c) Initialise and enable the CCU60 timers and main registers
8. Increase the speed reference if it is not the desired reference yet
9. Check the overvoltage and overtemperature pins in the endless loop
10. The process is handled by the interrupts
    a) Protection interrupts
    b) CCU60_CC62R interrupt calculates the speed and loads the next pattern and applies a dead-time for the BEMF measurement after commutation
    c) T13 triggers measures, computes the duty cycle reference and modulates the outputs

References

[1] Chang-Liang Xia. *Permanent magnet brushless DC motor drives and controls*. 1 edition, June 2012.

[2] Padmaraja Yedamale. Brushless DC (BLDC) motor fundamentals. *Microchip Technology Inc*, page 20, 2003.

[3] Stepper and BLDC Motors Animation. `http://en.nanotec.com/support/tutorials/stepper-motor-and-bldc-motors-animation`.

[4] BLDC Vector Control (FOC). `http://www.st.com/web/catalog/apps/SE413/AS380/AC901`.

[5] Artur Giedymin. Entwurf und Implementierung einer Regelung eines hochdynamischen Elektroantriebs für eine adaptive Wankstabilisierung nach ISO 26262. Master's thesis, Technische Universität Berlin, October 2012.

[6] Ernesto Vázquez-Sánchez José Carlos Gamazo-Real and Jaime Gómez-Gil. Position and Speed Control of Brushless DC Motors Using Sensorless Techniques and Application Trends. *Sensors*, July 2010.

[7] Satish Rajagopalan. *Detection of Rotor and Load Faults in BLDC Motors Operating Under Stationary and Non-Stationary Conditions*. PhD thesis, Georgia Institute of Technology, Atlanta, GA, USA, August 2006.

[8] P Damodharan and Krishna Vasudevan. Indirect back-emf zero crossing detection for sensorless bldc motor operation. In *Power Electronics and Drives Systems, 2005. PEDS 2005. International Conference on*, volume 2, pages 1107–1111. IEEE, 2005.

[9] Gui-Jia Su and John W McKeever. Low-cost sensorless control of brushless dc motors with improved speed range. *Power Electronics, IEEE Transactions on*, 19(2):296–302, 2004.

[10] JX Shen and S Iwasaki. Sensorless control of ultrahigh-speed pm brush-

less motor using pll and third harmonic back emf. *Industrial Electronics, IEEE Transactions on*, 53(2):421–428, 2006.

[11] JX Shen, ZQ Zhu, and David Howe. Sensorless flux-weakening control of permanent-magnet brushless machines using third harmonic back emf. *IEEE Transactions on Industry Applications*, 40(6):1629–1636, 2004.

[12] Daniel Torres. Sensorless BLDC Motor Control with Back-EMF Filtering Using a Majority Function. *Digi-Key Article Library*, October 2011. Contributed By Convergence Promotions LLC.

[13] U. Vinatha, S. Pola, and K.P. Vittal. Recent developments in control schemes of bldc motors. In *Proceedings of the IEEE International Conference on Industrial Technology (ICIT 2006)*, Mumbai, India, December 2006.

[14] Bilal Akin and Manish Bhardwaj. *Sensorless Trapezoidal Control of BLDC Motors*. Texas Instruments Incorporated, 655303, Dallas, Texas 75265, July 2013. SPRABQ7.

[15] Infineon Technologies AG, 81726 Munich, Germany. *XE166 family Easy Kit Manual*, 2007-06 edition, June 2007.

[16] Infineon Technologies AG, 81726 Munich, Germany. *XE167 Datasheet*, 2008-08 edition, August 2008.

[17] Infineon DAvE information website. `http://www.infineon.com/cms/en/product/microcontrollers/development-tools,-software-and-kits/dave-tm-version-2-low-level-code-generation/channel.html?channel=db3a3043134dde6001134ee4d3b30265`.

[18] Infineon Technologies AG, 81726 Munich, Germany. *Technical details on CAPCOM6 module*, January 2001. AP1607310.

[19] Infineon Technologies AG, 81726 Munich, Germany. *XE166 Derivatives Volume 2: Peripheral Units*, 2008-08 edition, August 2008. User's Manual, V2.1.

[20] Eduardo Viramontes. *BLDC Motor Control with Hall Effect Sensors Using the 9S08MP*. Systems and Applications Engineering Freescale Technical Support, April 2010. AN4058.

[21] Peter Thomas. *How to tune cascade loops*. Control Specialists Ltd., expertune user conference edition, April 2007.

[22] Infineon Technologies AG, 81726 Munich, Germany. *DSP Optimization Guide for XC2000, XE1 6 6 and XC166 Microcontroller Families with MAC Unit*, 2007-10 edition, October 2007. APP1611311.

[23] Infineon Technologies AG, 81726 Munich, Germany. *Speed control of Brushless DC motor with Hall sensor using DAvE Drive for Infineon XC164CM/CS microcontrollers*, 2007-07-04 edition, July 2007. AP1611710.

[24] Infineon Technologies AG, 81726 Munich, Germany. *Sensorless Control of BLDC Motor using XE164F Microcontroller*, 2010-03 edition, March 20110. AP16173.

[25] Infineon Technologies AG, 81726 Munich, Germany. *XE166 Derivatives Volume 1: System Units*, 2008-08 edition, August 2008. User's Manual, V2.1.

Appendices

Appendix A.

Software structure and full code

The software is structured based on the generated files by DAVE™ project. The created files are stored in the folder *Dave files*, whereas the own created files are in the folder *User files*.

The header files of the generated DAVE™ files are not modified, on the other hand, the source files have been modified in the gaps provided for that purpose to develop the control application.

The user files are 3:

FUNC.H Includes all the *INLINE* functions included in the main file.

FUNC.C Contains the global variables and functions created by the user.

CONFIG.H Defines the motor and application parameters needed.

There are other files used, which are provided by Infineon and Keil, such as the DSP library files (DspLib_Keil.h, FloatTo1Q15.c and the PI function). The main code files are attached in the following pages:

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\MAIN.C

```
//*****************************************************************************
// @Module        Project Settings
// @Filename      MAIN.C
// @Project       Aircraft.dav
//-----------------------------------------------------------------------------
// @Controller    Infineon XE167F-96F66
//
// @Compiler      Keil
//
// @Codegenerator 2.2
//
// @Description   This file contains the project initialization function.
//
//-----------------------------------------------------------------------------
// @Date          26/08/2014 17:45:06
//
//*****************************************************************************

// USER CODE BEGIN (MAIN_General,1)

// USER CODE END



//*****************************************************************************
// @Project Includes
//*****************************************************************************

#include "MAIN.H"

// USER CODE BEGIN (MAIN_General,2)
#include "FUNC.H"
// USER CODE END


//*****************************************************************************
// @Macros
//*****************************************************************************

// USER CODE BEGIN (MAIN_General,3)

// USER CODE END


//*****************************************************************************
// @Defines
//*****************************************************************************

// USER CODE BEGIN (MAIN_General,4)

// USER CODE END


//*****************************************************************************
// @Typedefs
//*****************************************************************************

// USER CODE BEGIN (MAIN_General,5)

// USER CODE END


//*****************************************************************************
// @Imported Global Variables
//*****************************************************************************

// USER CODE BEGIN (MAIN_General,6)
   extern BIT status_overvoltemp;
```

Page: 1

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\MAIN.C

```
// USER CODE END


//*****************************************************************************
// @Global Variables
//*****************************************************************************

// USER CODE BEGIN (MAIN_General,7)

// USER CODE END


//*****************************************************************************
// @External Prototypes
//*****************************************************************************

// USER CODE BEGIN (MAIN_General,8)
   extern void ADC_OFFSET(int CH_OFFSET);
   extern void RampUp(void);
// USER CODE END


//*****************************************************************************
// @Prototypes Of Local Functions
//*****************************************************************************

// USER CODE BEGIN (MAIN_General,9)

// USER CODE END


//*****************************************************************************
// @Function      void MAIN_vInit(void)
//
//-----------------------------------------------------------------------------
// @Description   This function initializes the microcontroller.
//
//-----------------------------------------------------------------------------
// @Returnvalue   None
//
//-----------------------------------------------------------------------------
// @Parameters    None
//
//-----------------------------------------------------------------------------
// @Date          26/08/2014
//
//*****************************************************************************

// USER CODE BEGIN (Init,1)

// USER CODE END

void MAIN_vInit(void)
{
  // USER CODE BEGIN (Init,2)

  // USER CODE END

  //   globally disable interrupts
  PSW_IEN        =  0;


  ///  ------------------------------------------------------------------------
  ///  Configuration of the System Clock:
  ///  ------------------------------------------------------------------------
  ///  - VCO clock used, input clock is connected
  ///  - input frequency is 8,00 MHz
  ///  - system clock is 66.00 MHz
```

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\MAIN.C

```
  MAIN_vUnlockProtecReg();     // unlock write security

  MAIN_vChangeFreq();          // load PLL control register

  //   ---------------------------------------------------------------------
  //   Initialization of the Peripherals:
  //   ---------------------------------------------------------------------

  //   initializes the Parallel Ports
  IO_vInit();

  //   initializes the Capture / Compare Unit 60 (CCU60)
  CCU60_vInit();

  //   initializes the Analog / Digital Converter  (ADC0)
  ADC0_vInit();

  //   initializes the Analog / Digital Converter (ADC1)
  ADC1_vInit();


  //   ---------------------------------------------------------------------
  //   Initialization of the Bank Select registers:
  //   ---------------------------------------------------------------------


  // USER CODE BEGIN (Init,3)

  // USER CODE END

  MAIN_vLockProtecReg();       // lock write security

  //   globally enable interrupts
  PSW_IEN        =  1;

} //  End of function MAIN_vInit


//****************************************************************************
// @Function      void MAIN_vUnlockProtecReg(void)
//
//----------------------------------------------------------------------------
// @Description   This function makes it possible to write one protected
//                register.
//
//----------------------------------------------------------------------------
// @Returnvalue   None
//
//----------------------------------------------------------------------------
// @Parameters    None
//
//----------------------------------------------------------------------------
// @Date          26/08/2014
//
//****************************************************************************

// USER CODE BEGIN (UnlockProtecReg,1)

// USER CODE END

void MAIN_vUnlockProtecReg(void)
{
  uword uwPASSWORD;

    SCU_SLC = 0xAAAA;                  // command 0
    SCU_SLC = 0x5554;                  // command 1
```

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\MAIN.C

```
    uwPASSWORD = SCU_SLS & 0x00FF;
    uwPASSWORD = (~uwPASSWORD) & 0x00FF;

    SCU_SLC = 0x9600 | uwPASSWORD;      // command 2
    SCU_SLC = 0x0000;                   // command 3

} //  End of function MAIN_vUnlockProtecReg


//****************************************************************************
// @Function      void MAIN_vLockProtecReg(void)
//
//----------------------------------------------------------------------------
// @Description   This function makes it possible to lock one protected
//                register.
//
//----------------------------------------------------------------------------
// @Returnvalue   None
//
//----------------------------------------------------------------------------
// @Parameters    None
//
//----------------------------------------------------------------------------
// @Date          26/08/2014
//
//****************************************************************************

// USER CODE BEGIN (LockProtecReg,1)

// USER CODE END

void MAIN_vLockProtecReg(void)
{
  uword uwPASSWORD;

    SCU_SLC = 0xAAAA;                   // command 0
    SCU_SLC = 0x5554;                   // command 1

    uwPASSWORD = SCU_SLS & 0x00FF;
    uwPASSWORD = (~uwPASSWORD) & 0x00FF;

    SCU_SLC = 0x9600 | uwPASSWORD;      // command 2
    SCU_SLC = 0x1800;                   // command 3; new PASSWOR is 0x00

    uwPASSWORD = SCU_SLS & 0x00FF;
    uwPASSWORD = (~uwPASSWORD) & 0x00FF;
    SCU_SLC = 0x8E00 | uwPASSWORD;      // command 4

} //  End of function MAIN_vLockProtecReg


//****************************************************************************
// @Function      void MAIN_vChangeFreq(void)
//
//----------------------------------------------------------------------------
// @Description   This function is used to select the external crystal and
//                configure the system frequency to 80Mhz/66Mhz.
//
//----------------------------------------------------------------------------
// @Returnvalue   None
//
//----------------------------------------------------------------------------
// @Parameters    None
//
//----------------------------------------------------------------------------
// @Date          26/08/2014
//
//****************************************************************************
```

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\MAIN.C

```
// USER CODE BEGIN (ChangeFreq,1)

// USER CODE END

void MAIN_vChangeFreq(void)
{
  SCS_SwitchToHighPrecBandgap();

  //For application and internal application resets, the complete PLL configuration cou
ld be avoided
  //The entry from application resets and internal application reset is covered in the
following differentiation
  //in int/ext clock in lock/unlocked state.

  if  ((SCU_PLLSTAT & 0x0004) == 0x0004)             // fR derived from Internal clock
        {
            //Normal startup state during boot and the clock
            //has to be in the next step configured on the external crystal
            //use XTAL/VCO, count XTAL clock

            SCS_StartXtalOsc(1);                     // Starts the crystal oscillator
            SCS_SwitchSystemClock(1);                // System clock is increased to ta
rget speed (80/66 MHz)

        }

  else                                               // fR derived from external crysta
l clock
        {
            if ((SCU_PLLSTAT & 0x1009) == 0x1009)    // fR derived from external crysta
l clock + VCO is locked
            {
               //usually after an application reset where clock need not be configured
again.
               //check K2/P/N values and decide whether these values have to be adapted
 based on application needs.
               NOP();
               //usually the PLL losss of Lock TRAP schould be enabled here.
            }
            else                                     //fR derived from external crystal
 clock + VCO is not locked

            {
               //estimate the K1 value and the current frequency
               //reduce K2/P/N values in steps so that the frequency
               //jumps is limited to 20MHz or factor of 5 whichever is minimum
               NOP();
            }


        }

} //  End of function MAIN_vChangeFreq


//****************************************************************************
// @Function      void main(void)
//
//----------------------------------------------------------------------------
// @Description   This is the main function.
//
//----------------------------------------------------------------------------
// @Returnvalue   None
//
//----------------------------------------------------------------------------
// @Parameters    None
//
```

Page: 5

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\MAIN.C

```
//-----------------------------------------------------------------------------
// @Date          26/08/2014
//
//*****************************************************************************

// USER CODE BEGIN (Main,1)

// USER CODE END

void main(void)
{
  // USER CODE BEGIN (Main,2)

  // USER CODE END

  MAIN_vInit();

  // USER CODE BEGIN (Main,3)
  ADC_OFFSET(1);                        // Calculate the offset for phase A voltage.
  ADC_OFFSET(4);                        // Calculate the offset for phase B voltage.
  ADC_OFFSET(7);                        // Calculate the offset for phase C voltage.
  ADC_OFFSET(3);                        // Calculate the offset for DC link current.
  ADC_OFFSET(2);                        // Calculate the offset for DC link voltage.
  INIT_CTRL_VAR();                      // Initialise the main variables for start-up.
  // USER CODE END

  while(1)
  {

   // USER CODE BEGIN (Main,4)
    TRAP_RESET();                       // Reset the software trap flag if requested by
 the user.
    MOTOR_OPERATION();                  // Start or Stop the Motor as per the present c
onditions.
    RampUp();                           // Ramp up(or down) the motor to the new speed.
    if(P5_IN & 0x4800)                  // If overvoltage or overtempreature detected..
.
    {
    status_overvoltemp = 1;                 // ... set the overtemperature and overvolt
age flag.
    Motor_stop();                           // ... stop the motor.
    }
   // USER CODE END

  }

} //  End of function main


// USER CODE BEGIN (MAIN_General,10)

// USER CODE END
```

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\MAIN.H

```
//*****************************************************************************
// @Module        Project Settings
// @Filename      MAIN.H
// @Project       Aircraft.dav
//-----------------------------------------------------------------------------
// @Controller    Infineon XE167F-96F66
//
// @Compiler      Keil
//
// @Codegenerator 2.2
//
// @Description   This file contains all function prototypes and macros for
//                the MAIN module.
//
//-----------------------------------------------------------------------------
// @Date          26/08/2014 17:45:06
//
//*****************************************************************************

// USER CODE BEGIN (MAIN_Header,1)

// USER CODE END



#ifndef _MAIN_H_
#define _MAIN_H_

//*****************************************************************************
// @Project Includes
//*****************************************************************************

// USER CODE BEGIN (MAIN_Header,2)
#include <mac.h>
// USER CODE END


//*****************************************************************************
// @Macros
//*****************************************************************************

// USER CODE BEGIN (MAIN_Header,3)

// USER CODE END


//*****************************************************************************
// @Defines
//*****************************************************************************
#define KEIL

#define SEG(func) (unsigned int)(((unsigned long)((void (far*)(void))func) >> 16))
#define SOF(func) (unsigned int)(((void (far *) (void))func))

// USER CODE BEGIN (MAIN_Header,4)

#define  C_BDATA    int bdata
#define  C_SBIT     sbit
#define  BIT volatile bit
#define  C_SDATA    sdata

#define  INLINE     static __inline
#define  C_ATBIT(var, bit_pos)  =(var)^(bit_pos)

#define C_LSHIFT(var, pos) (var)
#define C_RSHIFT(var, pos) (var)

#define C_MAC(icode, var1, var2) _mac_((icode), (var1), (var2))
```

Page: 1

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\MAIN.H

```
#define C_MACL(icode, var) _macl_((icode), (var))

#define C_CoLOAD(var) C_MACL(CoLOAD, (var))
#define C_CoLOAD_0_1(var1, var2) C_MAC(CoLOAD, 0, (var2))
#define C_CoLOAD_1_0(var1, var2) C_MAC(CoLOAD, (var1), 0)

#define C_CoMAX(var) C_MACL(CoMAX, (var))
#define C_CoMAX_0_1(var1, var2) C_MAC(CoMAX, 0, (var2))
#define C_CoMAX_1_0(var1, var2) C_MAC(CoMAX, (var1), 0)

#define C_CoMIN(var) C_MACL(CoMIN, (var))
#define C_CoMIN_0_1(var1, var2) C_MAC(CoMIN, 0, (var2))
#define C_CoMIN_1_0(var1, var2) C_MAC(CoMIN, (var1), 0)

#define C_CoSUB(var) C_MACL(CoSUB, (var))
#define C_CoSUB_0_1(var1, var2) C_MAC(CoSUB, 0, (var2))
#define C_CoSUB_1_0(var1, var2) C_MAC(CoSUB, (var1), 0)

#define C_CoMULsu(var1, var2) _mac_(CoMULsu, (var1), (var2))
#define C_CoMULu(var1, var2) _mac_(CoMULu, (var1), (var2))
#define C_CoMUL(var1, var2) _mac_(CoMUL, (var1), (var2))
#define C_CoSTORE() _lmac_()
#define C_CoABS() _macv_(CoABSacc)
#define C_CoSTOREMAS() _imac_sat_()
#define C_CoSTOREMAS_MAL() _lmac_sat_()

#define C_CoMACsu(var1, var2) _mac_(CoMACsu, var1, var2)
// USER CODE END


//***************************************************************************
// @Typedefs
//***************************************************************************

// USER CODE BEGIN (MAIN_Header,5)
struct PI_cont                                      // PI-controller parameter

{
    long yn;                                            // PI integral buffer
    int  kp;                                            // Proportional constant

    int  ki;                                            // Integral constant
    int  ymax;                                          // Maximal ouput limit
    int  ymin;                                          // Minimal ouput limit
};
// USER CODE END


//***************************************************************************
// @Imported Global Variables
//***************************************************************************

// USER CODE BEGIN (MAIN_Header,6)

// USER CODE END


//***************************************************************************
// @Global Variables
//***************************************************************************

// USER CODE BEGIN (MAIN_Header,7)

// USER CODE END


//***************************************************************************
// @Prototypes Of Global Functions
```

Page: 2

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\MAIN.H

```
//*****************************************************************************

void MAIN_vUnlockProtecReg(void);

void MAIN_vLockProtecReg(void);

void MAIN_vChangeFreq(void);

// USER CODE BEGIN (MAIN_Header,8)

// USER CODE END


//*****************************************************************************
// @Interrupt Vectors
//*****************************************************************************

// USER CODE BEGIN (MAIN_Header,9)

// USER CODE END


//*****************************************************************************
// @Project Includes
//*****************************************************************************

#include <Intrins.h>

#include  "XE16xREGS.H"
#include  "SCS.H"

#include  "IO.H"
#include  "CCU60.H"
#include  "ADC0.H"
#include  "ADC1.H"

// USER CODE BEGIN (MAIN_Header,10)
#include "DspLib_Keil.h"
#include "CONFIG.H"
// USER CODE END


#endif  // ifndef _MAIN_H_
```

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\ADC0.C

```
//*****************************************************************************
// @Module        Analog / Digital Converter  (ADC0)
// @Filename      ADC0.C
// @Project       Aircraft.dav
//-----------------------------------------------------------------------------
// @Controller    Infineon XE167F-96F66
//
// @Compiler      Keil
//
// @Codegenerator 2.2
//
// @Description   This file contains functions that use the ADC0 module.
//
//-----------------------------------------------------------------------------
// @Date          26/08/2014 17:45:07
//
//*****************************************************************************

// USER CODE BEGIN (ADC0_General,1)

// USER CODE END



//*****************************************************************************
// @Project Includes
//*****************************************************************************

#include "MAIN.H"

// USER CODE BEGIN (ADC0_General,2)

// USER CODE END


//*****************************************************************************
// @Macros
//*****************************************************************************

// USER CODE BEGIN (ADC0_General,3)

// USER CODE END


//*****************************************************************************
// @Defines
//*****************************************************************************

// USER CODE BEGIN (ADC0_General,4)

// USER CODE END


//*****************************************************************************
// @Typedefs
//*****************************************************************************

// USER CODE BEGIN (ADC0_General,5)

// USER CODE END


//*****************************************************************************
// @Imported Global Variables
//*****************************************************************************

// USER CODE BEGIN (ADC0_General,6)
   extern BIT status_overcurrent;              // Overcurrent detection
```

Page: 1

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\ADC0.C

```
   extern int C_SDATA oc_counter;                    // Counter to specify the time limi
t for motor Overcurrent
// USER CODE END


//*****************************************************************************
// @Global Variables
//*****************************************************************************

// USER CODE BEGIN (ADC0_General,7)

// USER CODE END


//*****************************************************************************
// @External Prototypes
//*****************************************************************************

// USER CODE BEGIN (ADC0_General,8)
   extern void Motor_stop(void);
// USER CODE END


//*****************************************************************************
// @Prototypes Of Local Functions
//*****************************************************************************

// USER CODE BEGIN (ADC0_General,9)

// USER CODE END


//*****************************************************************************
// @Function      void ADC0_vInit(void)
//
//-----------------------------------------------------------------------------
// @Description   This is the initialization function of the ADC function
//                library. It is assumed that the SFRs used by this library
//                are in reset state.
//
//                Following SFR fields will be initialized:
//                GLOBCTR  - Global Control
//                RSPR0    - Priority and Arbitration Register
//                ASENR    - Arbitration slot enable register
//                CHCTRx   - Channel Control Register x
//                RCRx     - Result Control Register x
//                KSCFG    - Module configuration Register
//                INPCR    - Input class Registers
//                CHINPRx  - Channel Interrupt register
//                EVINPRx  - Event Interrupt register
//                SYNCTR   - Synchronisation control register
//                LCBRx    - Limit check boundary register
//                PISEL    - Port input selection
//                QMR0     - Sequential 0 mode register
//                CRMR1    - Parallel mode register
//                QMR2     - Sequential 2 mode register
//
//-----------------------------------------------------------------------------
// @Returnvalue   None
//
//-----------------------------------------------------------------------------
// @Parameters    None
//
//-----------------------------------------------------------------------------
// @Date          26/08/2014
//
//*****************************************************************************
```

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\ADC0.C

```
// USER CODE BEGIN (ADC0_Init,1)

// USER CODE END

void ADC0_vInit(void)
{
  // USER CODE BEGIN (ADC0_Init,2)

  // USER CODE END

  ///  -----------------------------------------------------------------------
  ///  Configuration of ADC0 kernel configuration register:
  ///  -----------------------------------------------------------------------
  ADC0_KSCFG     =  0x0003;      // load ADC0 kernel configuration register

  ///  - the ADC module clock is enabled
  ///  - the ADC module clock = 66,00 MHz
  ///

  _nop_();  // one cycle delay

  _nop_();  // one cycle delay

  ///  -----------------------------------------------------------------------
  ///  Configure global control register:
  ///  -----------------------------------------------------------------------
  ///  --- Conversion Timing -----------------
  ///  - conversion time (CTC)    = 01,29 us

  ///  _Analog clock is 1/5th of module clock and digital clock is 1/1 times
  ///  of module clock

  ///  - the permanent arbitration mode is selected
  ADC0_GLOBCTR   =  0x0004;      // load global control register

  ///  -----------------------------------------------------------------------
  ///  Configuration of Arbitration Slot enable register and also the Source
  ///  Priority register:
  ///  -----------------------------------------------------------------------
  ///  - Arbitration Slot 0 is disabled

  ///  - Arbitration Slot 1 is enabled

  ///  - Arbitration Slot 2 is disabled

  ///  - the priority of request source 0 is low
  ///  - the wait-for-start mode is selected for source 0
  ///  - the priority of request source 1 is high
  ///  - the wait-for-start mode is selected for source 1
  ///  - the priority of request source 2 is low
  ///  - the wait-for-start mode is selected for source 2
  ADC0_ASENR     =  0x0002;      // load Arbitration Slot enable register

  ADC0_RSPR0     =  0x0032;      // load Priority and Arbitration register

  ///  -----------------------------------------------------------------------
  ///  Configuration of Channel Control Registers:
  ///  -----------------------------------------------------------------------
  ///  Configuration of Channel 3
  ///  - the result register0 is selected
  ///  - the limit check 7 is selected

  ///  - the reference voltage selected is Standard Voltage (Varef)

  ///  - the input class selected is Input Class 0

  ///  - LCBR0 is selected as upper boundary
```

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\ADC0.C

```
///  - LCBR3 is selected as lower boundary

ADC0_CHCTR3    =  0x007C;      // load channel control register

///  ------------------------------------------------------------------------
///  Configuration of Sample Time and Resolution:
///  ------------------------------------------------------------------------

///  Sample phase is extended by 48 Analog clock cycles

///  10 bit resolution selected

ADC0_INPCR0    =  0x0030;      // load input class0 register

///  10 bit resolution selected

ADC0_INPCR1    =  0x0000;      // load input class1 register

///  ------------------------------------------------------------------------
///  Configuration of Result Control Registers:
///  ------------------------------------------------------------------------
///  Configuration of Result Control Register 0
///  - the data reduction filter is disabled
///  - the event interrupt is disabled
///  - the wait-for-read mode is disabled

///  - the FIFO functionality is disabled

ADC0_RCR0      =  0x0000;      // load result control register 0

///  Configuration of Result Control Register 1
///  - the data reduction filter is disabled
///  - the event interrupt is disabled
///  - the wait-for-read mode is disabled

///  - the FIFO functionality is disabled

ADC0_RCR1      =  0x0000;      // load result control register 1

///  Configuration of Result Control Register 2
///  - the data reduction filter is disabled
///  - the event interrupt is disabled
///  - the wait-for-read mode is disabled

///  - the FIFO functionality is disabled

ADC0_RCR2      =  0x0000;      // load result control register 2

///  Configuration of Result Control Register 3
///  - the data reduction filter is disabled
///  - the event interrupt is disabled
///  - the wait-for-read mode is disabled

///  - the FIFO functionality is disabled

ADC0_RCR3      =  0x0000;      // load result control register 3

///  Configuration of Result Control Register 4
///  - the data reduction filter is disabled
///  - the event interrupt is disabled
///  - the wait-for-read mode is disabled

///  - the FIFO functionality is disabled

ADC0_RCR4      =  0x0000;      // load result control register 4

///  Configuration of Result Control Register 5
///  - the data reduction filter is disabled
```

Page: 4

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\ADC0.C

```
  ///  - the event interrupt is disabled
  ///  - the wait-for-read mode is disabled

  ///  - the FIFO functionality is disabled

  ADC0_RCR5      =  0x0000;      // load result control register 5

  ///  Configuration of Result Control Register 6
  ///  - the data reduction filter is disabled
  ///  - the event interrupt is disabled
  ///  - the wait-for-read mode is disabled

  ///  - the FIFO functionality is disabled

  ADC0_RCR6      =  0x0000;      // load result control register 6

  ///  Configuration of Result Control Register 7
  ///  - the data reduction filter is disabled
  ///  - the event interrupt is disabled
  ///  - the wait-for-read mode is disabled

  ///  - the FIFO functionality is disabled

  ADC0_RCR7      =  0x0000;      // load result control register 7

  ///  ---------------------------------------------------------------------
  ///  Configuration of Channel Interrupt Node Pointer Register:
  ///  ---------------------------------------------------------------------
  ///  - the SR0 line become activated if channel 3 interrupt is generated

  ADC0_CHINPR0   =  0x0000;      // load channel interrupt node pointer
                                 // register

  ADC0_CHINPR4   =  0x0000;      // load channel interrupt node pointer
                                 // register

  ADC0_CHINPR8   =  0x0000;      // load channel interrupt node pointer
                                 // register

  ADC0_CHINPR12  =  0x0000;      // load channel interrupt node pointer
                                 // register

  ///  ---------------------------------------------------------------------
  ///  Configuration of Event Interrupt Node Pointer Register for Source
  ///  Interrupts:
  ///  ---------------------------------------------------------------------
  ///  - the SR 0 line become activated if the event 1 interrupt is generated

  ADC0_EVINPR0   =  0x0000;      // load event interrupt set flag register

  ///  ---------------------------------------------------------------------
  ///  Configuration of Event Interrupt Node Pointer Register for Result
  ///  Interrupts:
  ///  ---------------------------------------------------------------------

  ADC0_EVINPR8   =  0x0000;      // load event interrupt set flag register


  ADC0_EVINPR12  =  0x0000;      // load event interrupt set flag register

  ///  ---------------------------------------------------------------------
  ///  Configuration of Service Request Nodes 0 - 3 :
  ///  ---------------------------------------------------------------------
  ///  SRN0 service request node configuration:
  ///  - SRN0 interrupt priority level (ILVL) = 7
  ///  - SRN0 interrupt group level (GLVL) = 0
  ///  - SRN0 group priority extension (GPX) = 0
```

Page: 5

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\ADC0.C

```
  ADC_0IC        =  0x005C;


  ///  ------------------------------------------------------------------------
  ///  Configuration of Limit Check Boundary:
  ///  ------------------------------------------------------------------------

  ADC0_LCBR0     =  0x0198;      // load limit check boundary register 0

  ADC0_LCBR1     =  0x0E64;      // load limit check boundary register 1

  ADC0_LCBR2     =  0x0540;      // load limit check boundary register 2

  ADC0_LCBR3     =  0x07C8;      // load limit check boundary register 3

  ///  ------------------------------------------------------------------------
  ///  Configuration of Gating source and External Trigger Control:
  ///  ------------------------------------------------------------------------
  ///  - No Gating source selected for Arbitration Source 0

  ///  - the trigger input ETR00 is selected for Source 0

  ///  - No Gating source selected for Arbitration Source 1

  ///  - the trigger input ETR00 is selected for Source 1

  ///  - No Gating source selected for Arbitration Source 2

  ///  - the trigger input ETR00 is selected for Source 1

  ADC0_PISEL     =  0x0444;      // load external trigger control register

  ///  ------------------------------------------------------------------------
  ///  Configuration of Conversion Queue Mode Register:Sequential Source 0
  ///  ------------------------------------------------------------------------
  ///  - the gating line is permanently Disabled
  ///  - the external trigger is disabled
  ///  - the trigger mode 0 is selected

  ADC0_QMR0      =  0x0000;      // load queue mode register

  ///  ------------------------------------------------------------------------
  ///  Configuration of Conversion Queue Mode Register:Sequential Source 2
  ///  ------------------------------------------------------------------------
  ///  - the gating line is permanently Disabled
  ///  - the external trigger is disabled
  ///  - the trigger mode 0 is selected

  ADC0_QMR2      =  0x0000;      // load queue mode register

  ///  ------------------------------------------------------------------------
  ///  Configuration of Conversion Request Mode Registers:Parallel Source
  ///  ------------------------------------------------------------------------
  ///  - the gating line is permanently Enabled
  ///  - the external trigger is disabled
  ///  - the source interrupt is disabled
  ///  - the autoscan functionality is disabled

  ADC0_CRMR1     =  0x0001;      // load conversion request mode register 1

  ///  ------------------------------------------------------------------------
  ///  Configuration of Synchronisation Registers:
  ///  ------------------------------------------------------------------------
  ///  - ADC0 is master
  ADC0_SYNCTR   |=  0x0010;      // Synchronisation register

  P5_DIDIS       =  0x0008;      // Port 5 Digital input disable register
```

Page: 6

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\ADC0.C

```
  ADC0_GLOBCTR  |=  0x0300;      // turn on Analog part


  // USER CODE BEGIN (ADC0_Init,3)

  // USER CODE END

} //  End of function ADC0_vInit


//*****************************************************************************
// @Function      void ADC0_viSRN0(void)
//
//-----------------------------------------------------------------------------
// @Description   This is the interrupt service routine for the Service
//                Request Node 0 of the ADC0 module.
//
//-----------------------------------------------------------------------------
// @Returnvalue   None
//
//-----------------------------------------------------------------------------
// @Parameters    None
//
//-----------------------------------------------------------------------------
// @Date          26/08/2014
//
//*****************************************************************************

// USER CODE BEGIN (ADC0_viSRN0,0)

// USER CODE END

void ADC0_viSRN0(void) interrupt ADC0_SRN0INT
{

    if((ADC0_CHINFR & 0x0008) ==  0x0008)    //Channel3 interrupt
    {
        ADC0_CHINCR = 0x0008;     // Clear Channel3 interrupt

      // USER CODE BEGIN (ADC0_viSRN0,7)
       if (((ADC1_RESR0>>2)&0x03FF) < 682)                          // motor instantane
ous current below shutdown
        {                                                               // current limi
t ?
            oc_counter++;                                               // if no, incre
ment s/w counter.
            if (oc_counter == TIME_CURRENT_MAX)                         // s/w counter
reached the specified limit ?
            {
                 status_overcurrent = 1;                                // if yes, stop
 the motor
                 Motor_stop();
            }
        }
        else                                                            // if motor 'cu
rrent' sample is higher
        {                                                               // than the spe
cified limit (10 A), stop the motor
            status_overcurrent = 1;
            Motor_stop();
        }
      // USER CODE END

    }


} //  End of function ADC0_viSRN0
```

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\ADC0.C

```
// USER CODE BEGIN (ADC0_General,10)
// USER CODE END
```

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\ADC0.H

```
//****************************************************************************
// @Module        Analog / Digital Converter  (ADC0)
// @Filename      ADC0.H
// @Project       Aircraft.dav
//----------------------------------------------------------------------------
// @Controller    Infineon XE167F-96F66
//
// @Compiler      Keil
//
// @Codegenerator 2.2
//
// @Description   This file contains all function prototypes and macros for
//                the ADC0 module.
//
//----------------------------------------------------------------------------
// @Date          26/08/2014 17:45:07
//
//****************************************************************************

// USER CODE BEGIN (ADC0_Header,1)

// USER CODE END


#ifndef _ADC0_H_

#define _ADC0_H_

//****************************************************************************
// @Project Includes
//****************************************************************************

// USER CODE BEGIN (ADC0_Header,2)

// USER CODE END


//****************************************************************************
// @Macros
//****************************************************************************

// USER CODE BEGIN (ADC0_Header,3)

// USER CODE END


//****************************************************************************
// @Defines
//****************************************************************************

//   This parameter identifies ADC0 analog channel 0
#define ADC0_ANA_0 0

//   This parameter identifies ADC0 analog channel 1
#define ADC0_ANA_1 1

//   This parameter identifies ADC0 analog channel 2
#define ADC0_ANA_2 2

//   This parameter identifies ADC0 analog channel 3
#define ADC0_ANA_3 3

//   This parameter identifies ADC0 analog channel 4
#define ADC0_ANA_4 4

//   This parameter identifies ADC0 analog channel 5
#define ADC0_ANA_5 5
```

Page: 1

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\ADC0.H

```
//   This parameter identifies ADC0 analog channel 6
#define ADC0_ANA_6 6

//   This parameter identifies ADC0 analog channel 7
#define ADC0_ANA_7 7

//   This parameter identifies ADC0 analog channel 8
#define ADC0_ANA_8 8

//   This parameter identifies ADC0 analog channel 9
#define ADC0_ANA_9 9

//   This parameter identifies ADC0 analog channel 10
#define ADC0_ANA_10 10

//   This parameter identifies ADC0 analog channel 11
#define ADC0_ANA_11 11

//   This parameter identifies ADC0 analog channel 12
#define ADC0_ANA_12 12

//   This parameter identifies ADC0 analog channel 13
#define ADC0_ANA_13 13

//   This parameter identifies ADC0 analog channel 14
#define ADC0_ANA_14 14

//   This parameter identifies ADC0 analog channel 15
#define ADC0_ANA_15 15

//   This parameter identifies ADC0 -Source 0
#define ADC0_SOURCE_0 0

//   This parameter identifies ADC0 -Source 1
#define ADC0_SOURCE_1 1

//   This parameter identifies ADC0 -Source 2
#define ADC0_SOURCE_2 2


// USER CODE BEGIN (ADC0_Header,4)

// USER CODE END


//*****************************************************************************
// @Typedefs
//*****************************************************************************


// USER CODE BEGIN (ADC0_Header,5)

// USER CODE END


//*****************************************************************************
// @Imported Global Variables
//*****************************************************************************

// USER CODE BEGIN (ADC0_Header,6)

// USER CODE END


//*****************************************************************************
// @Global Variables
//*****************************************************************************
```

Page: 2

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\ADC0.H

```
// USER CODE BEGIN (ADC0_Header,7)

// USER CODE END


//****************************************************************************
// @Prototypes Of Global Functions
//****************************************************************************

void ADC0_vInit(void);


// USER CODE BEGIN (ADC0_Header,8)

// USER CODE END


//****************************************************************************
// @Interrupt Vectors
//****************************************************************************

#define ADC0_SRN0INT    0x28

// USER CODE BEGIN (ADC0_Header,9)

// USER CODE END


#endif  // ifndef _ADC0_H_
```

Page: 3

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\ADC1.C

```
//*****************************************************************************
// @Module        Analog / Digital Converter (ADC1)
// @Filename      ADC1.C
// @Project       Aircraft.dav
//-----------------------------------------------------------------------------
// @Controller    Infineon XE167F-96F66
//
// @Compiler      Keil
//
// @Codegenerator 2.2
//
// @Description   This file contains functions that use the ADC1 module.
//
//-----------------------------------------------------------------------------
// @Date          26/08/2014 17:45:07
//
//*****************************************************************************

// USER CODE BEGIN (ADC1_General,1)

// USER CODE END



//*****************************************************************************
// @Project Includes
//*****************************************************************************

#include "MAIN.H"

// USER CODE BEGIN (ADC1_General,2)

// USER CODE END


//*****************************************************************************
// @Macros
//*****************************************************************************

// USER CODE BEGIN (ADC1_General,3)

// USER CODE END


//*****************************************************************************
// @Defines
//*****************************************************************************

// USER CODE BEGIN (ADC1_General,4)

// USER CODE END


//*****************************************************************************
// @Typedefs
//*****************************************************************************

// USER CODE BEGIN (ADC1_General,5)

// USER CODE END


//*****************************************************************************
// @Imported Global Variables
//*****************************************************************************

// USER CODE BEGIN (ADC1_General,6)
```

Page: 1

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\ADC1.C

```
// USER CODE END


//*****************************************************************************
// @Global Variables
//*****************************************************************************

// USER CODE BEGIN (ADC1_General,7)

// USER CODE END


//*****************************************************************************
// @External Prototypes
//*****************************************************************************

// USER CODE BEGIN (ADC1_General,8)

// USER CODE END


//*****************************************************************************
// @Prototypes Of Local Functions
//*****************************************************************************

// USER CODE BEGIN (ADC1_General,9)

// USER CODE END


//*****************************************************************************
// @Function      void ADC1_vInit(void)
//
//-----------------------------------------------------------------------------
// @Description   This is the initialization function of the ADC function
//                library. It is assumed that the SFRs used by this library
//                are in reset state.
//
//                Following SFR fields will be initialized:
//                GLOBCTR  - Global Control
//                RSPR0    - Priority and Arbitration Register
//                ASENR    - Arbitration slot enable register
//                CHCTRx   - Channel Control Register x
//                RCRx     - Result Control Register x
//                KSCFG    - Module configuration Register
//                INPCR    - Input class Registers
//                CHINPRx  - Channel Interrupt register
//                EVINPRx  - Event Interrupt register
//                SYNCTR   - Synchronisation control register
//                LCBRx    - Limit check boundary register
//                PISEL    - Port input selection
//                QMR0     - Sequential 0 mode register
//                CRMR1    - Parallel mode register
//                QMR2     - Sequential 2 mode register
//
//-----------------------------------------------------------------------------
// @Returnvalue   None
//
//-----------------------------------------------------------------------------
// @Parameters    None
//
//-----------------------------------------------------------------------------
// @Date          26/08/2014
//
//*****************************************************************************

// USER CODE BEGIN (ADC1_Init,1)
```

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\ADC1.C

```
// USER CODE END

void ADC1_vInit(void)
{
  // USER CODE BEGIN (ADC1_Init,2)

  // USER CODE END

  ///  -----------------------------------------------------------------------
  ///  Configuration of ADC0 kernel configuration register:
  ///  -----------------------------------------------------------------------
  ADC0_KSCFG     =  0x0003;      // load ADC0 kernel configuration register

  ///  - the ADC module clock is enabled
  ///  - the ADC module clock = 66,00 MHz
  ///

  _nop_();  // one cycle delay

  _nop_();  // one cycle delay

  ///  -----------------------------------------------------------------------
  ///  Configure global control register:
  ///  -----------------------------------------------------------------------
  ///  --- Conversion Timing -----------------
  ///  - conversion time (CTC)    = 01,29 us

  ///  _Analog clock is 1/5th of module clock and digital clock is 1/1 times
  ///  of module clock

  ///  - the permanent arbitration mode is selected
  ADC1_GLOBCTR   =  0x0004;      // load global control register

  ///  -----------------------------------------------------------------------
  ///  Configuration of Arbitration Slot enable register and also the Source
  ///  Priority register:
  ///  -----------------------------------------------------------------------
  ///  - Arbitration Slot 0 is enabled

  ///  - Arbitration Slot 1 is enabled

  ///  - Arbitration Slot 2 is disabled

  ///  - the priority of request source 0 is 2
  ///  - the wait-for-start mode is selected for source 0
  ///  - the priority of request source 1 is high
  ///  - the wait-for-start mode is selected for source 1
  ///  - the priority of request source 2 is low
  ///  - the wait-for-start mode is selected for source 2
  ADC1_ASENR     =  0x0003;      // load Arbitration Slot enable register

  ADC1_RSPR0     =  0x0032;      // load Priority and Arbitration register

  ///  -----------------------------------------------------------------------
  ///  Configuration of Channel Control Registers:
  ///  -----------------------------------------------------------------------
  ///  Configuration of Channel 1
  ///  - the result register0 is selected
  ///  - the limit check 0 is selected

  ///  - the reference voltage selected is Standard Voltage (Varef)

  ///  - the input class selected is Input Class 0

  ///  - LCBR0 is selected as upper boundary

  ///  - LCBR1 is selected as lower boundary
```

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\ADC1.C

```
  ADC1_CHCTR1    =  0x0004;      // load channel control register

  ///  Configuration of Channel 2
  ///  - the result register1 is selected
  ///  - the limit check 0 is selected

  ///  - the reference voltage selected is Standard Voltage (Varef)

  ///  - the input class selected is Input Class 0

  ///  - LCBR0 is selected as upper boundary

  ///  - LCBR1 is selected as lower boundary

  ADC1_CHCTR2    =  0x1004;      // load channel control register

  ///  Configuration of Channel 4
  ///  - the result register0 is selected
  ///  - the limit check 0 is selected

  ///  - the reference voltage selected is Standard Voltage (Varef)

  ///  - the input class selected is Input Class 0

  ///  - LCBR0 is selected as upper boundary

  ///  - LCBR1 is selected as lower boundary

  ADC1_CHCTR4    =  0x0004;      // load channel control register

  ///  Configuration of Channel 7
  ///  - the result register0 is selected
  ///  - the limit check 0 is selected

  ///  - the reference voltage selected is Standard Voltage (Varef)

  ///  - the input class selected is Input Class 0

  ///  - LCBR0 is selected as upper boundary

  ///  - LCBR1 is selected as lower boundary

  ADC1_CHCTR7    =  0x0004;      // load channel control register

  ///  ------------------------------------------------------------------------
  ///  Configuration of Sample Time and Resolution:
  ///  ------------------------------------------------------------------------

  ///  10 bit resolution selected

  ADC1_INPCR0    =  0x0000;      // load input class0 register

  ///  10 bit resolution selected

  ADC1_INPCR1    =  0x0000;      // load input class1 register

  ///  ------------------------------------------------------------------------
  ///  Configuration of Result Control Registers:
  ///  ------------------------------------------------------------------------
  ///  Configuration of Result Control Register 0
  ///  - the data reduction filter is disabled
  ///  - the event interrupt is disabled
  ///  - the wait-for-read mode is disabled

  ///  - the FIFO functionality is disabled

  ADC1_RCR0      =  0x0000;      // load result control register 0
```

Page: 4

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\ADC1.C

```
  ///  Configuration of Result Control Register 1
  ///  - the data reduction filter is disabled
  ///  - the event interrupt is disabled
  ///  - the wait-for-read mode is disabled

  ///  - the FIFO functionality is disabled

  ADC1_RCR1      =  0x0000;      // load result control register 1

  ///  Configuration of Result Control Register 2
  ///  - the data reduction filter is disabled
  ///  - the event interrupt is disabled
  ///  - the wait-for-read mode is disabled

  ///  - the FIFO functionality is disabled

  ADC1_RCR2      =  0x0000;      // load result control register 2

  ///  Configuration of Result Control Register 3
  ///  - the data reduction filter is disabled
  ///  - the event interrupt is disabled
  ///  - the wait-for-read mode is disabled

  ///  - the FIFO functionality is disabled

  ADC1_RCR3      =  0x0000;      // load result control register 3

  ///  Configuration of Result Control Register 4
  ///  - the data reduction filter is disabled
  ///  - the event interrupt is disabled
  ///  - the wait-for-read mode is disabled

  ///  - the FIFO functionality is disabled

  ADC1_RCR4      =  0x0000;      // load result control register 4

  ///  Configuration of Result Control Register 5
  ///  - the data reduction filter is disabled
  ///  - the event interrupt is disabled
  ///  - the wait-for-read mode is disabled

  ///  - the FIFO functionality is disabled

  ADC1_RCR5      =  0x0000;      // load result control register 5

  ///  Configuration of Result Control Register 6
  ///  - the data reduction filter is disabled
  ///  - the event interrupt is disabled
  ///  - the wait-for-read mode is disabled

  ///  - the FIFO functionality is disabled

  ADC1_RCR6      =  0x0000;      // load result control register 6

  ///  Configuration of Result Control Register 7
  ///  - the data reduction filter is disabled
  ///  - the event interrupt is disabled
  ///  - the wait-for-read mode is disabled

  ///  - the FIFO functionality is disabled

  ADC1_RCR7      =  0x0000;      // load result control register 7

  ///  ------------------------------------------------------------------------
  ///  Configuration of Channel Interrupt Node Pointer Register:
  ///  ------------------------------------------------------------------------
  ///  - the SR0 line become activated if channel 1 interrupt is generated
```

Page: 5

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\ADC1.C

```
  ///  - the SR0 line become activated if channel 2 interrupt is generated

  ADC1_CHINPR0    =  0x0000;      // load channel interrupt node pointer
                                  // register

  ///  - the SR0 line become activated if channel 4 interrupt is generated

  ///  - the SR0 line become activated if channel 7 interrupt is generated

  ADC1_CHINPR4    =  0x0000;      // load channel interrupt node pointer
                                  // register

  ///  ---------------------------------------------------------------------
  ///  Configuration of Event Interrupt Node Pointer Register for Source
  ///  Interrupts:
  ///  ---------------------------------------------------------------------
  ///  - the SR 0 line become activated if the event 0 interrupt is generated
  ///  - the SR 0 line become activated if the event 1 interrupt is generated

  ADC1_EVINPR0    =  0x0000;      // load event interrupt set flag register

  ///  ---------------------------------------------------------------------
  ///  Configuration of Event Interrupt Node Pointer Register for Result
  ///  Interrupts:
  ///  ---------------------------------------------------------------------

  ADC1_EVINPR8    =  0x0000;      // load event interrupt set flag register


  ADC1_EVINPR12   =  0x0000;      // load event interrupt set flag register

  ///  ---------------------------------------------------------------------
  ///  Configuration of Service Request Nodes 0 - 3 :
  ///  ---------------------------------------------------------------------

  ///  ---------------------------------------------------------------------
  ///  Configuration of Limit Check Boundary:
  ///  ---------------------------------------------------------------------

  ADC1_LCBR0      =  0x0198;      // load limit check boundary register 0

  ADC1_LCBR1      =  0x0E64;      // load limit check boundary register 1

  ADC1_LCBR2      =  0x0554;      // load limit check boundary register 2

  ADC1_LCBR3      =  0x0AA8;      // load limit check boundary register 3

  ///  ---------------------------------------------------------------------
  ///  Configuration of Gating source and External Trigger Control:
  ///  ---------------------------------------------------------------------
  ///  - No Gating source selected for Arbitration Source 0

  ///  - the trigger input ETR01 is selected for Source 0

  ///  - No Gating source selected for Arbitration Source 1

  ///  - the trigger input ETR00 is selected for Source 1

  ///  - No Gating source selected for Arbitration Source 2

  ///  - the trigger input ETR00 is selected for Source 1

  ADC1_PISEL      =  0x0444;      // load external trigger control register

  ///  ---------------------------------------------------------------------
  ///  Configuration of Conversion Queue Mode Register:Sequential Source 0
  ///  ---------------------------------------------------------------------
  ///  - the gating line is permanently Enabled
```

Page: 6

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\ADC1.C

```
  ///  - the external trigger is enabled

  ADC1_QMR0      =  0x0005;      // load queue mode register

  ///  ----------------------------------------------------------------------
  ///  Configuration of Conversion Queue Mode Register:Sequential Source 2
  ///  ----------------------------------------------------------------------
  ///  - the gating line is permanently Disabled
  ///  - the external trigger is disabled
  ///  - the trigger mode 0 is selected

  ADC1_QMR2      =  0x0000;      // load queue mode register

  ///  ----------------------------------------------------------------------
  ///  Configuration of Conversion Request Mode Registers:Parallel Source
  ///  ----------------------------------------------------------------------
  ///  - the gating line is permanently Enabled
  ///  - the external trigger is disabled
  ///  - the source interrupt is disabled
  ///  - the autoscan functionality is disabled

  ADC1_CRMR1     =  0x0001;      // load conversion request mode register 1

  ///  ----------------------------------------------------------------------
  ///  Configuration of Synchronisation Registers:
  ///  ----------------------------------------------------------------------
  ///  - ADC1 is master
  ADC1_SYNCTR   |=  0x0010;      // Synchronisation register

  P15_DIDIS      =  0x0096;      // Port 15 Digital input disable register

  ADC1_GLOBCTR  |=  0x0300;      // turn on Analog part


  // USER CODE BEGIN (ADC1_Init,3)

  // USER CODE END

} //  End of function ADC1_vInit



// USER CODE BEGIN (ADC1_General,10)

// USER CODE END
```

Page: 7

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\ADC1.H

```
//*****************************************************************************
// @Module        Analog / Digital Converter (ADC1)
// @Filename      ADC1.H
// @Project       Aircraft.dav
//-----------------------------------------------------------------------------
// @Controller    Infineon XE167F-96F66
//
// @Compiler      Keil
//
// @Codegenerator 2.2
//
// @Description   This file contains all function prototypes and macros for
//                the ADC1 module.
//
//-----------------------------------------------------------------------------
// @Date          26/08/2014 17:45:07
//
//*****************************************************************************

// USER CODE BEGIN (ADC1_Header,1)

// USER CODE END


#ifndef _ADC1_H_

#define _ADC1_H_

//*****************************************************************************
// @Project Includes
//*****************************************************************************

// USER CODE BEGIN (ADC1_Header,2)

// USER CODE END


//*****************************************************************************
// @Macros
//*****************************************************************************

// USER CODE BEGIN (ADC1_Header,3)

// USER CODE END


//*****************************************************************************
// @Defines
//*****************************************************************************

//   This parameter identifies ADC1 analog channel 0
#define ADC1_ANA_0 0

//   This parameter identifies ADC1 analog channel 1
#define ADC1_ANA_1 1

//   This parameter identifies ADC1 analog channel 2
#define ADC1_ANA_2 2

//   This parameter identifies ADC1 analog channel 3
#define ADC1_ANA_3 3

//   This parameter identifies ADC1 analog channel 4
#define ADC1_ANA_4 4

//   This parameter identifies ADC1 analog channel 5
#define ADC1_ANA_5 5
```

Page: 1

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\ADC1.H

```
//   This parameter identifies ADC1 analog channel 6
#define ADC1_ANA_6 6

//   This parameter identifies ADC1 analog channel 7
#define ADC1_ANA_7 7

//   This parameter identifies ADC1 -Source 0
#define ADC1_SOURCE_0 0

//   This parameter identifies ADC1 -Source 1
#define ADC1_SOURCE_1 1

//   This parameter identifies ADC1 -Source 2
#define ADC1_SOURCE_2 2


// USER CODE BEGIN (ADC1_Header,4)

// USER CODE END


//*****************************************************************************
// @Typedefs
//*****************************************************************************


// USER CODE BEGIN (ADC1_Header,5)

// USER CODE END


//*****************************************************************************
// @Imported Global Variables
//*****************************************************************************

// USER CODE BEGIN (ADC1_Header,6)

// USER CODE END


//*****************************************************************************
// @Global Variables
//*****************************************************************************

// USER CODE BEGIN (ADC1_Header,7)

// USER CODE END


//*****************************************************************************
// @Prototypes Of Global Functions
//*****************************************************************************

void ADC1_vInit(void);


// USER CODE BEGIN (ADC1_Header,8)

// USER CODE END


//*****************************************************************************
// @Interrupt Vectors
//*****************************************************************************


// USER CODE BEGIN (ADC1_Header,9)
```

Page: 2

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\ADC1.H

```
// USER CODE END

#endif  // ifndef _ADC1_H_
```

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\CCU60.C

```
//*****************************************************************************
// @Module        Capture / Compare Unit 60 (CCU60)
// @Filename      CCU60.C
// @Project       Aircraft.dav
//-----------------------------------------------------------------------------
// @Controller    Infineon XE167F-96F66
//
// @Compiler      Keil
//
// @Codegenerator 2.2
//
// @Description   This file contains functions that use the CCU60 module.
//
//-----------------------------------------------------------------------------
// @Date          26/08/2014 17:45:07
//
//*****************************************************************************

// USER CODE BEGIN (CCU60_General,1)

// USER CODE END



//*****************************************************************************
// @Project Includes
//*****************************************************************************

#include "MAIN.H"

// USER CODE BEGIN (CCU60_General,2)

// USER CODE END


//*****************************************************************************
// @Macros
//*****************************************************************************

// USER CODE BEGIN (CCU60_General,3)

// USER CODE END


//*****************************************************************************
// @Defines
//*****************************************************************************

// USER CODE BEGIN (CCU60_General,4)

// USER CODE END


//*****************************************************************************
// @Typedefs
//*****************************************************************************

// USER CODE BEGIN (CCU60_General,5)

// USER CODE END


//*****************************************************************************
// @Imported Global Variables
//*****************************************************************************

// USER CODE BEGIN (CCU60_General,6)
```

Page: 1

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\CCU60.C

```
    // ADC offset calculation
    extern long C_SDATA adc_offset_acumm;                  // Accumulator for averagin
g of ADC offset.
    extern int C_SDATA adc_offset_resA;                    // Computed ADC offset for
bemf A.
    extern int C_SDATA adc_offset_resB;                    // Computed ADC offset for
bemf B.
    extern int C_SDATA adc_offset_resC;                    // Computed ADC offset for
bemf C.
    extern int C_SDATA adc_offset_resVdc;                  // Computed ADC offset for
DC link voltage.
    extern int C_SDATA adc_offset_resIdc;                  // Computed ADC offset for
DC link current.


    extern BIT status_overcurrent;                 // Overcurrent detection.
    extern BIT status_hs_error;                    // Wrong hall event.
    extern BIT status_time_out;                    // Time_out detection.
    extern BIT status_ctrap;                       // Ctrap detection.
    extern BIT status_bemf;                        // Bemf detection.
    extern unsigned int C_SDATA speed_meas;        // Measured speed.
    extern int C_SDATA hall_counter;               // Counter for a valid speed measur
ement.
    extern int const SP_Tab_r_l[];                 // Hall pattern.
    extern int C_SDATA direction;                  // Index-offset for reading the swi
tching pattern, depending on the specified motor direction.
    extern int C_SDATA PI_spd_out;                 // PI speed output.
    extern int C_SDATA PI_curr_out;                // PI current output.
    extern int spd_loop;                           // Speed closed loop prescaler.
    extern struct PI_cont C_SDATA PI_spd;          // PI speed variable.
    extern struct PI_cont C_SDATA PI_curr;         // PI current variable.
    extern unsigned int C_SDATA speed_meas;        // Measured speed.
    extern unsigned int C_SDATA speed_ref;         // Desired motor speed reference.
    extern int C_SDATA Idc_meas;                   // DC link current measurement.
    extern unsigned int C_SDATA speed_meas;        // Variable to store the actual mot
or speed (sector).
    extern int C_SDATA increment_free;             // Flag to specify that T13 interru
pt has occured.
    extern unsigned int C_SDATA bemf_aux;          // Stores the current value of T12
to change to sensorless mode.
    extern int C_SDATA slope;                      // Slope of the bemf: Rising = 0, F
alling = 1.
    extern unsigned int C_SDATA zc_time;           // Time between zero crossing point
s in sensorless mode
    extern unsigned int C_SDATA compare_value;     // DC link voltage/2

// USER CODE END


//******************************************************************************
// @Global Variables
//******************************************************************************

// USER CODE BEGIN (CCU60_General,7)
   int C_SDATA speed_ref_pu;                       // Reference speed in pu.
   int C_SDATA speed_meas_pu;                      // Actual speed in pu.
// USER CODE END


//******************************************************************************
// @External Prototypes
//******************************************************************************

// USER CODE BEGIN (CCU60_General,8)
  extern void Motor_stop(void);
  extern int Speed_calc(unsigned int time_meas, int* speed_counter, int limit);
  extern int pi_controller64(long *pi_parameter,int reference,int actual);
  extern void Sensorless_calc(void);
```

Page: 2

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\CCU60.C

```
// USER CODE END


//*****************************************************************************
// @Prototypes Of Local Functions
//*****************************************************************************

// USER CODE BEGIN (CCU60_General,9)

// USER CODE END


//*****************************************************************************
// @Function      void CCU60_vInit(void)
//
//-----------------------------------------------------------------------------
// @Description   This is the initialization function of the CCU60 function
//                library. It is assumed that the SFRs used by this library
//                are in reset state.
//
//-----------------------------------------------------------------------------
// @Returnvalue   None
//
//-----------------------------------------------------------------------------
// @Parameters    None
//
//-----------------------------------------------------------------------------
// @Date          26/08/2014
//
//*****************************************************************************

// USER CODE BEGIN (Init,1)

// USER CODE END

void CCU60_vInit(void)
{
  // USER CODE BEGIN (Init,2)

  // USER CODE END

  ///  -----------------------------------------------------------------------
  ///  Configuration of KERNEL REGISTERS :
  ///  -----------------------------------------------------------------------
  ///  - CCU60 Module is enabled.
  ///  - The CCU60 module clock = 66,000 MHz.
  ///  - T12 is enabled.
  ///  - T13 is enabled.
  ///  - MCM is enabled.

  CCU60_KSCFG    =  0x0003;      // Kernel State Configuration Register

  _nop_();    //No operation function as Delay
  _nop_();    //No operation function as Delay
  _nop_();    //No operation function as Delay
  _nop_();    //No operation function as Delay

  ///  -----------------------------------------------------------------------
  ///  Configuration of CCU60 Timer 12:
  ///  -----------------------------------------------------------------------
  ///  - Timer 12 Input clock factor (T12CLK) is 0
  ///  - prescaler factor is 1
  ///  - Timer 12 run bit is reset
  ///  - Single shot mode is disabled
  ///  - Timer 12 works in edge aligned mode
  ///  - Interrupt on period match is enabled
  ///  - Interrupt on one match is disabled
  ///  - No External run selection is selected.
```

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\CCU60.C

```
  ///  - Timer mode is selected.
  ///  -

  CCU60_T12PR    =  0xFFFF;      // load CCU60 T12 period register

  ///  ----------------------------------------------------------------------
  ///  Configuration of T13HR Signal:
  ///  ----------------------------------------------------------------------
  ///  - Signal T13HRA is selected as Input

  ///  ----------------------------------------------------------------------
  ///  Configuration of CCU60 Timer 13:
  ///  ----------------------------------------------------------------------
  ///  - Timer 13 Input Clock factor (T13CLK) is 0
  ///  - prescaler factor is 0
  ///  - Timer 13 run bit is reset
  ///  - Trigger control is disabled
  ///  - No External run selection is selected.
  ///  - Timer mode is selected.
  ///  -
  ///  - Single shot mode is disabled
  ///  - Interrupt on period match is enabled
  ///  - Interrupt on compare match is enabled

  CCU60_T13PR    =  0x0A4F;      // load CCU60 T13 period register

  CCU60_TCTR0    =  0x0008;      // load CCU60 timer control register 0
  CCU60_TCTR2    =  0x0020;      // load CCU60 timer control register 2

  ///  ----------------------------------------------------------------------
  ///  Configuration of Multi Channel Mode:
  ///  ----------------------------------------------------------------------
  ///  - Multi channel mode is enabled

  ///  - switching selelction:
  ///  - Transfer on T12 channel 1 compare match

  ///  - Switching synchronization:
  ///  - Synchronisation on T13 period match



  ///  ----------------------------------------------------------------------
  ///  Configuration of CCU60 Channel 0:
  ///  ----------------------------------------------------------------------
  ///  - Hall sensor mode is selected
  ///  - The output CC60 is set to the passive state
  ///  - The trap functionality of the pin CC60 is enabled
  ///  - The passive level of the output CC60 is '0'
  ///  - The output COUT60 is set to the passive state
  ///  - The trap functionality of the pin COUT60 is enabled
  ///  - The passive level of the output COUT60 is '0'
  ///  - Dead time generation is disabled

  ///  - Generation interrupt on flag ICC60R is disabled
  ///  - Generation interrupt on flag ICC60F is disabled

  CCU60_CC60SR   =  0x0000;      // Load CCU60 capture/compare shadow
                                 // register for channel 0

  ///  ----------------------------------------------------------------------
  ///  Configuration of CCU60 Channel 1:
  ///  ----------------------------------------------------------------------
  ///  - Hall sensor mode is selected
  ///  - The output CC61 is set to the passive state
  ///  - The trap functionality of the pin CC61 is enabled
  ///  - The passive level of the output CC61 is '0'
  ///  - The output COUT61 is set to the passive state
```

Page: 4

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\CCU60.C

```
///  - The trap functionality of the pin COUT61 is enabled
///  - The passive level of the output COUT61 is '0'
///  - Dead time generation is disabled

///  - Generation interrupt on flag ICC61R is disabled
///  - Generation interrupt on flag ICC61F is disabled

CCU60_CC61SR   =  0x0000;      // Load CCU60 capture/compare shadow
                               // register for channel 1

///  ------------------------------------------------------------------------
///  Configuration of CCU60 Channel 2:
///  ------------------------------------------------------------------------
///  - Hall sensor mode is selected
///  - The output CC62 is set to the passive state
///  - The trap functionality of the pin CC62 is enabled
///  - The passive level of the output CC62 is '0'
///  - The output COUT62 is set to the passive state
///  - The trap functionality of the pin COUT62 is enabled
///  - The passive level of the output COUT62 is '0'
///  - Dead time generation is disabled

///  - Generation interrupt on flag ICC62R is enabled
///  - Generation interrupt on flag ICC62F is disabled

CCU60_CC62SR   =  0x0000;      // Load CCU60 capture/compare shadow
                               // register for channel 2

///  ------------------------------------------------------------------------
///  Configuration of CCU60 Channel 3:
///  ------------------------------------------------------------------------
///  - T13 output is not inverted


CCU60_CC63SR   =  0x0000;      // load CCU60 capture/compare shadow
                               // register for channel 3


CCU60_T12DTC   =  0x0003;      // load CCU60 dead time control register for
                               // timer T12

CCU60_T12MSEL  =  0x0888;      // load CCU60 T12 campture/compare mode
                               // select register

CCU60_MODCTR   =  0x2A80;      // load CCU60 modulation control register

CCU60_MCMCTR   =  0x0014;      // load CCU60 multi channel mode control
                               // register

///  ------------------------------------------------------------------------
///  Configuration of CCU60 trap control:
///  ------------------------------------------------------------------------
///  - Trap can only be generated by SW by setting the bit TRPF
///  - The trap state is left immediately without any synchronization to
///    T12 or T13
///  - The trap state can be left as soon as bit TRPF is reset by SW
///    (according to TRPPEN, TRPM0 and TRPM1)
///  - Trap interrupt is enabled


CCU60_TRPCTR   =  0x3F07;      // Load CCU60 trap control register

///  ------------------------------------------------------------------------
///  Configuration of CCU60 interrupt control:
///  ------------------------------------------------------------------------
///  - For channel 0 interrupts is node I1 selected
///  - For channel 1 interrupts is node I1 selected
///  - For channel 2 interrupts is node I2 selected
```

Page: 5

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\CCU60.C

```
///  - For correct hall event interrupt is node I2 selected
///  - For error interrupts is node I1 selected
///  - For T12 interrupts is node I1 selected
///  - For T13 interrupts is node I3 selected


CCU60_INP      =  0x35A5;      // Load CCU60 capture/compare interrupt node
                               // pointer register

CCU60_IEN      =  0x2790;      // Load CCU60 capture/compare interrupt
                               // enable register


///  ---------------------------------------------------------------------
///  Configuration of the used CCU60 Channel Port Pins:
///  ---------------------------------------------------------------------
///  - P10.0 is used for CCU60 output(CC60)
///  - P10.1 is used for CCU60 output(CC61)
///  - P10.2 is used for  CCU60 output(CC62)
///  - P10.3 is used for  CCU60 output(COUT60)
///  - P10.4 is used for CCU60 output(COUT61)
///  - P10.5 is used for CCU60 output(COUT62)
///  - P10.7 is used for  CCU60 input(CCPOS0A)
///  - P10.8 is used for  CCU60 input(CCPOS1A)
///  - P10.9 is used for CCU60 input(CCPOS2A)
///  - P10.6 is used for  CCU60 input(CTRAPA)

P10_IOCR00 = 0x00A0;    //set direction register
P10_IOCR01 = 0x00A0;    //set direction register
P10_IOCR02 = 0x00A0;    //set direction register
P10_IOCR03 = 0x00A0;    //set direction register
P10_IOCR04 = 0x00A0;    //set direction register
P10_IOCR05 = 0x00A0;    //set direction register

CCU60_PISELL   =  0xC000;      // Load CCU60 Port Input Selection register

///  ---------------------------------------------------------------------
///  Configuration of the used CCU60 Channels Interrupts:
///  ---------------------------------------------------------------------
///  NodeI1 service request node configuration:
///  - NodeI1 interrupt priority level (ILVL) = 8
///  - NodeI1 interrupt group level (GLVL) = 0
///  - NodeI1 group priority extension (GPX) = 0

CCU60_1IC      =  0x0060;

///  NodeI2 service request node configuration:
///  - NodeI2 interrupt priority level (ILVL) = 5
///  - NodeI2 interrupt group level (GLVL) = 1
///  - NodeI2 group priority extension (GPX) = 0

CCU60_2IC      =  0x0055;

///  NodeI3 service request node configuration:
///  - NodeI3 interrupt priority level (ILVL) = 6
///  - NodeI3 interrupt group level (GLVL) = 2
///  - NodeI3 group priority extension (GPX) = 0

CCU60_3IC      =  0x005A;


///  ---------------------------------------------------------------------
///  Configuration of T12, T13 ---- CCU60_TCTR4 Register:
///  ---------------------------------------------------------------------
///  - Enable shadow transfer of T12 and T13
///  - Timer 12 run bit is reset
///  - Timer 13 run bit is reset
CCU60_TCTR4    =  0x4040;      // load CCU60 timer control register 4
```

Page: 6

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\CCU60.C

```
  // USER CODE BEGIN (NodeI3,3)

  // USER CODE END

} //  End of function CCU60_vInit


//*****************************************************************************
// @Function      void CCU60_viNodeI1(void)
//
//-----------------------------------------------------------------------------
// @Description   This is the interrupt service routine for the CCU60 node
//                I1. If the content of the corresponding compare timer
//                (configurable) equals the content of the capture/compare
//                register or if a capture event occurs at the associated
//                port pin, the interrupt request flag is set and an
//                interrupt is triggered (only if enabled).
//                Please note that you have to add application specific code
//                to this function.
//
//-----------------------------------------------------------------------------
// @Returnvalue   None
//
//-----------------------------------------------------------------------------
// @Parameters    None
//
//-----------------------------------------------------------------------------
// @Date          26/08/2014
//
//*****************************************************************************

// USER CODE BEGIN (NodeI1,1)

// USER CODE END

void CCU60_viNodeI1(void) interrupt CCU60_NodeI1_INT
{
  // USER CODE BEGIN (NodeI1,2)

  // USER CODE END

  if(CCU60_IS & 0x0080)  // if CCU60_IS_T12PM
  {
    // Timer T12 period match detection

    // USER CODE BEGIN (NodeI1,19)
      status_time_out=1;                                    // ... set the time-out
 status flag.
      Motor_stop();                                         // ... stop the motor.
    // USER CODE END

    CCU60_ISR |= 0x0080;  // clear flag CCU60_IS_T12PM
  }

  if(CCU60_IS & 0x0400)  // if CCU60_IS_TRPF
  {
    // Trap detection

    // USER CODE BEGIN (NodeI1,17)
       status_ctrap=1;                                      // ... set the trap sta
tus flag.
       Motor_stop();                                        // ... stop the motor.
    // USER CODE END

    CCU60_ISR |= 0x0400;  // clear flag CCU60_IS_TRPF
  }
```

Page: 7

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\CCU60.C

```
  if(CCU60_IS & 0x2000)  // if CCU60_IS_WHE
  {
    // Wrong hall event detection

    // USER CODE BEGIN (NodeI1,18)
      status_hs_error=1;                                    // ... set the hall sen
sor error status flag.

      if(status_bemf)                                       // If bemf detection is
 possible...
      {
        bemf_aux = CCU60_T12;                                   // ... save the act
ual T12 value.
        if(slope == 0 || slope == 1)                            // If bemf not dete
cted yet in this sector...
        {
            CCU60_T12MSEL   = 0x0111;                               // ... change h
all sensor mode to sensorless mode.
            CCU60_IEN       &= 0x97FF;                              // ... disable
wrong hall event or idle interrupts.
            CCU60_TCTR4     = 0x0045;                               // ... T12 rese
t and stop.
            CCU60_T12       = zc_time + bemf_aux;                   // ... set the
counter time to the half of the sector time (actual zc_time).
            CCU60_T12PR     = (zc_time << 1) + zc_time;             // ... set the
time out limit to 1.5 of the sector time.
            CCU60_TCTR4     = 0x0040;                               // ... shadow t
ransfer request for T12.
            CCU60_TCTR4     = 0x0002;                               // ... start T1
2.
            if(bemf_aux < CCU60_CC61R)                              // If the commu
tation was not done yet...
            {
             CCU60_MCMOUTS |= 0x0080;                                   // ... forc
e commutation.
            }
            if(bemf_aux < CCU60_CC62R)                              // If the updat
e of the pattern was not done yet...
            {
             CCU60_MCMOUTS  = CCU60_MCMOUTS = SP_Tab_r_l[((CCU60_MCMOUT & 0x3800) >> 11
) + direction]; // ... load next commutation pattern.
            }
        }
        else                                                    // If bemf has alre
ady been detected in this sector...
        {
             CCU60_T12MSEL  = 0x0111;                               // ... change h
all sensor mode to sensorless mode.
             CCU60_IEN      &= 0x97FF;                              // ... disable
wrong hall event or idle interrupts.
             CCU60_TCTR4    = 0x0045;                               // ... T12 rese
t and stop.
             CCU60_T12      = bemf_aux - zc_time;                   // ... set the
counter time to the difference between the actual value and half the sector time.
             CCU60_T12PR    = (zc_time << 1) + zc_time;             // ... set the
time out limit to 1.5 the sector time.
             CCU60_CC60SR   = (zc_time << 1);                       // ... set the
CC60R for speed measurement with the time between zero crossing points.
             CCU60_CC61SR   = zc_time - 5;                          // ... initiali
ze the CCU61 register with half of the time between zero crossings,
                                                                    // so that it c
ommutates within the required time (with an advance).
             CCU60_CC62SR   = CCU60_CC61SR + 10;                    // ... initiali
ze CCU62 some time after to calculate the speed.
             CCU60_TCTR4    = 0x0040;                               // ... shadow t
ransfer request for T12.
             CCU60_TCTR4    = 0x0002;                               // ... start T1
```

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\CCU60.C

```
2.
        }
      }
      else                                                    // If bemf detection i
s not possible...
      {
      Motor_stop();                                             // ... stop the mot
or.
      }
    // USER CODE END

    CCU60_ISR |= 0x2000;  // clear flag CCU60_IS_WHE
  }


} //  End of function CCU60_viNodeI1


//****************************************************************************
// @Function      void CCU60_viNodeI2(void)
//
//----------------------------------------------------------------------------
// @Description   This is the interrupt service routine for the CCU60 node
//                I2. If the content of the corresponding compare timer
//                (configurable) equals the content of the capture/compare
//                register or if a capture event occurs at the associated
//                port pin, the interrupt request flag is set and an
//                interrupt is triggered (only if enabled).
//                Please note that you have to add application specific code
//                to this function.
//
//----------------------------------------------------------------------------
// @Returnvalue   None
//
//----------------------------------------------------------------------------
// @Parameters    None
//
//----------------------------------------------------------------------------
// @Date          26/08/2014
//
//****************************************************************************

// USER CODE BEGIN (NodeI2,1)

// USER CODE END

  int arrexp[8]={0,5,3,1, 6,4,2,0,}; //for clockwise
  unsigned int MCMData[8] = {0x190e, 0x190e, 0x3223, 0x132c,  0x2c38,  0x0d32, 0x260b,
0x190e};  //FOR CLOCLKWISE DIRECTION WORKING FINE

  //int arrexp[8]={0,3,6,2,5,1,4,0,}; //for anti clockwise direction
  //unsigned int MCMData[8] = {0x292c, 0x292c, 0x1A0B, 0x0B23,  0x3432,  0x250E, 0x1638
, 0x292c};  // FOR ANTI - CLOCLKWISE DIRECTION WORKING FINE

void CCU60_viNodeI2(void) interrupt CCU60_NodeI2_INT
{
  // USER CODE BEGIN (NodeI2,2)

  // USER CODE END

  if(CCU60_IS & 0x0010)  // if CCU60_IS_ICC62R
  {
    // Capture, Compare match rising edge detection an channel 2

    // USER CODE BEGIN (NodeI2,14)

    CCU60_MCMOUTS = SP_Tab_r_l[((CCU60_MCMOUT & 0x3800) >> 11) + direction];        //
... load the next commutation pattern.
```

Page: 9

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\CCU60.C

```
    speed_meas = Speed_calc(CCU60_CC60R, &hall_counter, (18*PP));                    //
... calculate the speed.

    if (slope == 2)                                                                  //
If a falling edge of the bemf has been detected...
    {
        slope = 0;
    // ... we need to look for rising edge.
    }
    if (slope == 3)                                                                  //
If a rising edge of the bemf has been detected...
    {
        slope = 1;
    // ... we need to look for falling edge.
    }

    if (hall_counter < 2)                                                            //
If motor speed is too small...
    {
        speed_meas = 0;
    }
    // ... reset to 0 (Only done at the first measurements of the speed).
    // USER CODE END

    CCU60_ISR |= 0x0010;  // clear flag CCU60_IS_ICC62R
  }


} //  End of function CCU60_viNodeI2


//*****************************************************************************
// @Function      void CCU60_viNodeI3(void)
//
//-----------------------------------------------------------------------------
// @Description   This is the interrupt service routine for the CCU60 node
//                I3. If the content of the corresponding compare timer
//                (configurable) equals the content of the capture/compare
//                register or if a capture event occurs at the associated
//                port pin, the interrupt request flag is set and an
//                interrupt is triggered (only if enabled).
//                Please note that you have to add application specific code
//                to this function.
//
//-----------------------------------------------------------------------------
// @Returnvalue   None
//
//-----------------------------------------------------------------------------
// @Parameters    None
//
//-----------------------------------------------------------------------------
// @Date          26/08/2014
//
//*****************************************************************************

// USER CODE BEGIN (NodeI3,1)

// USER CODE END

void CCU60_viNodeI3(void) interrupt CCU60_NodeI3_INT
{
  // USER CODE BEGIN (NodeI3,2)
   unsigned int x;
  // USER CODE END

  if(CCU60_IS & 0x0100)  // if CCU60_IS_T13CM
  {
    // Timer T13 compare match detection
```

Page: 10

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\CCU60.C

```
    // USER CODE BEGIN (NodeI3,22)
    bemf_aux = (CCU60_MCMOUT & 0x003F);                                    // ...
read the present modulation pattern.

    for(x=0; x<200; x++)
    {
     _nop_();                                                              // No o
peration function delay
    }

    if((bemf_aux == BEMF_AUX_VAL11) || (bemf_aux == BEMF_AUX_VAL12))       // If p
hase A is inactive...
    {
        ADC1_CRPR1 = 0x0001;                                                //
... convert channel 1 of ADC1.
    }
    if((bemf_aux == BEMF_AUX_VAL21) || (bemf_aux == BEMF_AUX_VAL22))       // If p
hase B is inactive...
    {
        ADC1_CRPR1 = 0x0004;                                                //
... convert channel 4 of ADC1.
    }
    if((bemf_aux == BEMF_AUX_VAL31) || (bemf_aux == BEMF_AUX_VAL32))       // If p
hase B is inactive...
    {
        ADC1_CRPR1 = 0x0007;                                                //
... convert channel 4 of ADC1.
    }
    ADC0_CRPR1 = 0x0003;                                                   // ...
convert DC link current.
    ADC1_QINR0 = 0x0082;                                                   // ...
request DC link voltage measurement (on T13 period match).
    compare_value   =  ((ADC1_RESR1>>3)&0x01FF)-adc_offset_resVdc;         // ...
set the compare value to half the DC link voltage.
    // USER CODE END

    CCU60_ISR |= 0x0100;  // clear flag CCU60_IS_T13CM
  }

  if(CCU60_IS & 0x0200)  // if CCU60_IS_T13PM
  {
    // Timer T13 period match detection

    // USER CODE BEGIN (NodeI3,21)
        if(slope==1 || slope==0)                                           // If b
emf still not detected...
        {
            Sensorless_calc();                                              //
... check the bemf and calculate the new parameters if necessary.
        }

       if(spd_loop==1)                                                     // If 1
 ms reached...
        {
        speed_ref_pu  = FloatTo1Q15((float)(speed_ref/MAX_SPEED));          //
... calculate the speed reference in pu (1Q15).
        speed_meas_pu = FloatTo1Q15((float)(speed_meas/MAX_SPEED));         //
... calculate the measured speed in pu (1Q15).
        PI_spd_out = pi_controller64(&PI_spd.yn, speed_ref_pu, speed_meas_pu);      //
... compute the I_ref.
        spd_loop=0;                                                         //
... reset the counter.
        }
        else                                                               // If 1
 ms not reached...
        {
        spd_loop++;                                                         //
```

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\CCU60.C

```
... increase the counter.
        }
        Idc_meas = ((ADC1_RESR0<<3) & 0x7FE0)-adc_offset_resIdc;                // ...
save the measured current in pu (1Q15).
        PI_curr_out = pi_controller64(&PI_spd.yn, PI_spd_out, Idc_meas);        // ...
compute the duty_cycle.
        CCU60_CC63SR = (int)((3300*((long)(32767-PI_curr_out)))>>15);           // ...
set the duty cycle to T13.
        CCU60_TCTR4 = 0x4000;                                                   // ...
enable shadow transfer.
        increment_free = 1;                                                     // ...
set flag for speed ramp.
    // USER CODE END

    CCU60_ISR |= 0x0200;  // clear flag CCU60_IS_T13PM
  }


} //  End of function CCU60_viNodeI3



// USER CODE BEGIN (CCU60_General,10)

// USER CODE END
```

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\CCU60.H

```
//*****************************************************************************
// @Module        Capture / Compare Unit 60 (CCU60)
// @Filename      CCU60.H
// @Project       Aircraft.dav
//-----------------------------------------------------------------------------
// @Controller    Infineon XE167F-96F66
//
// @Compiler      Keil
//
// @Codegenerator 2.2
//
// @Description   This file contains all function prototypes and macros for
//                the CCU60 module.
//
//-----------------------------------------------------------------------------
// @Date          26/08/2014 17:45:07
//
//*****************************************************************************

// USER CODE BEGIN (CCU60_Header,1)

// USER CODE END


#ifndef _CCU60_H_
#define _CCU60_H_

//*****************************************************************************
// @Project Includes
//*****************************************************************************

// USER CODE BEGIN (CCU60_Header,2)

// USER CODE END


//*****************************************************************************
// @Macros
//*****************************************************************************

// USER CODE BEGIN (CCU60_Header,3)

// USER CODE END


//*****************************************************************************
// @Defines
//*****************************************************************************

//   This parameter identifies CCU60 timer 12
#define CCU60_TIMER_12 12

//   This parameter identifies CCU60 timer 13
#define CCU60_TIMER_13 13

//   This parameter identifies CCU60 channel 0
#define CCU60_CHANNEL_0 0

//   This parameter identifies CCU60 channel 1
#define CCU60_CHANNEL_1 1

//   This parameter identifies CCU60 channel 2
#define CCU60_CHANNEL_2 2

//   This parameter identifies CCU60 channel 3
#define CCU60_CHANNEL_3 3
```

Page: 1

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\CCU60.H

```
// USER CODE BEGIN (CCU60_Header,4)

// USER CODE END


//****************************************************************************
// @Typedefs
//****************************************************************************

// USER CODE BEGIN (CCU60_Header,5)

// USER CODE END


//****************************************************************************
// @Imported Global Variables
//****************************************************************************

// USER CODE BEGIN (CCU60_Header,6)

// USER CODE END


//****************************************************************************
// @Global Variables
//****************************************************************************

// USER CODE BEGIN (CCU60_Header,7)

// USER CODE END


//****************************************************************************
// @Prototypes Of Global Functions
//****************************************************************************

void CCU60_vInit(void);


// USER CODE BEGIN (CCU60_Header,8)

// USER CODE END


//****************************************************************************
// @Interrupt Vectors
//****************************************************************************

#define CCU60_NodeI1_INT    0x31
#define CCU60_NodeI2_INT    0x32
#define CCU60_NodeI3_INT    0x33


// USER CODE BEGIN (CCU60_Header,9)

// USER CODE END


#endif  // ifndef _CCU60_H_
```

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\FUNC.C

```
#include "MAIN.H"

// Variable declaration

    // ADC offset calculation
    long C_SDATA adc_offset_acumm;              // Accumulator for averaging of ADC off
set.
    int C_SDATA adc_offset_resA;                // Computed ADC offset for bemf A.
    int C_SDATA adc_offset_resB;                // Computed ADC offset for bemf B.
    int C_SDATA adc_offset_resC;                // Computed ADC offset for bemf C.
    int C_SDATA adc_offset_resVdc;              // Computed ADC offset for DC link volt
age.
    int C_SDATA adc_offset_resIdc;              // Computed ADC offset for DC link curr
ent.

    // Declaration of bit-variables required to store motor status
    C_BDATA status_word;                                    // Full motor status word.

    C_SBIT status_on            C_ATBIT(status_word,0);     // ON.

    C_SBIT status_off           C_ATBIT(status_word,1);     // OFF.

    C_SBIT status_overcurrent   C_ATBIT(status_word,2);     // Overcurrent detection.
    C_SBIT status_hs_error      C_ATBIT(status_word,3);     // Hall sensor error detect
ion.
    C_SBIT status_time_out      C_ATBIT(status_word,4);     // Time_out detection.
    C_SBIT status_ctrap         C_ATBIT(status_word,5);     // Ctrap detection.
    C_SBIT status_bemf          C_ATBIT(status_word,6);     // Status of bemf detection
.
    C_SBIT status_overvoltemp   C_ATBIT(status_word,7);     // Overvoltage or overtempr
eature detection.

    // Global variables
    int C_SDATA direction;                          // Index-offset for reading the swi
tching pattern, depending on the specified motor direction.
    int C_SDATA once;                               // Variable to allow motor start.
    unsigned int C_SDATA sv_start_ref;              // Start speed reference for motor
start.
    unsigned int C_SDATA sv_start_ref0;             // Copy of start reference to check
 changes in end reference and recalculate ramp time.
    unsigned int C_SDATA sv_end_ref;                // End speed reference for motor st
art.
    int C_SDATA motor_status;                       // Motor stop command.
    int C_SDATA direction_user;                     // Variable which stores the desire
d direction of rotation.
    unsigned long C_SDATA rampup_time;              // Time to achieve the end referenc
e.
    int C_SDATA trap_reset;                         // Set to '1' to reset the 'Latched
 CTRAP' condition.
    unsigned int C_SDATA speed_ref;                 // Desired motor speed reference.
    unsigned int C_SDATA circ_index;                // Index to access the circular buf
fer used in motor speed computation.
    unsigned int C_SDATA speed_buffer[6*PP];        // Buffer to preserve the previous
samples of motor speed.
    int C_SDATA oc_counter;                         // Counter to specify the time limi
t for motor overcurrent.
    unsigned long C_SDATA time_step;                // Sets the increment frequency of
the speed reference.
    unsigned long C_SDATA counter1;                 // Counter to increase the speed re
ference.
    unsigned long C_SDATA speed_acumm;              // Accumulator to calculate the ave
rage motor speed.
    int C_SDATA hall_counter;                       // Counter to start taking into acc
ount speed measurements.
    int C_SDATA speed_counter;                      // Counter to start measuring avera
ge speed.
    int C_SDATA increment_free;                     // Flag to specify that T13 interru
```

Page: 1

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\FUNC.C

```
pt has occured.
    int C_SDATA spd_loop;                           // Speed closed loop prescaler.
    int C_SDATA Idc_meas;                           // DC link current measurement.
    unsigned int C_SDATA speed_meas;                // Variable to store the actual mot
or speed (sector).

    // BEMF
    unsigned int C_SDATA compare_value;             // DC link voltage/2
    int C_SDATA slope;                              // Slope of the bemf: Rising = 0, F
alling = 1.
    int C_SDATA CH_conv;                            // Stores the last converted channe
l to apply the corresponding offset.
    unsigned int C_SDATA zc_time;                   // Time between zero crossing point
s in sensorless mode.
    unsigned int C_SDATA bemf_aux;                  // Stores the current value of T12
to change to sensorless mode.
    unsigned int C_SDATA back_emf_measure;          // Last measured bemf.
    int C_SDATA slope_known;                        // Slope of the bemf is detected .
    int C_SDATA bemf_counter1;                      // Counter for detection of bemf.

    // PI variables
    struct PI_cont C_SDATA PI_spd;                  // PI speed variable.
    struct PI_cont C_SDATA PI_curr;                 // PI current variable.
    int C_SDATA PI_spd_out;                         // PI speed output.
    int C_SDATA PI_curr_out;                        // PI current output.


// Function declaration

    void ADC_OFFSET(int CH_OFFSET);
    void Init_control(void);
    void Motor_start(void);
    void Motor_stop(void);
    void Commutation_init(void);
    void RampUp(void);
    void Speed_ref_ramp(void);
    int  Speed_calc(unsigned int time_meas, int* speed_counter, int limit);
    void Sensorless_calc(void);

// Modulation pattern
    #define MCMOUTS_CTE_R(curhs,exphs,mcmps)      (curhs << 11) | (exphs << 8) | (mcmps
)
    #define MCMOUTS_CTE_L(curhs,exphs,mcmps)      (curhs << 11) | (exphs << 8) | (mcmps
)

    int const SP_St_Tab_r_l[16] =
          // For motor start  (0-7: Clockwise, 8-15: Counterclockwise)
        {
            0x0000, MCMOUTS_CTE_R(1,3,0x24), MCMOUTS_CTE_R(2,6,0x09), MCMOUTS_CTE_R(3,2
,0x21),
            MCMOUTS_CTE_R(4,5,0x12), MCMOUTS_CTE_R(5,1,0x06), MCMOUTS_CTE_R(6,4,0x18),
0x0000,
            0x0000, MCMOUTS_CTE_L(1,5,0x18), MCMOUTS_CTE_L(2,3,0x06), MCMOUTS_CTE_L(3,1
,0x12),
            MCMOUTS_CTE_L(4,6,0x21), MCMOUTS_CTE_L(5,4,0x09), MCMOUTS_CTE_L(6,2,0x24),
0x0000
        };
    int const SP_Tab_r_l[16] =
          // Normal operation (0-7: Clockwise, 8-15: Counterclockwise)
        {
            0x0000, MCMOUTS_CTE_R(3,2,0x21), MCMOUTS_CTE_R(6,4,0x18), MCMOUTS_CTE_R(2,6
,0x09),
            MCMOUTS_CTE_R(5,1,0x06), MCMOUTS_CTE_R(1,3,0x24), MCMOUTS_CTE_R(4,5,0x12),
0x0000,
            0x0000, MCMOUTS_CTE_L(5,4,0x09), MCMOUTS_CTE_L(3,1,0x12), MCMOUTS_CTE_L(1,5
,0x18),
            MCMOUTS_CTE_L(6,2,0x24), MCMOUTS_CTE_L(4,6,0x21), MCMOUTS_CTE_L(2,3,0x06),
0x0000
```

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\FUNC.C

```
        };

// Function implementation

//*************************************************************************
// @Function        void ADC_OFFSET(int CH_OFFSET)
// Description      This function computes the ADC offset value, by taking
//                  65,536 samples of the ADC measured data. This computed
//                  offset value is preserved in the variable adc_offset_resX.
// Parameters       Channel to be converted
//-------------------------------------------------------------------------
void ADC_OFFSET(int CH_OFFSET)
{
    unsigned int i = 0;
    adc_offset_acumm = 0;                                           // Reset the ac
cumulator.
    switch(CH_OFFSET)
    {
        case 1:                                                     // Phase A.
            for(i = 0; i < 65535; i++)                                  // Until al
l samples not accumulated, be in the loop.
            {
                ADC1_CRPR1 = 0x0001;                                        // Requ
est 'start sampling' command to the ADC.
                while(!(ADC1_VFR & 0x0001));                                // Wait
 until valid conversion.
                adc_offset_acumm += (ADC1_RESR0>>2 & 0x03FF);               // Accu
mulate the sample, right aligned.
            }
            adc_offset_resA = (int)(adc_offset_acumm >> 16);            // Do the a
verage of all the samples.
            break;
        case 4:                                                     // Phase B
            for(i = 0; i < 65535; i++)                                  // Until al
l samples not accumulated, be in the loop.
            {
                ADC1_CRPR1 = 0x0004;                                        // Requ
est 'start sampling' command to the ADC.
                while(!(ADC1_VFR & 0x0001));                                // Wait
 until valid conversion.
                adc_offset_acumm += (ADC1_RESR0>>2 & 0x03FF);               // Accu
mulate the sample, right aligned.
            }
            adc_offset_resB = (int)(adc_offset_acumm >> 16);            // compute
the average of all the samples.
            break;
        case 7:                                                     // Phase C
            for(i = 0; i < 65535; i++)                                  // Until al
l samples not accumulated, be in the loop.
            {
                ADC1_CRPR1 = 0x0007;                                        // Requ
est 'start sampling' command to the ADC.
                while(!(ADC1_VFR & 0x0001));                                // Wait
 until valid conversion.
                adc_offset_acumm += (ADC1_RESR0>>2 & 0x03FF);               // Accu
mulate the sample, right aligned.
            }
            adc_offset_resC = (int)(adc_offset_acumm >> 16);            // Do the a
verage of all the samples.
            break;
        case 3:                                                     // DC link curr
ent.
            for(i = 0; i < 65535; i++)                                  // Until al
l samples not accumulated, be in the loop.
            {
                ADC0_CRPR1 = 0x0003;                                        // Requ
est 'start sampling' command to the ADC.
                while(!(ADC0_VFR & 0x0001));                                // Wait
```

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\FUNC.C

```
 until valid conversion.
                adc_offset_acumm += ((ADC0_RESR0 & 0x0FFC) << 3);               // Accu
mulate the sample, 1Q15.
            }
            adc_offset_resIdc = (int)(adc_offset_acumm >> 16);              // Do the a
verage of all the samples.
            break;
        case 2:                                                            // DC link volt
age.
            for(i = 0; i < 65535; i++)                                          // Until al
l samples not accumulated, be in the loop.
            {
                ADC1_CRPR1 = 0x0002;                                            // Requ
est 'start sampling' command to the ADC.
                while(!(ADC1_VFR & 0x0002));                                    // Wait
 until valid conversion.
                adc_offset_acumm += (ADC1_RESR0>>3 & 0x01FF);                   // Accu
mulate the sample, 1Q15.
            }
            adc_offset_resVdc = (int)((adc_offset_acumm >> 16)-511);        // Do the a
verage of all the samples.
            break;
    }
}

//*****************************************************************************
// @Function        void Motor_start(void)
// Description      This function executes the functions required for Motor
//                  start-up such as initialization of motor related variables,
//                  initialization of motor SFRs, etc.
//                  It also ensures that the start reference is always lower
//                  than the end reference.
//-----------------------------------------------------------------------------
void Motor_start(void)
{
    if(once == 0)                                           // Execute only the first t
ime.
    {
        once = 0x0001;                                      // Set the flag so that thi
s code never executes until next motor start-up.
        status_word = 0x0001;                               // Set 'status_on' flag to
convey to the user that 'Motor_start' command is issued.

        if(sv_start_ref > sv_end_ref)                       // Start reference limitati
on.
        {
            sv_start_ref = sv_end_ref;                          // Ensure that the star
t reference never exceeds the end reference.
        }

        sv_start_ref0 = sv_start_ref;                       // Make another copy of sta
rt voltage reference to check later if the end reference changes.
        speed_ref =  sv_start_ref;                          // Set start speed referenc
e into actual speed reference.

        Init_control();                                     // Initialisation of all th
e variables required for motor start-up.
        Commutation_init();                                 // Initialisation of all SF
Rs required for motor start-up.
    }
}

//*****************************************************************************
// @Function        void Init_control(void)
// Description      This function initialises various variables required
//                  for motor start-up.
//-----------------------------------------------------------------------------
void Init_control(void)
```

Page: 4

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\FUNC.C

```
{

    int i = 0;
    for(i = 0;i<(6*PP);i++)
    {
        speed_buffer[i] = 0;        // Initialise the speed buffer.
    }
    speed_acumm     = 0x00000008;   // Initialise speed accumulator with 8 so that divi
sion by 8 won't result in zero.
    circ_index      = 0x0000;       // Initialise the circular index for speed measurem
ent.
    speed_counter   = 0x0000;       // Initialise the speed counter.
    hall_counter    = 0x0000;       // Initialise the hall counter.

    counter1        = 0x00000000;   // Initialise counter 1.
    time_step       = 0x00000000;   // Initialise the time_step for the ramp-up.

    slope_known     = 0x0000;       // Initialise the detection of the bemf slope.
    bemf_counter1   = 0x0000;       // Initialise bemf counter 1.
    slope           = 0x0000;       // Initialise the slope of the bemf.

    spd_loop        = 0;            // Initialise the speed closed loop prescaler.
    PI_spd.yn       = 0x00000000;   // Initialise the integral part of the speed PI.
    PI_curr.yn      = 0x00000000;   // Initialise the integral part of the current PI.

    oc_counter      = 0x0000;       // Initialise the s/w time base for overcurrent det
ection.
}

//*****************************************************************************
// @Function        void Commutation_init(void)
// Description      This function initialises the SFRs required for motor start-up
//                  such as loading of the modulation register, enabling of the timers,
//                  enabling of the interrupts etc.
//-----------------------------------------------------------------------------
void Commutation_init(void)
{
    int aux = 0;

    CCU60_MCMCTR = 0x0000;                          // No switching or synchronization
available.

    CCU60_CC61SR  = 3;                              // Initialise CC61 register to avoi
d hall sensor spikes.
    CCU60_CC62SR  = CCU60_CC61SR + 5;               // Initialise CC62 some time later
than CC61 to calculate speed and set the new pattern.
    CCU60_T12     = CCU60_CC62SR + 1;               // Initialize T12 timer with a suff
iciently larger value so that
                                                    // it doesn't generate an interrupt
 immediately.

    CCU60_CC63SR  = 0x0CE4;                         // Initialize channel 3 shadow regi
ster with value corresponding to 0% duty cycle.

    CCU60_TCTR4   = 0x4040;                         // Select the events to trigger sha
dow transfers of T12 & T13.

    aux = ((P10_IN & 0x0380)>>7) + direction;       // Read the hall sensor pins from t
he corresponding port.

    CCU60_MCMOUTS = SP_St_Tab_r_l[aux] | 0x8080;    // Load the Modulation pattern corr
esponding to the hall sensor data
                                                    // and force the shadow transfer of
 the modulation pattern to the actual register.
    CCU60_MCMOUTS = SP_Tab_r_l[aux];                // Load the expected hall and modul
ation patterns into shadow register.

    CCU60_IEN    |= 0x6000;                         // Enable idle and wrong hall event
```

Page: 5

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\FUNC.C

```
 interrupts.

    CCU60_TCTR4   = 0x0600;                         // T13 Run --> Start PWM.


    CCU60_MCMCTR  = 0x0014;                         // Trigger shadow tranfers at CC61R
 compare match and synchronized to T13 zero match.

    CCU60_TCTR4   = 0x0002;                         // T12 Run.

    CCU60_TRPCTR |= 0x8000;                         // Enable CCU6 trap.

}

//***************************************************************************
// @Function        void Motor_stop(void)
// Description      This function implements the functions required for motor
//                  stop such as switching off the PWM, all MOSFETs, stopping
//                  all timers etc.
//---------------------------------------------------------------------------

void Motor_stop(void)
{

    CCU60_MCMCTR    = 0x0000;               // Enable shadow tranfers direct mode.


    CCU60_MCMOUTS   = 0x8080;               // All transistors OFF.

    CCU60_TRPCTR    &= 0x7FFF;              // Trap disable.


    CCU60_TCTR4     = 0x0505;               // T13/T12 Reset, T13/T12 Stop -->  No PWM
active.

    CCU60_CC63SR    = 0x0CE4;               // Re-initialise shadow register channel 3
with duty cycle = 0,
                                            // so that it is ready with the correct val
ue when 'motor start' command is requested.


    CCU60_TCTR4     = 0x4000;               // Shadow Tranfer by T13.

    CCU60_IEN       &= 0x3FFF;              // Wrong hall event and idle interrupts dis
abled.

    status_off      = 1;                    // Set motor off.
    status_on       = 0;                    // Clear motor on.

    if(motor_status == 1)                   // If motor stop command detected...
    {
        once = 0x0000;                          // ... clear variable 'once' so that it
 executes motor start function when user requests 'start' command.
    }

    if(direction_user > 1)                  // If 'direction_user' is higher than 1...
    {
        direction_user = 1;                     // ... ensure that 'direction_user' nev
er takes an invalid value.
    }

    direction = direction_user * 8;         // Set the offset for the pattern based on
'direction_user'.

}

//***************************************************************************
```

Page: 6

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\FUNC.C

```
// @Function        void RampUp(void)
// Description     This function verifies whether the user has entered a new
//                 'end reference' and accordingly ramps-up the duty cycle,
//                 to reach the new 'end reference'.
//-----------------------------------------------------------------------------
void RampUp(void)
{

        int ref_difference = 0;                             // Initialise the differenc
e.
        if(increment_free == 1)                             // If the T13 period match
timer interrupt has occured...
        {
          increment_free = 0;                                   // ... clear the interr
upt flag.
          counter1 += 64;                                       // ... increase the cou
nter.
        }

        if(counter1 > time_step)                            // If the 'time_step' for r
amping has elapsed...
        {
          Speed_ref_ramp();                                     // ... increment the re
ference speed by one count.
          counter1 = 0;                                         // ... reset the counte
r.
        }

        if(sv_start_ref0 != sv_end_ref)                     // If the end speed referen
ce has been changed by the user...
        {

          C_CoLOAD_0_1(0, C_LSHIFT(sv_end_ref, 16));            // ... then calculate t
he new difference that the motor has to achieve.
          C_CoSUB_0_1(0, C_LSHIFT(sv_start_ref0, 16));
          C_CoABS();
          ref_difference = C_CoSTOREMAS();
          if(ref_difference > 1)                                    // ... substract th
e already done increment.
          {
            ref_difference -= 1;
          }
          time_step = rampup_time  / ref_difference;            // ... calculate the ne
w 'time_step' for the new difference in speed.
          sv_start_ref0 = sv_end_ref;                           // ... assign the new r
eference so it doesn't calculate again.
        }

}

//*****************************************************************************
// @Function        void Speed_ref_ramp(void)
// Description     This function increments/decrements the actual speed
//                 reference until it becomes equal to the speed end reference.
//-----------------------------------------------------------------------------
void Speed_ref_ramp(void)
{
    if(sv_end_ref > speed_ref)                          // If the end reference hasn't b
een reached...
    {
    speed_ref += 1;                                         // ... increment speed refe
rence.
    }
    if(sv_end_ref < speed_ref)                          // If the actual reference is gr
eater than end reference...
    {
    speed_ref -= 1;                                         // ... decrement the speed
reference.
```

Page: 7

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\FUNC.C

```
    }

}

//*****************************************************************************
// @Function        void Speed_calc(void)
// Description      This function computes the actual motor speed from the
//                  measured time between two hall events or zero crossing points.
//-----------------------------------------------------------------------------
int Speed_calc(unsigned int time_meas, int* speed_counter, int limit)
{
    unsigned int  time = 0;
    unsigned long speed_cal = 0;

    speed_acumm  += time_meas;                              // Accumulate the new measu
red time.

    speed_acumm  -= speed_buffer[circ_index];               // Subtract out the oldest
sample from the accumulator.

    speed_buffer[circ_index] = time_meas;                   // Preserve the latest samp
le in the buffer.

    if(*speed_counter < limit)                              // If motor is starting...
    {
        *speed_counter = *speed_counter + 1;                    // ... not enough sampl
es.
        time = time_meas;                                       // ... use the measured
 sample directly, instead of the average.
        speed_cal = SPEED_CONSTANT/(6 * PP);                    // ... load the divisio
n factor to calculate motor speed in RPM.
    }
    else                                                    // If motor has passed ramp
 up stage....
    {                                                           // ... the accumulator
has sufficient samples.
        if((int)((long)speed_acumm >> 16)  > 0)                 // If the accumulated t
ime is too big...
        {
            time = (int)(speed_acumm >> 3);                         // ... scale it dow
n to a smaller number (lose precision).
            speed_cal = SPEED_CONSTANT >> 3;                        // ... load the div
ision factor to calculate motor speed in RPM.
        }
        else
        {                                                       // If the accumulated t
ime is not too big...
            time = (int) (speed_acumm);                         // ... don't scale
it down so that no precision is lost.
            speed_cal = SPEED_CONSTANT;                         // ... load the div
ision factor to calculate motor speed.
        }
    }
    circ_index++;                                           // Increment index for circ
ular buffer.
    if(circ_index > ((6 * PP) - 1))                         // If the end of circular b
uffer is reached...
    {
        circ_index = 0;                                         // ... reset the it to
'0'.
    }

    return ((int)(speed_cal / time));                       // Compute motor speed.

}

//*****************************************************************************
// @Function        void Sensorless_calc(void)
```

Page: 8

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\FUNC.C

```
// Description      This function maintains the state operation of the motor
//                  in case the hall sensors don't work.
//------------------------------------------------------------------------------
void Sensorless_calc(void)
{
    CH_conv = ADC1_RESR0>>12 & 0x0007;                                      // Store th
e channel converted.
    switch(CH_conv)                                                         // Check wh
ich phase is being measured.
    {
        case 1:                                                                 // Phas
e A.
            back_emf_measure  = (ADC1_RESR0>>2 & 0x03FF)-adc_offset_resA;           //
Substract the corresponding offset of phase A.
            break;
        case 4:                                                                 // Phas
e B.
            back_emf_measure  = (ADC1_RESR0>>2 & 0x03FF)-adc_offset_resB;           //
Substract the corresponding offset of phase B.
            break;
        case 7:                                                                 // Phas
e C.
            back_emf_measure  = (ADC1_RESR0>>2 & 0x03FF)-adc_offset_resC;           //
Substract the corresponding offset of phase C.
            break;
    }
    if(!slope_known)                                                        // If the s
lope is not known yet...
    {
        if(slope == 0)                                                          // If w
e are looking for a rising edge...
        {
            if(back_emf_measure >= compare_value)                               //
If the bemf is greater than half of bus voltage...
            {
                bemf_counter1++;                                                //
... increment samples counter.
                if(bemf_counter1 == 4)                                          //
If enough valid bemf samples detected...
                {
                    slope           = 1;
    // ... look for a falling edge.
                    zc_time         = CCU60_T12;
    // ... save the captured time between two bemf zero crossings (in sensorless mode).
                    bemf_counter1   = 0;
    // ... reset the counter.
                }
            }
        }
        if(slope == 1)                                                          // If w
e are looking for a rising edge...
        {
            if(back_emf_measure < compare_value)                                //
If the bemf is lower than half of bus voltage...
            {
                bemf_counter1++;                                                //
... increment samples counter.
                if(bemf_counter1 == 4)                                          //
If enough valid bemf samples detected...
                {
                    slope           = 0;
    // ... look for a falling edge.
                    zc_time         = CCU60_T12;
    // ... save the captured time between two bemf zero crossings (in sensorless mode).

                    bemf_counter1   = 0;
    // ... reset the counter.
                }
```

Page: 9

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\FUNC.C

```
            }
        }
        if(zc_time > 70)                                                    // If t
he value has sense (not just after the commutation)...
        {
        slope_known = 1;                                                     //
... the slope is already known.
        }
    }
    else                                                                  // If the s
lope is known...
    {
        if(slope == 0)                                                      // If w
e are looking for a rising edge...
        {
            if(back_emf_measure >= compare_value)                            //
If the bemf is greater than half of bus voltage...
            {

                bemf_counter1++;
    // ... increase 'bemf_counter'.

                if(bemf_counter1 == 2)
    // If two valid samples are detected...
                {
                    zc_time         = CCU60_T12;
        // ... save the captured time between two zero crossings (in sensorless mode).


                    status_bemf     = 1;
        // ... set the bemf detection status flag.
                    bemf_counter1   = 0;
        // ... reset sample counter.
                    slope           = 3;
        // ... slope = 3, so that next we start looking for falling edge.
                    if(status_hs_error)
        // If sensorless mode...
                    {
                        CCU60_CC60SR    = zc_time;
            // ... set the CC60R for speed measurement with the time between zero cross
ing points.
                        CCU60_CC61SR    = (zc_time>>1) - 5;
            // ... initialize the CCU61 register with half of the time between zero cro
ssings,

            // so that it commutates within the required time (with an advance).
                        CCU60_CC62SR    = CCU60_CC61SR + 10;
            // ... initialize CCU62 some time after to calculate the speed.
                        CCU60_T12PR     = (zc_time>>1) + zc_time;
            // ... set time-out to 1.5 times the sector time.
                        CCU60_TCTR4     = 0x0045;
            // ... T12 Reset and Stop (updated CC61R, CC62R and T12PR).
                        CCU60_TCTR4     = 0x0002;
            // ... start T12 again.
                    }
                }
            }
        }
        else if(slope == 1)                                                 // If w
e are looking for a falling edge...
        {
            if(back_emf_measure < compare_value)                             //
If the bemf is lower than half of bus voltage...
            {
                bemf_counter1++;
    // ... increase 'bemf_counter'.
                if(bemf_counter1 == 2)
    // If two valid samples are detected...
```

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\FUNC.C

```
                {
                    zc_time         = CCU60_T12;
        // ... save the captured time between two zero crossings (in sensorless mode).


                    status_bemf     = 1;
        // ... set the bemf detection status flag.
                    bemf_counter1   = 0;
        // ... reset sample counter.
                    slope           = 2;
        // ... slope = 2, so that next we start looking for rising edge.
                    if(status_hs_error)
        // If sensorless mode...
                    {
                        CCU60_CC60SR    = zc_time;
            // ... set the CC60R for speed measurement with the time between zero cross
ing points.
                        CCU60_CC61SR    = (zc_time>>1) - 5;
            // ... initialize the CCU61 register with half of the time between zero cro
ssings,

            // so that it commutates within the required time (with an advance).
                        CCU60_CC62SR    = CCU60_CC61SR + 10;
            // ... initialize CCU62 some time after to calculate the speed.
                        CCU60_T12PR     = (zc_time>>1) + zc_time;
            // ... set time-out to 1.5 times the sector time.
                        CCU60_TCTR4     = 0x0045;
            // ... T12 Reset and Stop (updated CC61R, CC62R and T12PR).
                        CCU60_TCTR4     = 0x0002;
            // ... start T12 again.
                    }
                }
            }
        }
    }
}
```

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\FUNC.H

```
// Import global variables

    // Status word
    extern C_BDATA  status_word;                    // Full status motor state.

    extern BIT status_ctrap;                        // Ctrap detection.

    // Global variables
    extern int C_SDATA direction;                   // Index-offset for reading the swi
tching pattern, depending on the specified motor direction.
    extern int C_SDATA once;                        // Counter to specify the time limi
t for motor Overcurrent.
    extern unsigned int C_SDATA sv_start_ref;       // Start speed reference for motor
start.
    extern unsigned int C_SDATA sv_end_ref;         // End speed reference for motor st
art.
    extern int C_SDATA motor_status;                // Motor stop command.
    extern int C_SDATA direction_user;              // User desired direction.
    extern unsigned long C_SDATA rampup_time;       // Time to achieve the end referenc
e.
    extern int C_SDATA trap_reset;                  // Set to '1' to reset the 'Latched
 CTRAP' condition.

    // PI variables
    extern struct PI_cont C_SDATA PI_spd;           // PI speed variable.
    extern struct PI_cont C_SDATA PI_curr;          // PI current variable.

// Import functions
    extern void Motor_start(void);
    extern void Motor_stop(void);

// Function inlines

//*************************************************************************************
************
// @Function        void INIT_PI_VAR(void)
// Description      This function initialises the PI gain & Limit variables with the co
rresponding
//                  user values in 'CONFIG.H'.
//-------------------------------------------------------------------------------------
------------
INLINE void INIT_PI_VAR(void)
{
    PI_spd.ymax  = PI_SPEED_YMAX;       // Initialise the PI upper limit variable.
    PI_spd.ymin  = PI_SPEED_YMIN;       // Initialise the PI lower limit variable.
    PI_spd.kp    = PI_SPEED_KP;         // Initialise the PI proportional gain variable
.
    PI_spd.ki    = PI_SPEED_KI;         // Initialise the PI integral gain variable.

    PI_curr.ymax  = PI_CURR_YMAX;       // Initialise the PI upper limit variable.
    PI_curr.ymin  = PI_CURR_YMIN;       // Initialise the PI lower limit variable.
    PI_curr.kp    = PI_CURR_KP;         // Initialise the PI proportional gain variable
.
    PI_curr.ki    = PI_CURR_KI;         // Initialise the PI integral gain variable.
}

//*************************************************************************************
************
// @Function        void MOTOR_START_PARAMS(void)
// Description      This Function loads the values for the motor start.
//-------------------------------------------------------------------------------------
------------
INLINE void MOTOR_START_PARAMS(void)
{
    trap_reset      =   1;                      // Set the 'trap_reset' flag so that 't
rap' flag is cleared under power ON.
    status_ctrap    =   0;                      // Clear the trap flag to allow the mot
or to run.
```

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\FUNC.H

```
    sv_end_ref      =   MOTOR_END_REF;          // Initialise the end reference with pr
edefined value.
    sv_start_ref    =   MOTOR_START_REF;        // Initialise the start reference with
predefined value.
    rampup_time     =   RAMP_UP_TIME;           // Initialise the ramp-up time with pre
defined value.
    direction_user  =   MOTOR_DIRECTION;        // Initialise the desired direction.
    direction       =   direction_user * 8;     // Initialise the motor direction and m
ultiplies by 8 to select the apropiate commutation pattern.
    motor_status    =   0;                      // Clear the motor status flag to allow
 start.
}

//*************************************************************************************
************
//*************************************************************************************
************
// @Function        void INIT_CTRL_VAR(void)
// Description      This function initialises the various intermediate variables that a
re required to
//                  be set to pre-defined default values before any motor related opera
tion.
//-------------------------------------------------------------------------------------
------------
INLINE void INIT_CTRL_VAR(void)
{
    direction        =  0x0;        // Initialise default motor direction to clockwise.
    once             =  0x0;        // Clear the variable 'once' so that by default mot
or stop is assumed.
    status_word      =  0x0002;     // Initialise status flags to reflect motor stopped
 condition.
    INIT_PI_VAR();                  // Execute PI initialisation routine.
    MOTOR_START_PARAMS();           // Execute control panel dependent variable initial
isation routine.
}

//*************************************************************************************
************
// @Function        void TRAP_RESET(void)
// Description      This Function will reset the trap flag if requested.
//-------------------------------------------------------------------------------------
------------
INLINE void TRAP_RESET(void)
{
    if (trap_reset == 1)        // If trap reset requested...
    {
        trap_reset      = 0;        // ... acknowledge the request.
        status_ctrap    = 0;        // ... reset the trap flag.
    }
}

//*************************************************************************************
************
// @Function        void MOTOR_OPERATION(void)
// Description      This Function will start or stop the motor depending on present inp
uts.
//-------------------------------------------------------------------------------------
------------
INLINE void MOTOR_OPERATION(void)
{
    if (status_ctrap)               // If trap flag is set...
    {
        motor_status = 1;               // ... stop de motor.
    }
    else                            // If trap flag is not set...

    {
        if (motor_status == 0)          // If the start of the motor is requested...
```

Page: 2

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\FUNC.H

```
        {
            Motor_start();                  // ... start the motor.
        }
        else                              // If the start of the motor is not requested..
.
        {
            Motor_stop();                   // ... stop the motor.
        }
    }
}
```

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\CONFIG.H

```
// Define parameters and configuration

    // Speed PI
    #define PI_SPEED_YMAX       16368           // Corresponding to an ADC read of 7.5
A (shifted to 1Q15) [ADC range 0 A - 15 A, nominal current = 7 A]
    #define PI_SPEED_YMIN       0               // The minimum output of the speed PI i
s 0 A
    #define PI_SPEED_KP         358             // Kp has to be divided by 64, to scale
 it for the output calculation, where it is multiplied by 64    (0.7 in 1Q15 divided by
 64)
    #define PI_SPEED_KI         229             // Ki parameter of the speed PI control
ler in per unit (0.007 in 1Q15)

    // Current PI
    #define PI_CURR_YMAX        32112           // Corresponding to 98% of duty cycle [
0.98 to 1Q15]
    #define PI_CURR_YMIN        0               // Corresponding to 0% of duty cycle
    #define PI_CURR_KP          205             // Kp has to be divided by 64, to scale
 it for the output calculation, where it is multiplied by 64    (0.4 in 1Q15 divided by
 64)
    #define PI_CURR_KI          16              // Ki parameter of the current PI contr
oller in per unit (0.0005 in 1Q15)

    // Motor parameters
    #define PP                  4               // Pole pairs
    #define MAX_SPEED           9000            // Max. speed that the motor can achiev
e without field weakening (No load speed)

    // User configuration
    #define MOTOR_DIRECTION     0               // Clockwise = 0, Counterclockwise = 1
    #define RAMP_UP_TIME        1600000         // Time to achieve the speed end refere
nce (value = time*64*fpwm = 1s*64*25000s-1
    #define MOTOR_START_REF     600             // Start reference of the motor speed
    #define MOTOR_END_REF       3000            // End reference of the motor speed
    #define TIME_CURRENT_MAX    1000            // Maximum time overcurrent (up to 10 A
) is allowed before stopping the motor [TIMER_CURRENT_MAX=tmax/T13CM=40ms/40us]

    // Speed calculation
    #define SPEED_CONSTANT      15468750        // SPEED_CONSTANT=(60s/min)/(Step size
of T12)

    // Sensorless
    #define BEMF_AUX_VAL11  0x18                // Phase A inactive (B+ C-)
    #define BEMF_AUX_VAL12  0x24                // Phase A inactive (C+ B-)
    #define BEMF_AUX_VAL21  0x12                // Phase B inactive (A+ C-)
    #define BEMF_AUX_VAL22  0x21                // Phase B inactive (C+ A-)
    #define BEMF_AUX_VAL31  0x06                // Phase C inactive (A+ B-)
    #define BEMF_AUX_VAL32  0x09                // Phase C inactive (B+ A-)
```

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\DSPLIB_KEIL.H

```
/*****************************************************************************
*   Module      Header File
*   Filename    DspLib_Keil.h
*   Project     DSP library for C166S V2 Core
*----------------------------------------------------------------------------
*   Compiler:   Keil
*
*   Version:    V 1.0
*
*   Description:
*               This header file lists
*               1. Typedef statements
*               2. All the functions of DSP Library
*----------------------------------------------------------------------------
*   Date:       March 2008
*****************************************************************************/


#ifndef _DSPLIB_KEIL_H_
#define _DSPLIB_KEIL_H_

// #include <XC161.h>               /* special function register XC161   */


typedef short   DataS;          //16-bit data
typedef long    DataL;          //32-bit data

//****************************************************************************
// This contains the user defined data types for the complex functions
//----------------------------------------------------------------------------

typedef struct
{
    short real;
    short imag;
}CplxS;

typedef struct
{
    long real;
    long imag;
}CplxL;


//****************** Arithmetic Operations ***********************************
extern void CplxAdd_16(
                    CplxS*  X,          // first 1Q15 data
                    CplxS*  Y,          // second 1Q15 data
                    CplxS*  R           // output
                    );
extern void CplxSub_16(
                    CplxS*  X,          // first 1Q15 data
                    CplxS*  Y,          // second 1Q15 data
                    CplxS*  R           // output
                    );
extern void  CplxMul_16(
                    CplxS*  X,          // first 1Q15 data
                    CplxS*  Y,          // second 1Q15 data
                    CplxS*  R           // result
                );
extern DataL Mul_32(
                 DataL*     X,          // first 1Q15 data
                 DataL*     Y           // second 1Q15 data
                );

//***************************** FIR Filters **********************************
extern DataS Fir_16(
                DataS*  H,          // pointer to filter coefficient vector
```

Page: 1

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\DSPLIB_KEIL.H

```
                DataS*  IN,          // the new input sample
                DataS   N_h,         // Filter order
                DataS*  D_buffer     // pointer to delay buffer
                );
extern DataS Fir_32(
                DataL*  H,           // pointer to filter coefficient vector
                DataS*  IN,          // the new input sample
                DataS   N_h,         // Filter order
                DataS*  D_buffer     // pointer to delay buffer
                );
extern void Fir_dec(
                DataS*  x,           // pointer to input x
                DataS*  h,           // pointer to coefficent vector h
                DataS*  y,           // pointer to output y
                DataS*  d_buffer,    // pointer to delay buffer
                DataS   N_x,         // size of x
                DataS   N_h,         // filter order
                DataS   D            // decimating factor
                );
extern void Fir_inter(
                DataS*  x,           // pointer to input x
                DataS*  h,           // pointer to coefficent vector h
                DataS*  y,           // pointer to output y
                DataS*  d_buffer,    // pointer to delay buffer
                DataS   N_x,         // size of x
                DataS   N_h,         // filter order
                DataS   I            // interpolating factor
                );
extern void Fir_lat(
                DataS*  x,           // pointer to input x
                DataS*  K,           // pointer to lattice coefficent vector K
                DataS*  y,           // pointer to output y
                DataS*  u,           // pointer to delay buffer
                DataS   N_x,         // size of x
                DataS   M            // filter nummber of stages
                );
extern void Fir_sym(
                DataS*  x,           // pointer to input x
                DataS*  h,           // pointer to coefficent vector h
                DataS*  y,           // pointer to output y
                DataS*  d_buffer,    // pointer to delay buffer
                DataS   N_x,         // size of x
                DataS   N_h          // half of filter order
                );
extern void Fir_cplx(
                CplxS*  x,           // pointer to complex input x
                CplxS*  h,           // pointer to complex coefficent vector h
                CplxS*  y,           // pointer to complex output y
                DataS*  d_buffer,    // pointer to delay buffer
                DataS   N_x,         // size of x
                DataS   N_h          // half of filter order
                );
extern void Fir_16_Blk(
                DataS*  H,           // pointer to filter coefficient vector
                DataS*  IN,          // the new input sample
                DataS*    R,           //Pointer to output buffer
                DataS*  D_buffer,    // pointer to delay buffer
                DataS   N_h,         // Filter order
                DataS   N_x
                );

//*************************** IIR Filters ************************************
extern DataS IIR_1(
                DataS*  B_A,     // pointer to filter coefficients
                DataS*  IN,      // the new input sample
                DataS   N,       // Filter order for input X
                DataS*  x_y      // delay buffer
                );
```

Page: 2

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\DSPLIB_KEIL.H

```
extern DataS IIR_2(
                DataS*  h,      // pointer to filter coefficients
                DataS*  IN,     // the new input sample
                DataS   N,      // Filter order
                DataS*  u       // state variable vector
                );
extern void IIR_bi_1(
                DataS*  x,          // pointer to complex input x
                DataS*  h,          // pointer to complex coefficent vector h
                DataS*  y,          // pointer to complex output y
                DataS*  u_w,        // pointer to state veriables
                DataS   N_x,        // size of x
                DataS   N_biq       // number of the biquads
                );
extern DataS IIR_bi1_smpl(
                DataS*  h,          // pointer to complex coefficent vector h
                DataS*  x,          // pointer to complex input x
                DataS   N_biq,      // number of the biquads
                DataS*  u_w     // pointer to state veriables
                );
extern void IIR_bi_2(
                DataS*  x,          // pointer to complex input x
                DataS*  h,          // pointer to complex coefficent vector h
                DataS*  y,          // pointer to complex output y
                DataS*  u,          // pointer to state veriables
                DataS   N_x,        // size of x
                DataS   N_biq       // number of the biquads
                );
extern DataS IIR_bi2_smpl(
                DataS*  h,          // pointer to complex coefficent vector h
                DataS*  x,          // pointer to complex input x
                DataS   N_biq,      // number of the biquads
                DataS*  u           // pointer to state veriables
                );
extern void IIR_bi2_32(
                DataS*  x,          // pointer to complex input x
                DataL*  h,          // pointer to complex 32-bit coefficent vector h
                DataS*  y,          // pointer to complex output y
                DataS*  u,          // pointer to state veriables
                DataS   N_x,        // size of x
                DataS   N_biq       // number of the biquads
                );
extern void IIR_lat(
                DataS*  x,          // pointer to input x
                DataS*  K,          // pointer to lattice coefficent vector K
                DataS*  y,          // pointer to output y
                DataS*  u,          // pointer to delay buffer
                DataS   N_x,        // size of x
                DataS   N_h         // filter order
                );
extern void IIR_bi_24(
                DataS*  x,          // pointer to complex input x
                DataS*  h,          // pointer to complex coefficent vector h
                DataS*  y,          // pointer to complex output y
                DataS*  u,          // pointer to state veriables
                DataS   N_x,        // size of x
                DataS   N_biq       // number of the biquads
                );
extern void IIR_bi_2_Blk(
                DataS*  IN,         // the new input sample
                DataS*  h,          // pointer to filter coefficients
                DataS*  Y,          // the Output sample
                DataS*  u_w,            // state variable vector
                DataS   N_biq ,         // number of the biquads
                DataS   N_x         // number of input and output samples
                );
//*************************** Mathematical functions ***************************
extern DataS P_series(
```

Page: 3

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\DSPLIB_KEIL.H

```
                DataS*  a,      // pointer to coefficients
                DataS*  IN,     // pointer to the input value
                DataS   N       // the order of series
                );
extern void Windowing(
                DataS*  h,      // pointer to coefficients
                DataS*  x,      // pointer to the input value vector
                DataS*  y,      // poniter to output vector
                DataS   N       // the length of the window
                );
extern DataS Sine(
                        DataS x           //input
                );
extern DataS div_q15(
                        DataS     x,      // dividend
                        DataS     y       // divisor
                );
extern DataS div_q31(
                        DataL     x,      // dividend
                        DataS     y       // divisor
                );
extern DataS Sqrt(  DataS   x );

//***************************** Matrix *****************************************
extern void  Matrix_mul(
                DataS*  x_1,    // pointer to 1. input matrix
                DataS*  x_2,    // pointer to 2. input matrix
                DataS*  y,      // poniter to output matrix
                DataS   row1,   // number of rows in matrix 1
                DataS   col1,   // number of columns in matrix 1
                DataS   row2,   // number of rows in matrix 2
                DataS   col2    // number of columns in matrix 2
                );
extern void  Matrix_trans(
                DataS*  x,      // pointer to input matrix
                DataS*  y,      // poniter to output matrix
                DataS   row,    // number of rows in matrix
                DataS   col     // number of columns in matrix
                );

//***************************** Adaptive Filters *******************************
extern DataS Adap_filter_16(
                DataS*  h,          // pointer to coeffcient vector h with 16 bits
                DataS*  IN,         // pointer to the new input sample
                DataS*  D,          // pointer to expected signal at time n
                DataS*  error,      // pointer to error signal
                DataS   N_h,        // filter order
                DataS   Step,       // adaptive gain
                DataS*  d_buffer    // delay buffer
                );


extern DataS Adap_filter_32(
                DataL*  h,          // pointer to coeffcient vector h with 32 bits
                DataS*  IN,         // pointer to the new input sample
                DataS*  D,          // pointer to expected signal at time n
                DataS*  error,      // pointer to error signal
                DataS   N_h,        // filter order
                DataS   Step,       // adaptive gain
                DataS*  d_buffer    // delay buffer
                );

//***************************** FFT Transform **********************************
extern DataS FloatTo1Q15(
                float   x               // floating input value
                );
extern float Q15toFloat(
                DataS   x               //1Q15 input
```

Page: 4

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\DSPLIB_KEIL.H

```
                );
extern DataS Bit_reverse(
                DataS*  X,      // index input vector
                DataS   N       // the size of the vector
                );
extern void real_DIT_FFT(
                DataS*  x,          // inpu vector
                DataS*  index,      // bit reversed input indecies
                DataS   exp,        // exponent of vector size
                DataS*  table,      // trigonomic function table
                DataS*  X           // output of FFT
                );
extern void real_DIF_IFFT(
                DataS*  x,          // output of IFFT
                DataS*  index,      // bit reversed input indecies
                DataS   exp,        // exponent of vector size
                DataS*  table,      // trigonomic function table
                DataS*  X           // inpu vector
                );
extern void real_DIT_IFFT(
                DataS*  x,          // output of IFFT
                DataS*  index,      // bit reversed input indecies
                DataS   exp,        // exponent of vector size
                DataS*  table,      // trigonomic function table
                DataS*  X           // inpu vector
                );
//***************************** Statistical functions *************************
extern void Auto_raw(
                DataS*  x,          // Pointer to input vector in 1Q15 format
                DataS*  y,          // Pointer to output vector in 1Q15 format
                DataS   N_x,        // Size of input vector
                DataS   N_y         // Size of output vector, N_y <= N_x
              );
extern void Auto_bias(
                DataS*  x,          // Pointer to input vector in 1Q15 format
                DataS*  y,          // Pointer to output vector in 1Q15 format
                DataS   N_x,        // Size of input vector
                DataS   N_y         // Size of output vector, N_y <= N_x
              );
extern void Auto_unbias(
                DataS*  x,          // Pointer to input vector in 1Q15 format
                DataS*  y,          // Pointer to output vector in 1Q15 format
                DataS   N_x,        // Size of input vector
                DataS   N_y         // Size of output vector, N_y <= N_x
              );
extern void Cross_raw(
                DataS*  x1,         // Pointer to the first input vector in 1Q15 format
                DataS*  x2,         // Pointer to the second input vector in 1Q15 forma
t
                DataS*  y,          // Pointer to output vector in 1Q15 format
                DataS   N_x1,       // Size of first input vector
                DataS   N_x2        // Size of second input vector, N_x2 <= N_x1
              );
extern void Cross_bias(
                DataS*  x1,         // Pointer to the first input vector in 1Q15 format
                DataS*  x2,         // Pointer to the second input vector in 1Q15 forma
t
                DataS*  y,          // Pointer to output vector in 1Q15 format
                DataS   N_x1,       // Size of first input vector
                DataS   N_x2        // Size of second input vector, N_x2 <= N_x1
              );
extern void Cross_unbias(
                DataS*  x1,         // Pointer to the first input vector in 1Q15 format
                DataS*  x2,         // Pointer to the second input vector in 1Q15 forma
t
                DataS*  y,          // Pointer to output vector in 1Q15 format
                DataS   N_x1,       // Size of first input vector
                DataS   N_x2        // Size of second input vector, N_x2 <= N_x1
```

Page: 5

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\DSPLIB_KEIL.H

```
              );
extern void Convol(
                DataS*  x,          // pointer to vector x
                DataS*  h,          // pointer to input vector x
                DataS*  y,          // pointer to output
                DataS*  d_buffer,   // pointer to delay buffer
                DataS   N_x,        // size of vector x
                DataS   N_h         // size of vector h
              );
//------------------- END OF FILE --------------------------------------------
#endif
```

Page: 6

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\FloatTo1Q15.c

```
/*****************************************************************************
;  Module:      FloatTo1Q15
;  Filename:    FloatTo1Q15.c
;  Project:     DSP library for C166S V2 Core
;-----------------------------------------------------------------------------
;  Compiler:    Keil
;
;  Version:     V1.0
;
;  Description: Changing the float format to 1Q15 format
;
;  Date:        Sept 2007
;
;  Copyright:   Infineon Technologies AG
;*****************************************************************************/

/*****************************************************************************
; DataS FloatTo1Q15(float   X);
;
; INPUTS:
;       X       the input value in float format
;
; RETURN:
;       Y       the output value in 1Q15 format
;
; ALGORITHM: According to the IEEE-754 format
;
; REGISTER USAGE:
;   1.  From .c file to .asm file:
;           (R8)=mmmmmmmmmmmmmmmm
;           (R9)=seeeeeeeemmmmmmm (s:sign; e:exponent; m:mantissa)
;
;   2.  From .asm file to .c file:
;           (R4)=y
;
; Assumption:
;
;*****************************************************************************/

#include "DspLib_Keil.h"

DataS FloatTo1Q15(float x)
{
   __asm
  {
//read the exponent bits in R3
    MOV     R3,R9       ;(R3)=R9
    SHL     R3,#1h      ;(R3)<<1
    SHR     R3,#8h      ;(R3)>>8 = e
    SUB     R3,#7fh
    JMPR    cc_ULT,next ;if((R3)<127), jump
//if x=1
    MOV     R4,#07fffh  ;(R4)=32767, in the case x=1
//read sign bit
    MOV     R2,R9       ;(R2)=(R9)
    SHR     R2,#0fh     ;(R2)=(R2)>>15 =s
    JMPR    cc_Z,ende
//if x=-1
    MOV     R4,#8000h   ;(R4)=32768, in the case x=-1

    RET

next:
//read the mantissa in R4
    MOV     R4,R9       ;(R4)=(R9)
    SHL     R4,#9h      ;(R4)<<9
    SHR     R4,#1h      ;(R4)>>1
    SHR     R8,#8h      ;(R8)>>8
```

Page: 1

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\FloatTo1Q15.c

```
    ADD     R4,R8       ;(R4)=(R4)+(R8)
    SHR     R4,#1h      ;(R4)=(R4)>>1
    ADD     R4,#4000h   ;(R4)=(R4)+2^14

//output in 1Q15 format
    ADD     R3,#1h      ;(R3)=(R3)+1
    NEG     R3          ;(R3)=-(R3)
    CMP     R3,#0fh
    JMPR    cc_SLT,next1;if (R3)<15, jump
    MOV     R4,#0h;

    RET

next1:
    SHR     R4,R3
//read the sign bit R2
    MOV     R2,R9       ;(R2)=(R9)
    SHR     R2,#0fh     ;(R2)=(R2)>>15 =s
    JMPR    cc_Z,ende
    NEG     R4

ende:

    RET

 }
}
//------------------- END OF FILE ---------------------------------------------
```

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\PI_CTRL.C

```
//*****************************************************************************/
// @Module         PI Controller
// @Filename       PI_CTRL.C
// @Controller     XC16x Series
//
//
// @Description    This file contains functions for PI Control algorithms
//
//*****************************************************************************

//*****************************************************************************
// @Function
//
// int pi_controller64(long *pi_parameter,int reference,int actual)
//
//-----------------------------------------------------------------------------
// @Description
//
//  PI-Controller
//      derived from transfer function G = kp + 1/(p*Ti)
//
//      e(k)   = reference - actual          T0 =
//  yn(k+1)= yn(k)   + ki * e(k)             ki = T0/Ti
//  y(k+1) = yn(k+1) + kp * e(k) * 64
//-----------------------------------------------------------------------------
// Computing time  42 CPU-cycle
//
//-----------------------------------------------------------------------------
// @Returnvalue
//
//-----------------------------------------------------------------------------
// @Parameters
//
//--------------------------- Arguments -------------------------------------
//  reference : reference value
//  actual    : actual value
//
//  struct pi_parameter
//      {
//      int yn;   Integral buffer
//      int kp;   Proportional Constant
//      int ki;   Integral Constant
//      int ymax; Limit value max
//      int ymin; Limit value min
//      };
//-----------------------------------------------------------------------------
//  Register modified:
//
//  R1,R3,R4,R5,R6,R7,R8,R9,R10
//
//-----------------------------------------------------------------------------
//
//*****************************************************************************

sfr  MCW      = 0xFFDC;       //MAC Control Word

int pi_controller64(long *pi_parameter,int reference,int actual)
{
long a;
#pragma asm

        mov     R12,MCW         ;Save MCW register
        mov     MCW,#1536       ;Set saturation and shift left
        mov     R11,ZEROS       ;Load zero in R11
        CoLOAD  R11,R9          ;Load Accumulator (High) with R9 (reference)
        CoSUB   R11,R10         ;error = reference - actual
        CoSTORE R9 ,MAS         ;Load error in R9
        mov     R3,[R8+]        ;
```

Page: 1

C:\Users\Pablo\Documents\Universidad\PFC\Software\Definitivo\PI_CTRL.C

```
        CoLOAD  R3,[R8+]        ;Load yn (integral buffer) in accumulator
        mov     R1,R8           ;Save parameters addres in R1
        mov     R5,[R1+]        ;Load Kp (proportional Constant) in R5
        mov     R6,[R1+]        ;Load Ki = T0/Ti (integral Constant) in R6
        CoMAC   R6,R9           ;yn = Ki * error + yn
        mov     R6,[R1+]        ;Load ymax (limit value max)
        mov     R7,[R1+]        ;Load ymin (limit value min)
        CoMIN   R11,R6          ;Limit max yn
        CoMAX   R11,R7          ;Limit min yn
        CoSTORE R4,MAH          ;Store yn-high in R4
        CoSTORE R3,MAL          ;Store yn-low  in R3
        mov     [-R8],R4        ;Store R4 in integral buffer(High)
        mov     [-R8],R3        ;Store R3 in integral buffer(Low)
        CoMUL   R5,R9           ;Kp * error
        CoSHL   #6              ;64 * Kp * error
        CoADD   R3,R4           ;y = yn + (64 * Kp * error)
        CoMIN   R11,R6          ;Limit max y
        CoMAX   R11,R7          ;Limit min y
        CoSTORE R4,MAS          ;Store y-high in R4 (return register)
        mov     MCW,R12         ;Restore MCW register

#pragma endasm

a = reference;
a = actual;
a = *pi_parameter;

}
```

Page: 2

Appendix B.

Flowcharts

B.1. Flowcharts of the control process

In this appendix are displayed the flowcharts of the control process, to provide an easier way to understand the whole process and the purpose of different variables and functions.

The flowcharts are sorted by order of appearance if starting the code by the *main.c*.

The index of these flowcharts can be found in the list of figures.

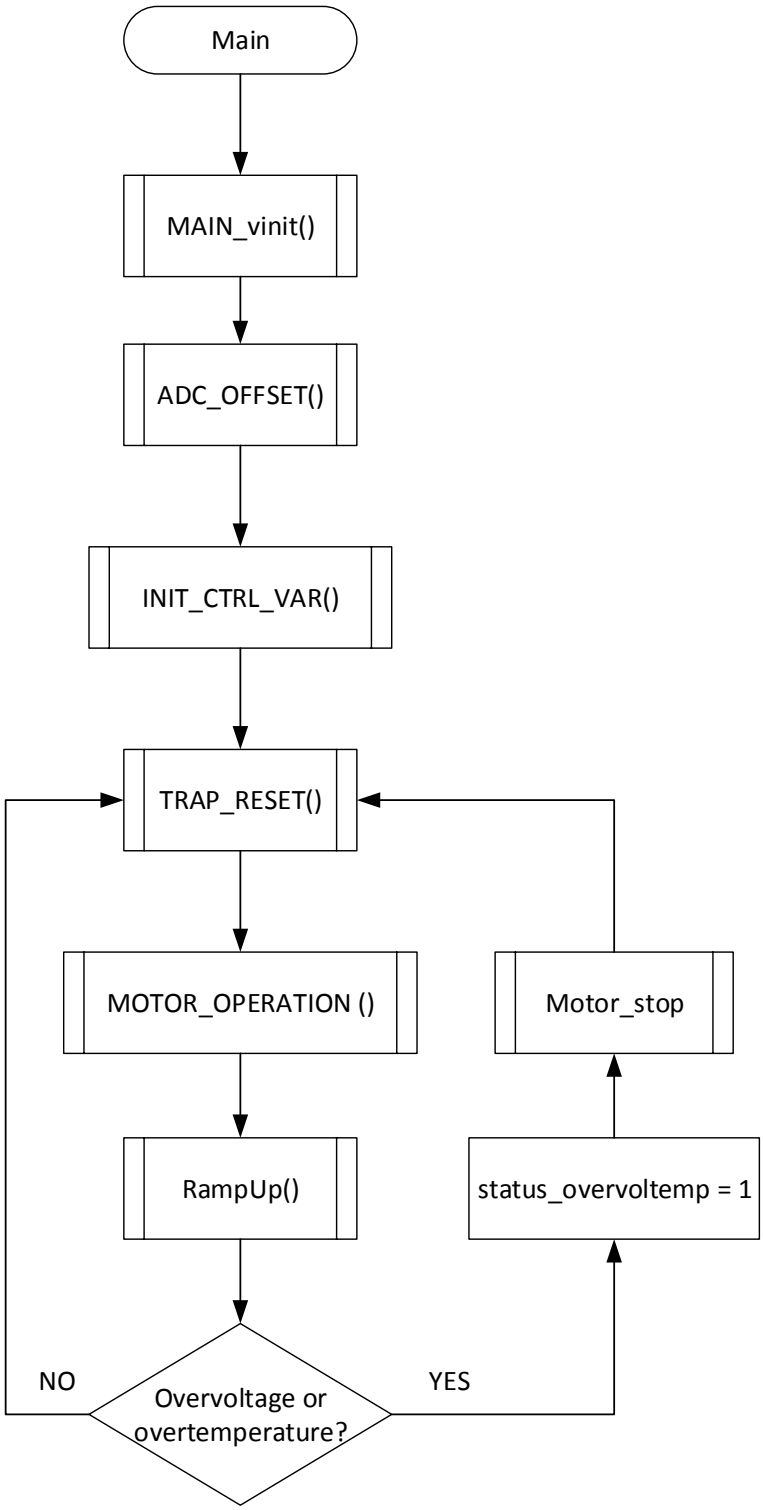


Figure B.1.: Flowchart of Main function

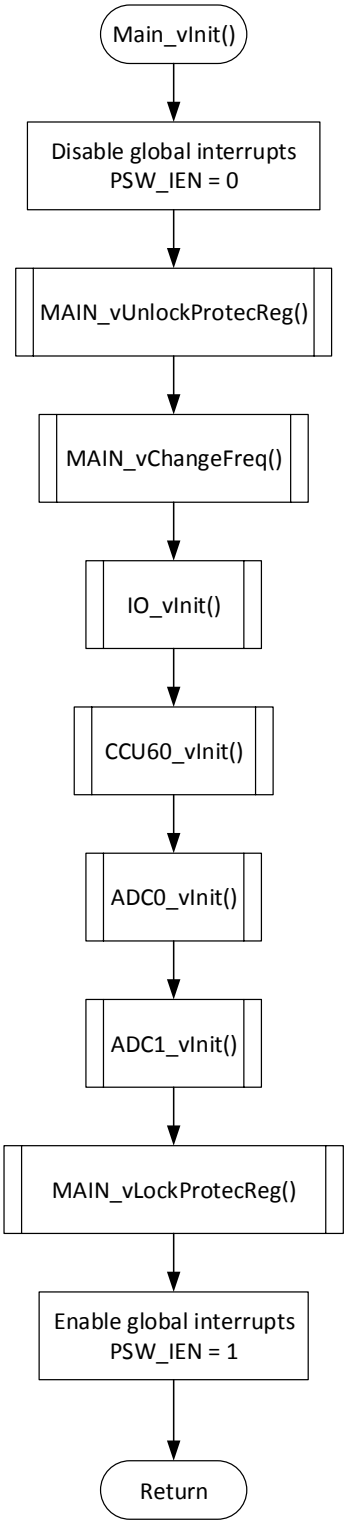


Figure B.2.: Flowchart of Main_vInit

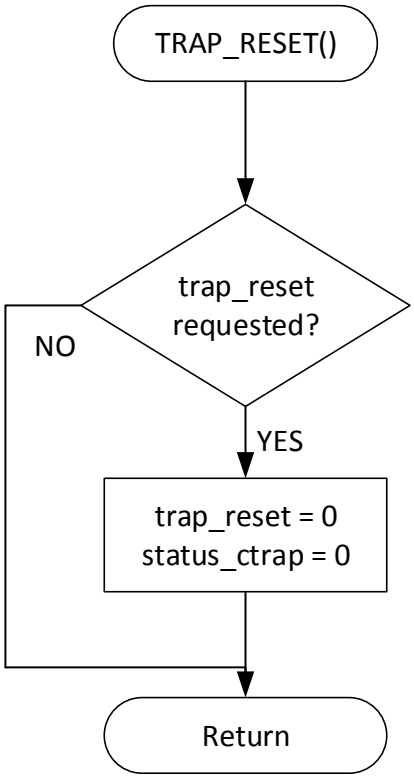


Figure B.3.: Flowchart of TRAP_RESET

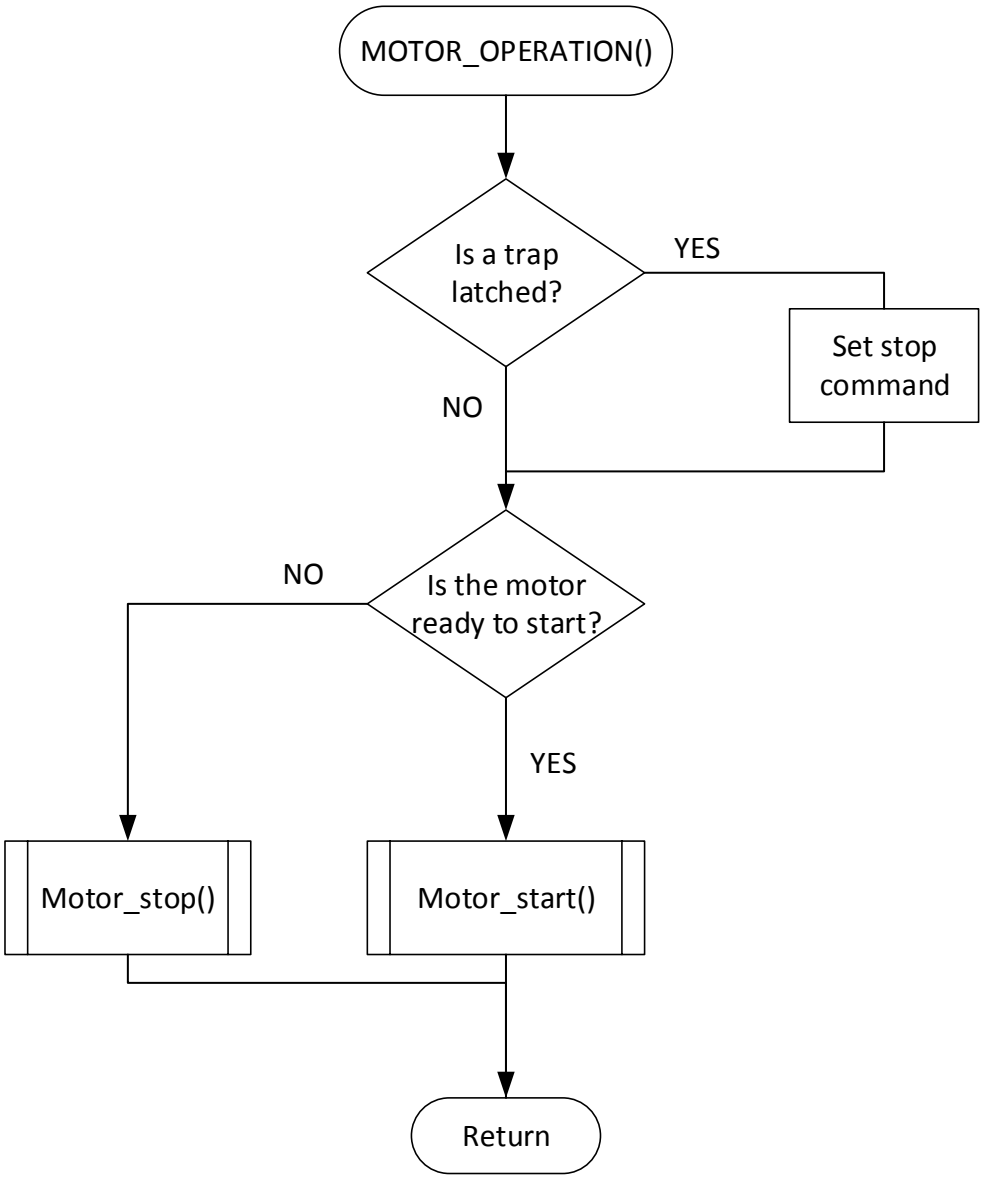


Figure B.4.: Flowchart of MOTOR_OPERATION

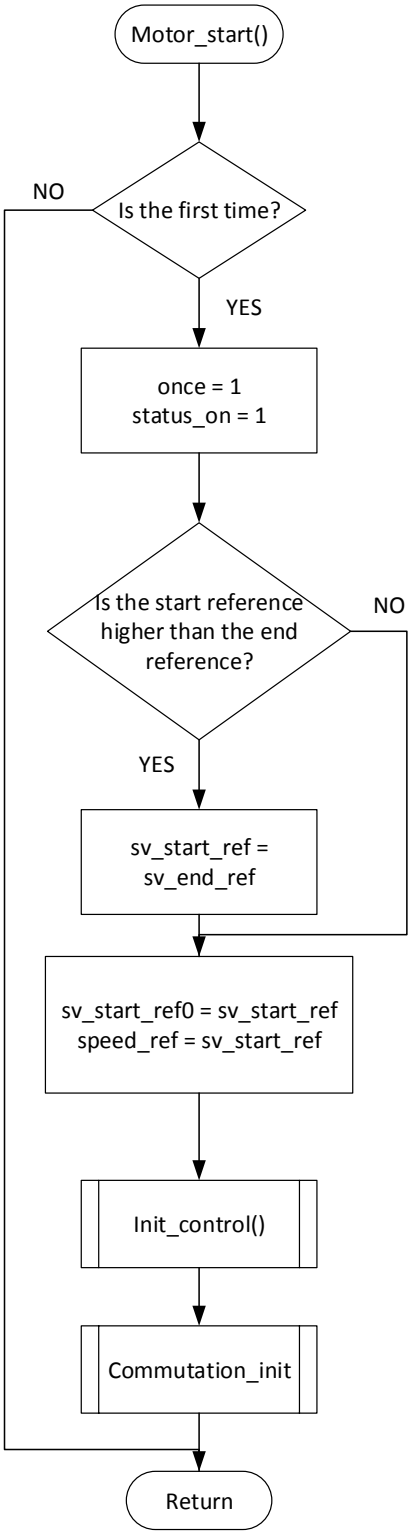


Figure B.5.: Flowchart of Motor_start

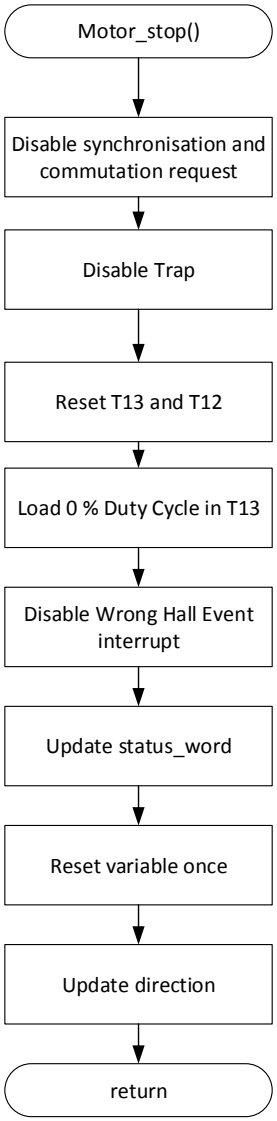


Figure B.6.: Flowchart of Motor_stop

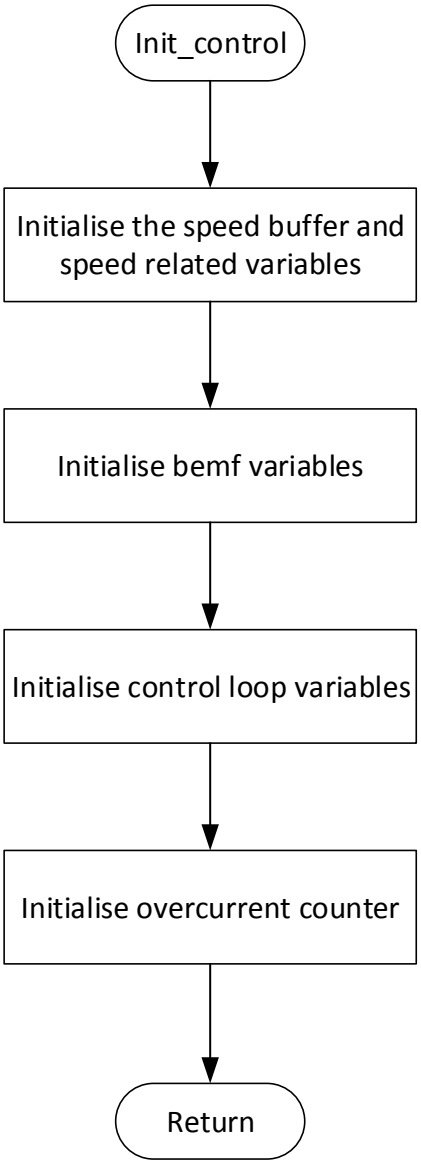


Figure B.7.: Flowchart of Init_control

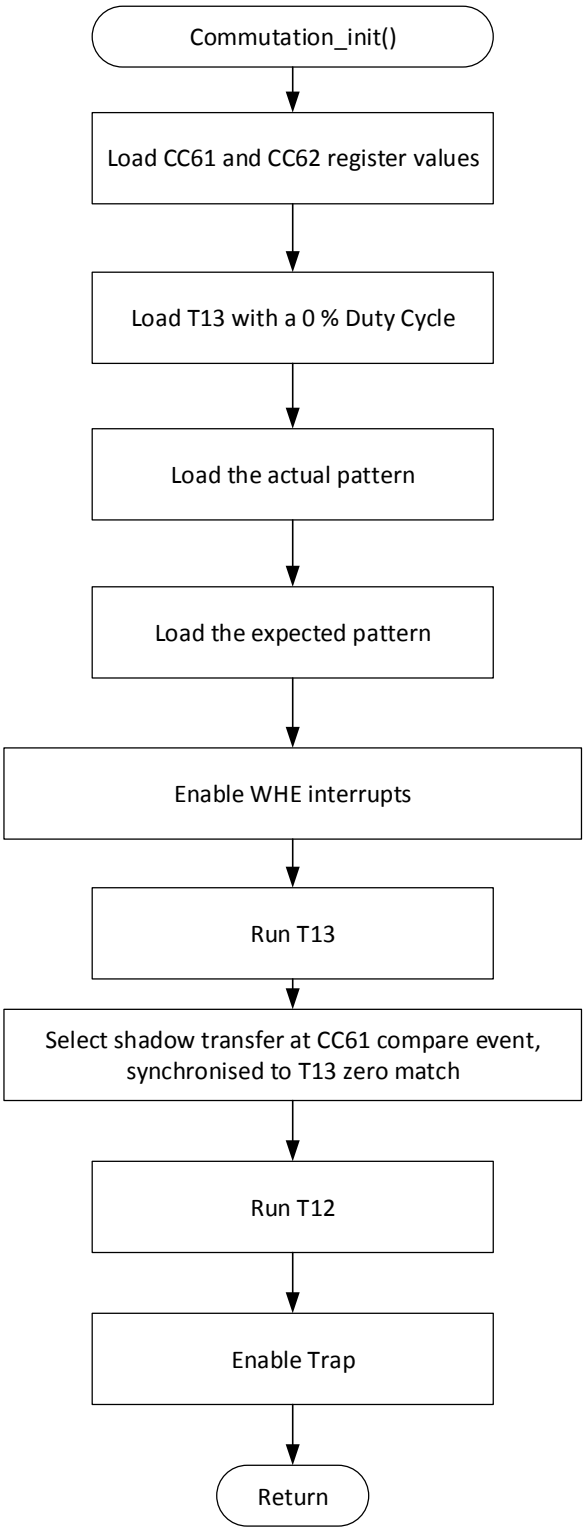


Figure B.8.: Flowchart of Commutation_init

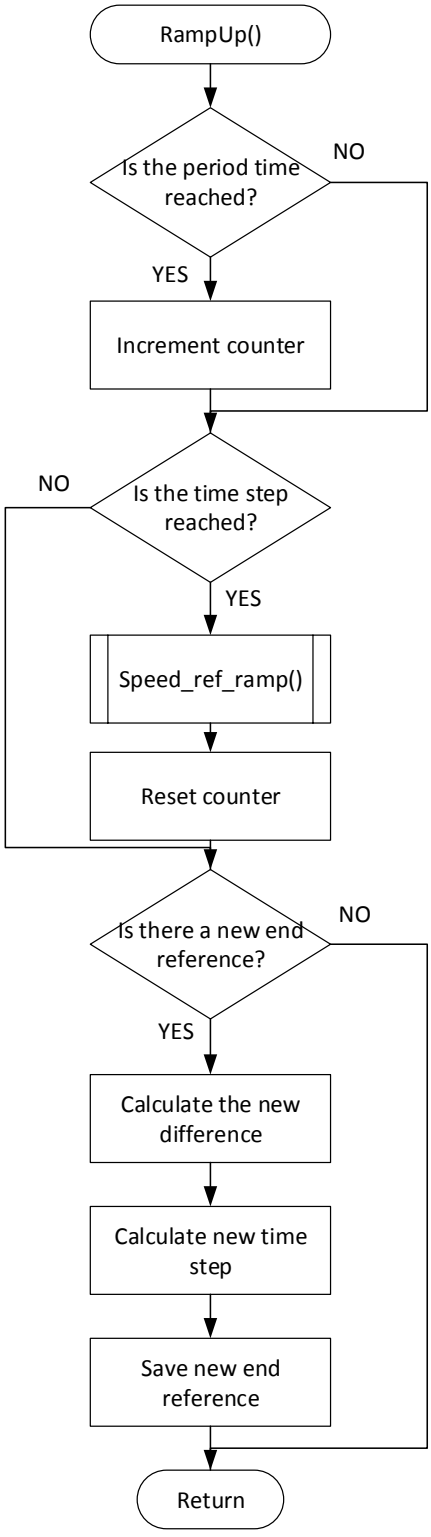


Figure B.9.: Flowchart of RampUp

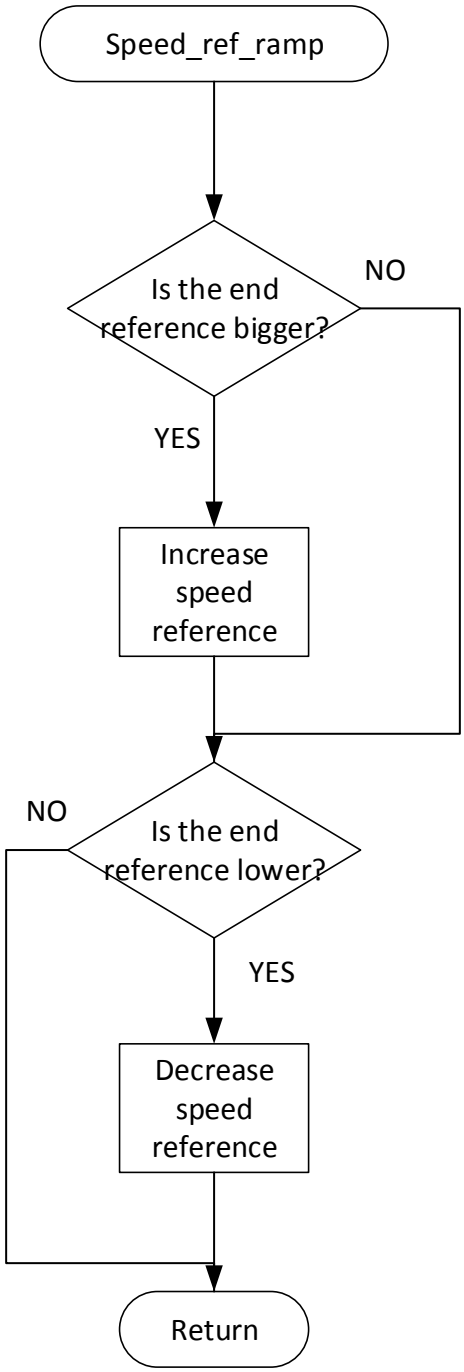


Figure B.10.: Flowchart of Speed_ref_ramp

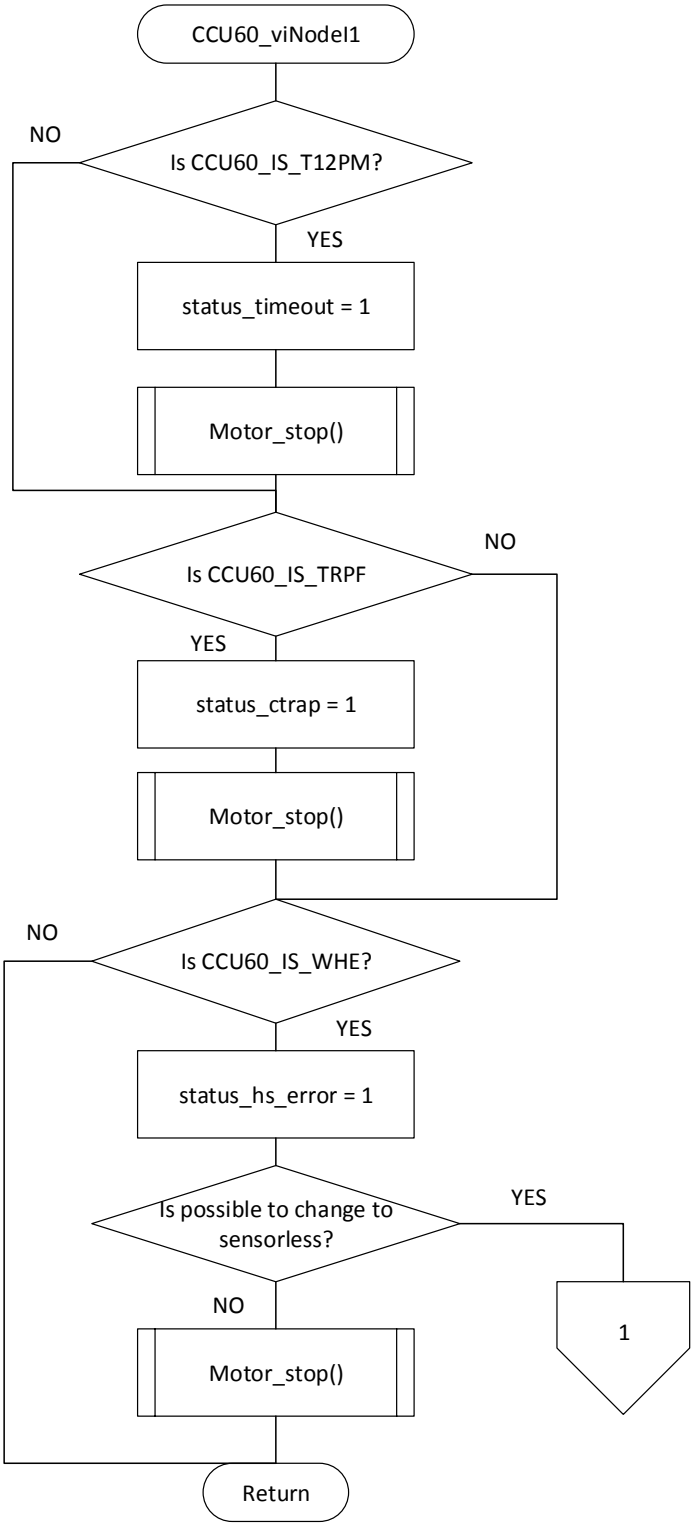


Figure B.11.: Flowchart of CCU60_viNodeI1

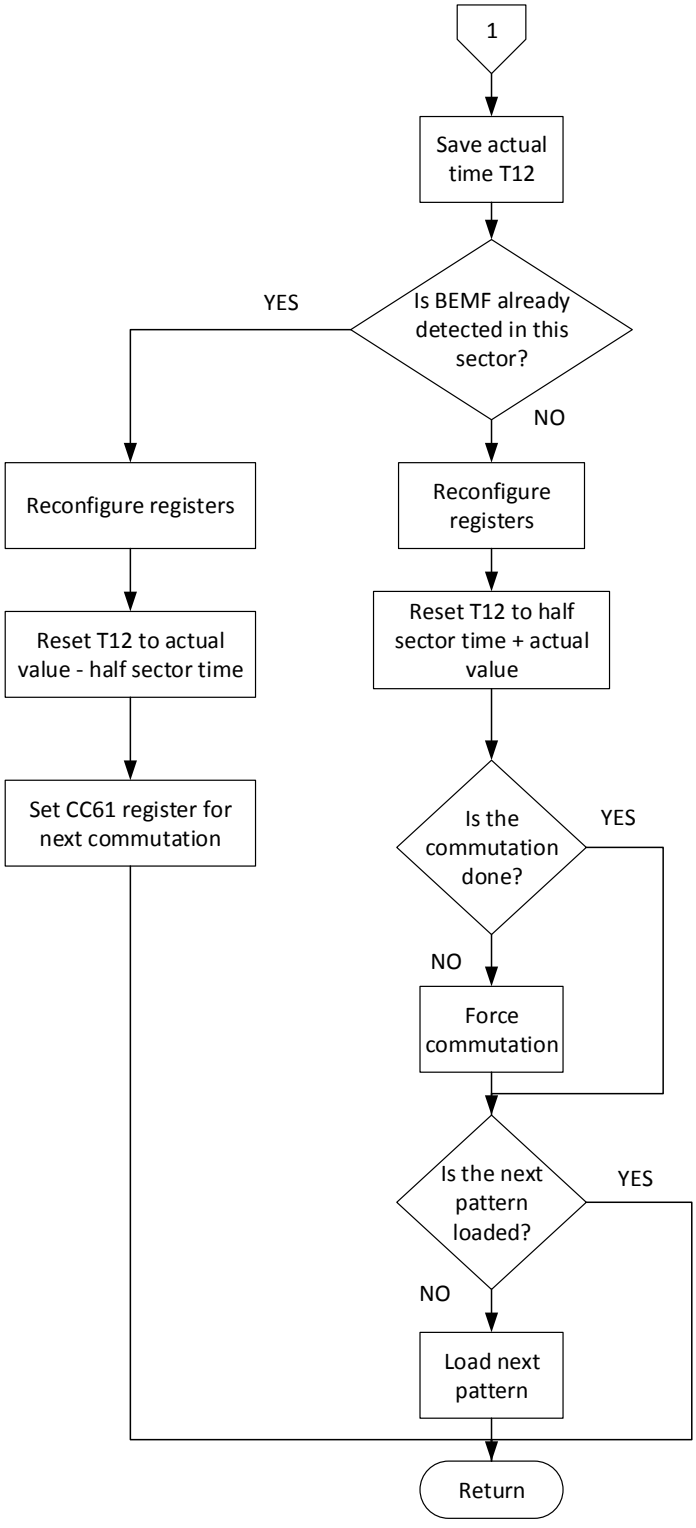


Figure B.12.: Flowchart of Transition

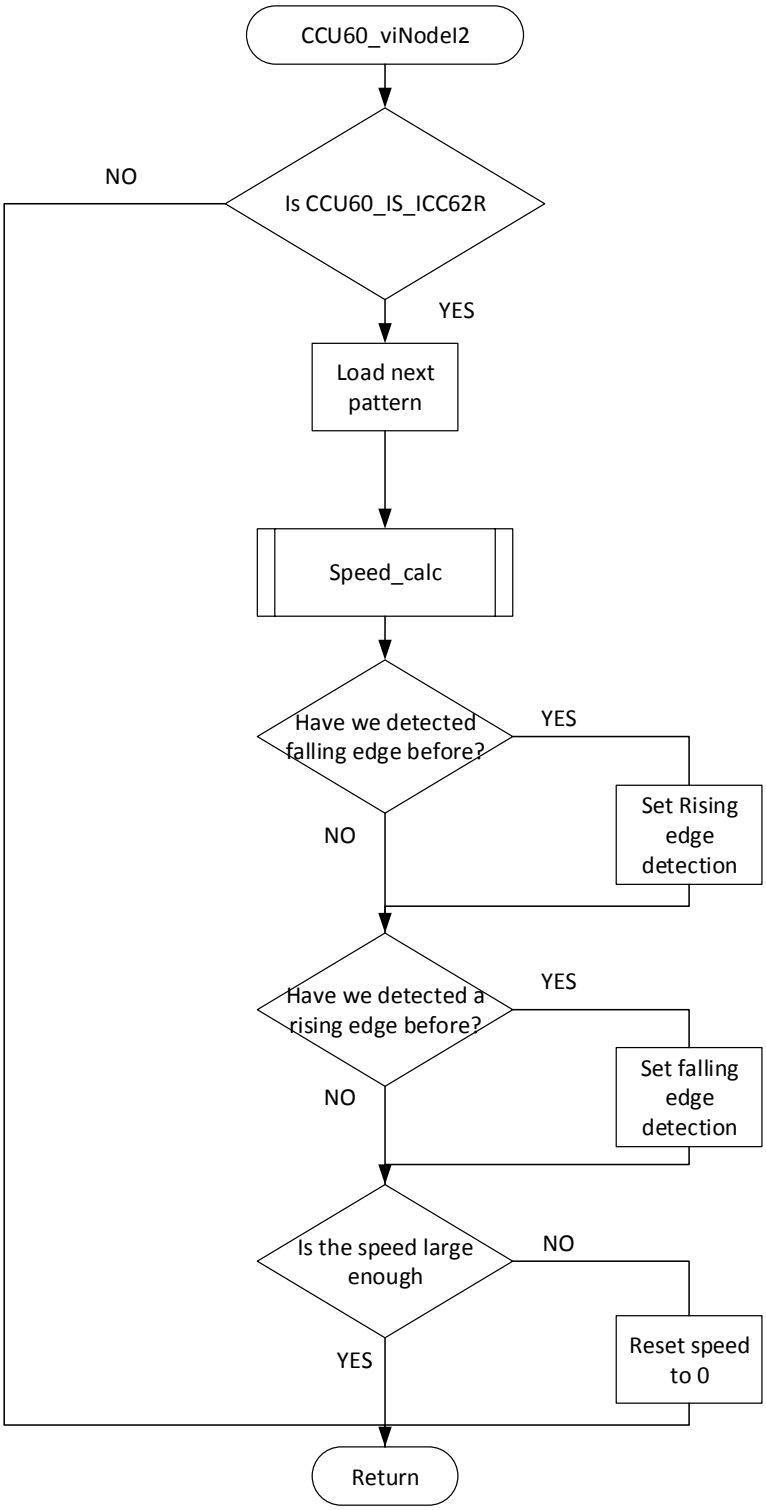


Figure B.13.: Flowchart of CCU60_viNodeI2

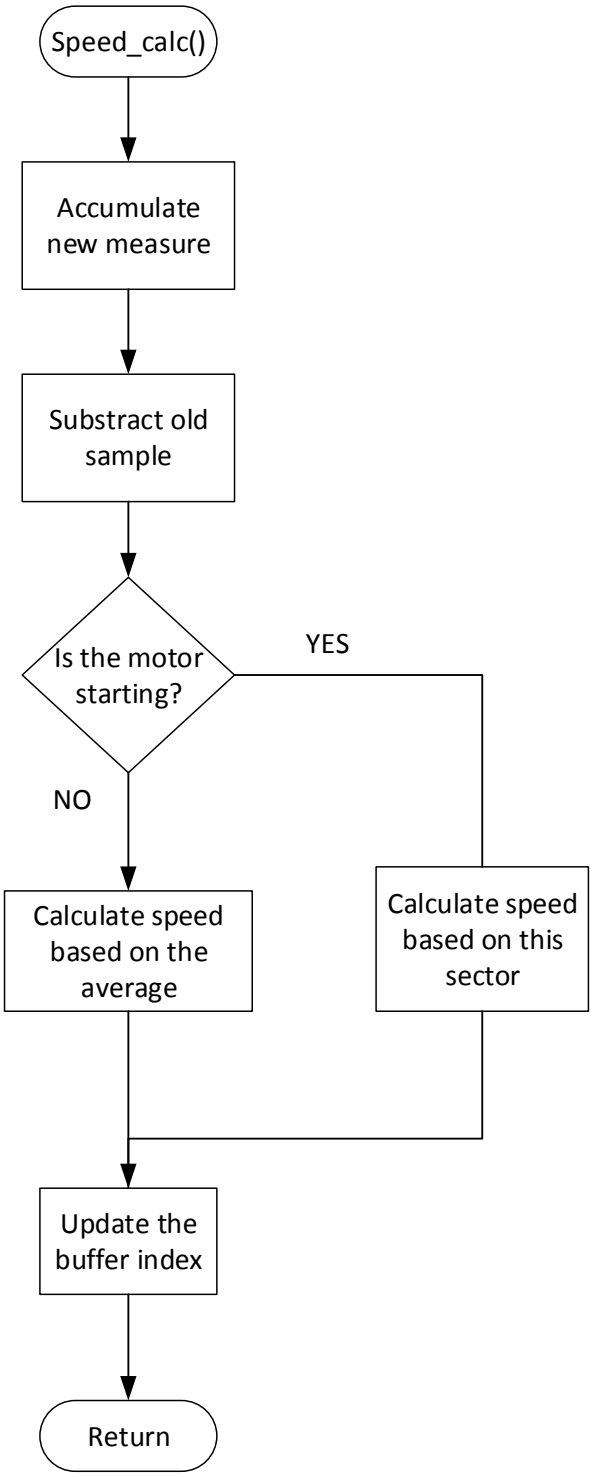


Figure B.14.: Flowchart of Speed_calc

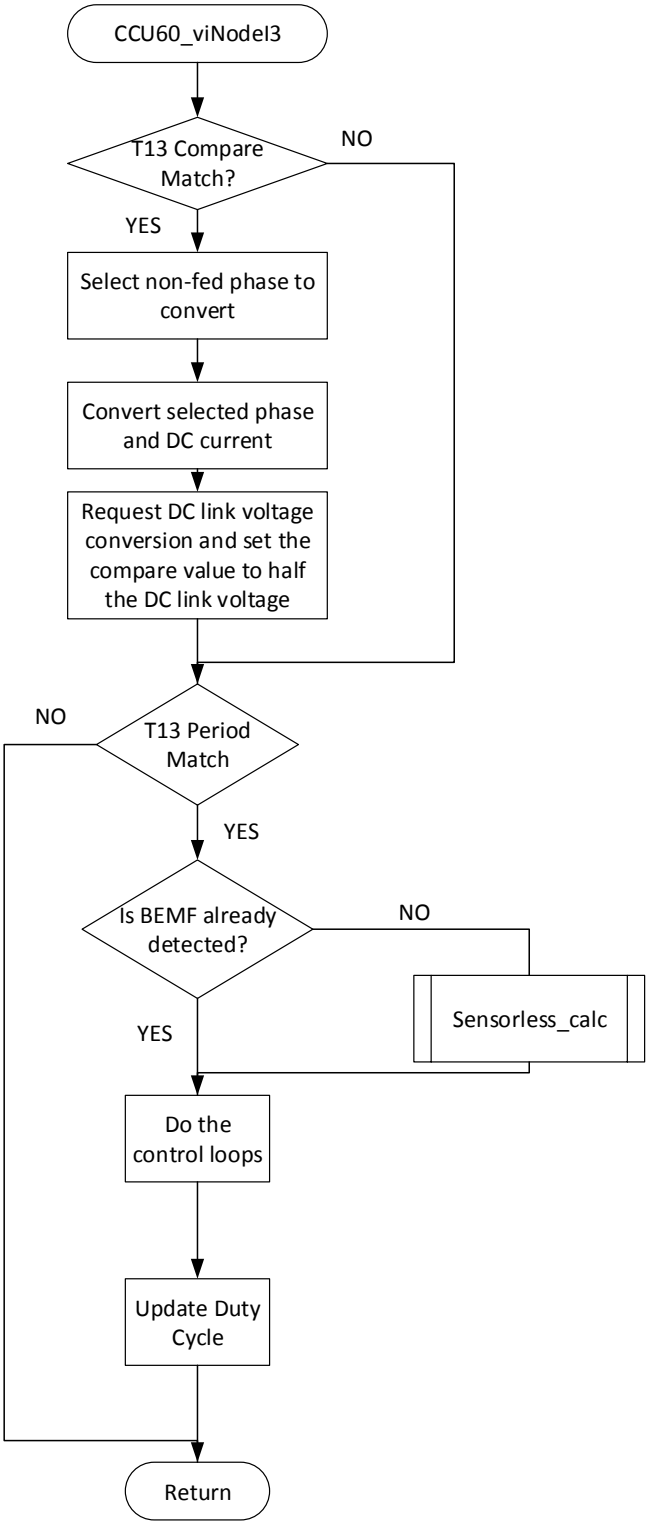


Figure B.15.: Flowchart of CCU60_viNodeI3

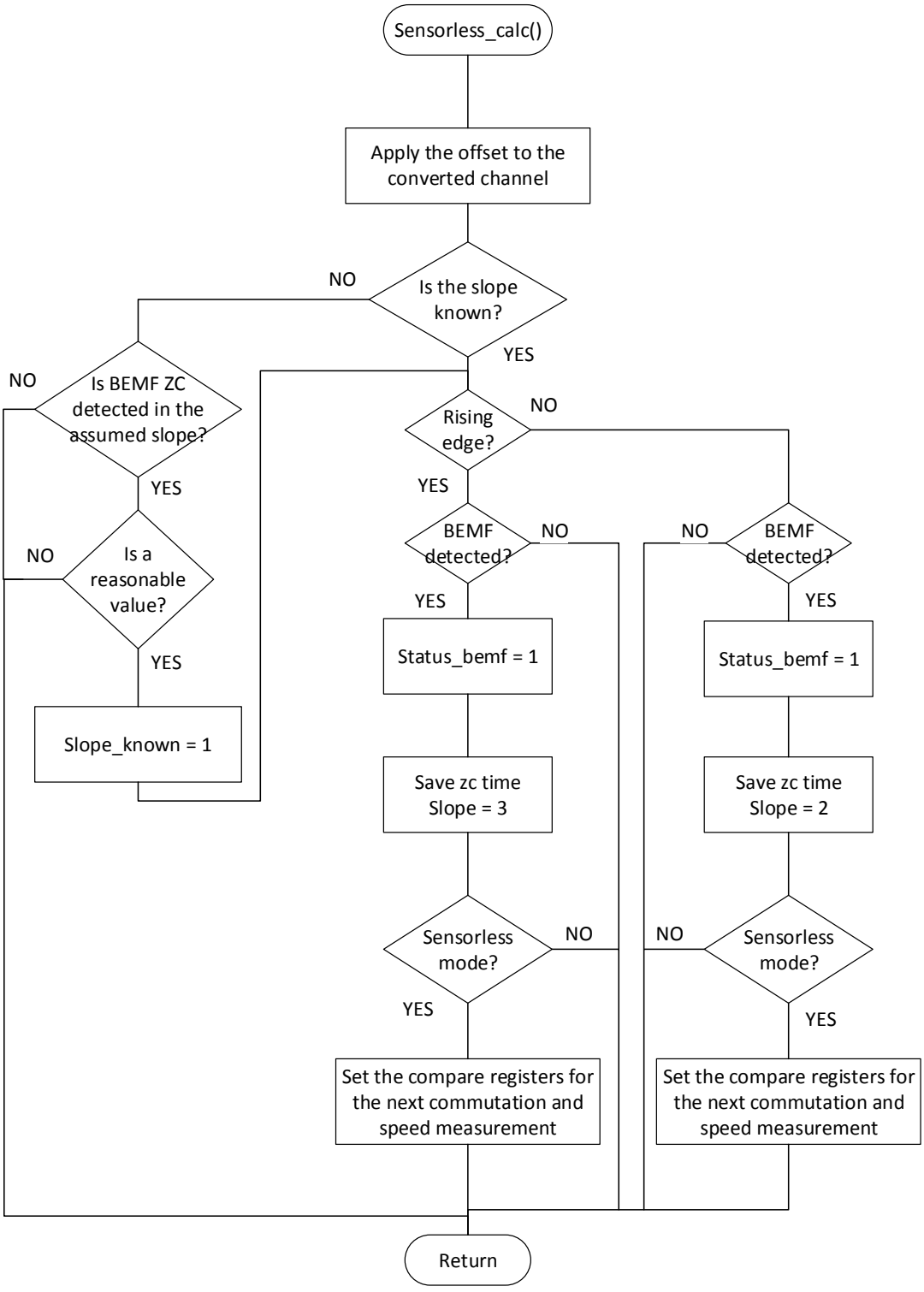


Figure B.16.: Flowchart of Sensorless_calc

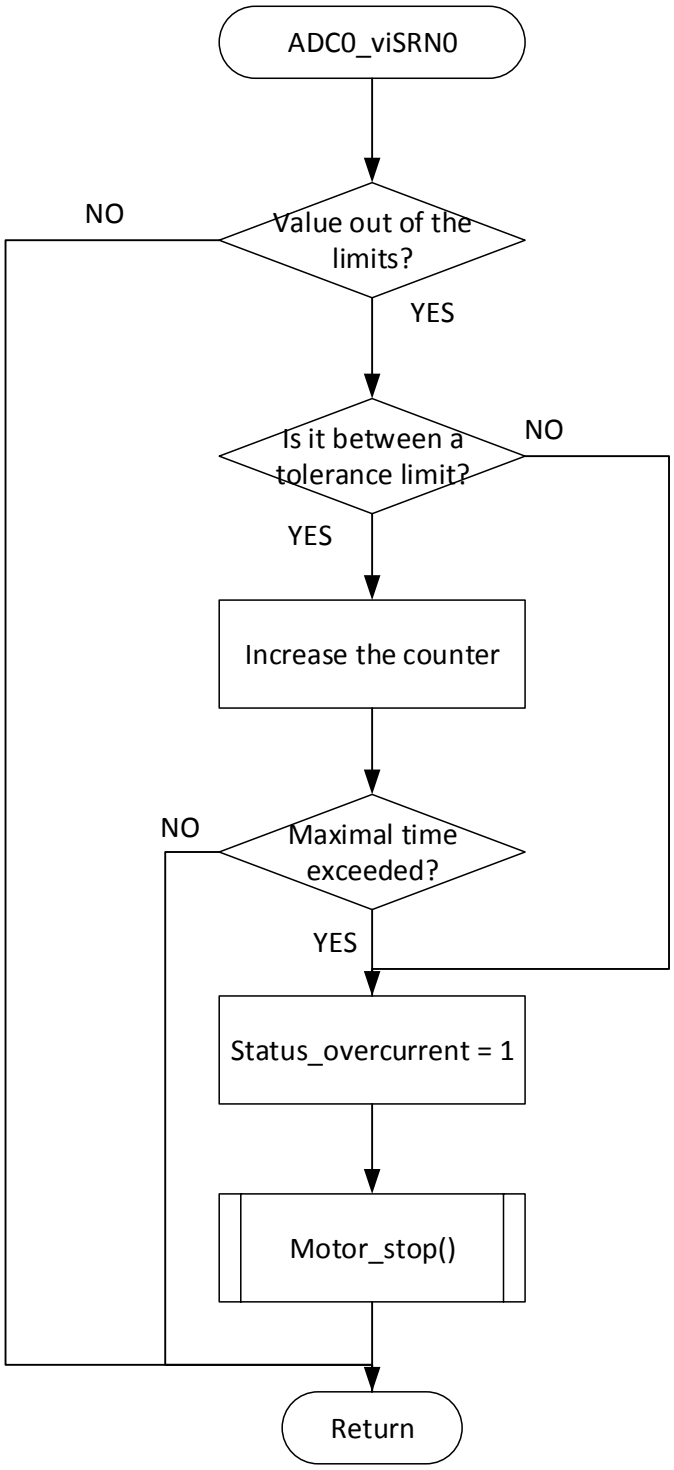


Figure B.17.: Flowchart of ADC0_viSRN0