AN2290
Application note

Flux control simulink and software library of a PMSM

Introduction

This application note describes a software library for the electric motor control implementing a (FOC) Flux Oriented Control on an ST10 microcontroller.

Contents

List of tables

List of figures

1 Introduction

This document describes a software library for the electric motor control implementing a (FOC) Flux Oriented Control on ST10 Microcontroller.

The library consists of:

- Simulink Library;
- Software Library.

The FOC Simulink Library is a set of Simulink blocks for implementing in Matlab-Simulink environment the functions and the algorithms used in the electric motor control. These blocks can be used either to conceive and to test new electric motor controls and to produce automatic generated code in ANSI C, downloadable on microcontroller.

The Software Library is a set of routines for the electric motor control obtained from the code generated in automatic, starting from FOC Simulink library, and then optimized in Assembler. The Software Library is equivalent to FOC Simulink Library from point of view of bit accuracy, same API.

This document begins with an introduction on Flux Oriented Control, the permanent magnet synchronous machine (PMSM) and a short description of its mathematical model. Then, it describes the Flux Oriented Control implementation on Simulink and details the space vector modulation (SVM) technique and the used algorithm.

After the technical introduction, the Simulink Flux Control Library is described, followed by the Software Flux Control Library.

The last part describes the code generation process from the Simulink blocks of this library.

2 FOC-flux oriented control

The Flux Oriented Control (FOC) is a vectorial control strategy that consists of controlling the stator currents represented by a space vector, phase angle and magnitude, by which the terminology “vector control”.

This vector control form is based on three major points:

- the machine current and voltage vectors;
- the transformation of a three phase speed and time dependent system into two co-ordinate time invariant system;
- the effective Pulse Width Modulation pattern generation.

These lead the control of AC machine to acquire every advantage of DC machine control, very similar at the control of separately excited DC machine where the useful torque is proportional to the product of the **field current**, i_f (flux-producing current), and the **armature current**, i_a(torque-producing current) and torque and flux are controlled independently.

$$T_{el} = c \cdot i_f \cdot i_a = c_1 \cdot \psi_f \cdot i_a \qquad [2.1]$$

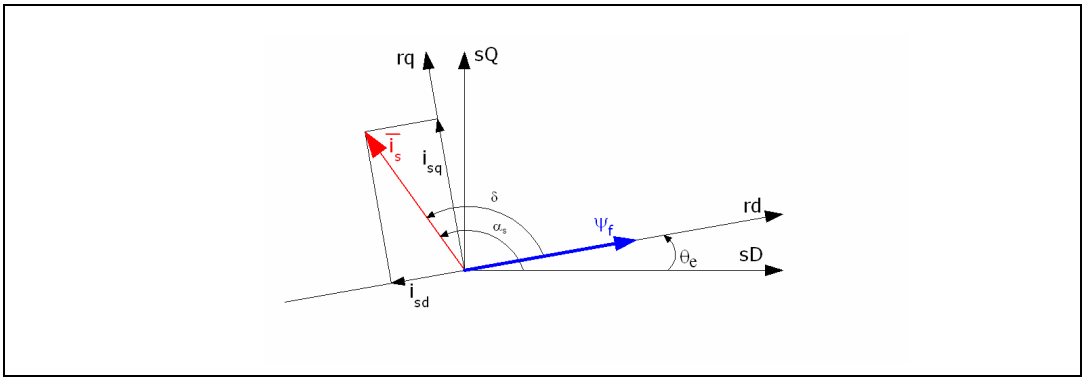
In fact using the **vector theory**, it is easy to show that an expression of the electric torque similar to one of DC machine can be expressed for AC motor, all depends on choice of an appropriate frame where to project the machine model.

The core of FOC control is the projection of a three phases (a,b,c) time and speed dependent system into two co-ordinate (*d,q)* time invariant system, how explained in the following.

Choosing a (*d*,*q)* frame where the *d* axis has the same direction of rotor magnet flux ψ_f, it is possible to verify that the produced electromagnetic torque is proportional to the magnet flux and the quadrature-axis stator current component (torque-producing stator current i_{sq}).

In the *Figure 1*, the Vector Diagram is shown:

Figure 1. Vector diagram



From this figure it is possible to understand that the direct-axis stator current component i_{sd} is the only component in able to modify the field flux, weakening it.

Supposing constant field flux (setting to zero i_{sdref}, if field weakening is no used), a quick torque response is obtained changing only the quadrature-axis stator current component, i_{sq}, by means of a current-controlled PWM inverter, as shows the following equation (2.2):

$$T_{el} = \frac{3}{2} \cdot p\psi_f \left|\bar{i}_s\right| \sin(\alpha_s - \theta_e) = \frac{3}{2} \cdot p\psi_f \left|\bar{i}_s\right| \sin(\delta) = \frac{3}{2} \cdot p\psi_f i_{sq} \quad [2.2]$$

We can conclude saying the Flux Oriented Control, handling instantaneous electrical quantities, is a very accurate control in every working operation, either in steady state and in transient, achieving high dynamic performance in terms of response times and power conversion.

In this application note, the FOC control is explained using a particular AC machine, called Permanent Magnetic Synchronous Machine (PMSM).

A short description of this machine, its features and mathematical model, follow before to explain in details the FOC implementation.

2.1 PMSM

The necessity of reducing the charge of the combustion engine and of eliminating the weight due to the mechanical connections in several applications, like in automotive field, induces to use more and more electric motors, that assure a wide range in speed and torque control satisfying the load demand.

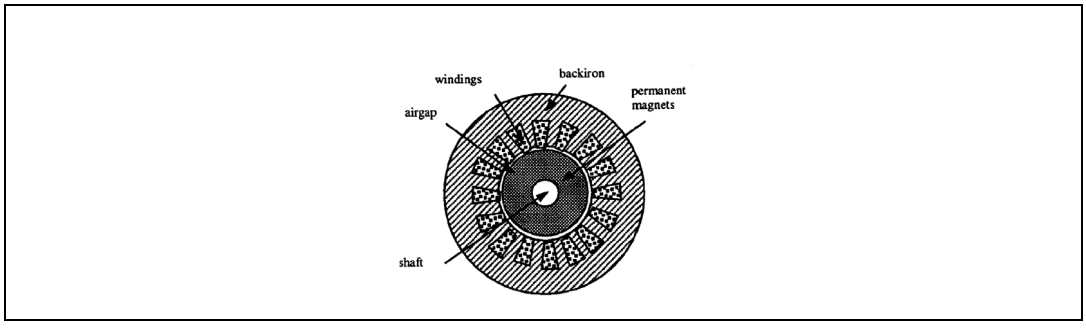
The DC machine fulfils these requirements but needs periodic maintenance.

The AC machine, like induction motor and brushless permanent magnet motor, hasn't brushes, and its rotor is more robust because there aren't commutator and/or rings. That means a very low maintenance, other than increases the power-to-weight ratio and efficiency.

In particular, in the automotive field the Permanent Magnet Synchronous Machine (PMSM) seems to be the best solution.

The brushless permanent magnet motors (PMSMs) have the same electromagnetic structure of a synchronous machine, without the brushes. As shown in the cross-section in the *Figure 2*, they have a wound stator, similar to an induction machine, and a permanent magnet rotor that replaces a rotor fed with dc current, like a synchronous machine. Besides, they need of an internal or external device for sensing of the rotor position, like Hall sensors, encoder or resolver.

Figure 2. Cross-section of PMSM



In fact the PMSMs are not self-commuting motors and to produce useful torque, the currents and the voltages applied to stator phases must be controlled as a function of rotor position.

Therefore it is generally required to count the rotor position with a sensor so that the inverter phases which feed it, acting at any time, are commuted depending on the rotor position.

That explains the necessity of a closed-loop speed/position feedback.

There are two kinds of brushless permanent magnet machines classifiable in account of the shape of the BEMF (back-electromagnetic force):

- DC brushless machine having trapezoidal flux distribution and a trapezoidal BEMF fed by quasi-square wave currents;
- AC brushless machine having approximately sinusoidal air-gap flux density and a quasi-sinusoidal BEMF fed by sinusoidal stator currents.

Generally the DC brushless machines have a simpler control strategy than AC brushless machines.

For trapezoidal flux distributions, to impose quasi-square wave currents on stator windings, it is only needed a six position sensor, with a resolution of at least 60° electrical degrees.

On the contrary, for the sinusoidal current type, the angular position needs to be known with a very accurate precision in order to control each of the three phases currents.

For both kinds, the high reliability of control makes this type of machine a powerful system for electric vehicle application.

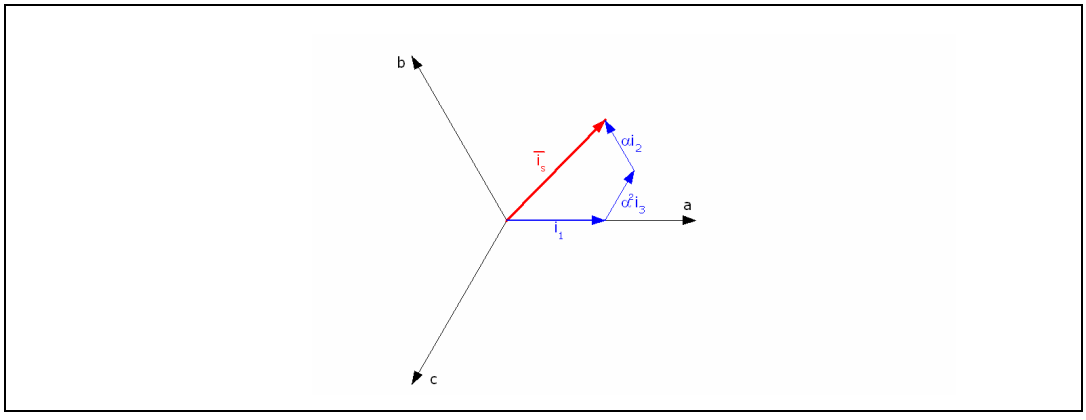
2.2 Mathematical model of the machine

In order to model the fields produced by the stator windings in terms of windings current, “current space vectors” are used. The current space vector for a given winding has the direction of the field produced by that winding and a magnitude proportional to the current through the winding. This allows us to represent the total stator field as a current space vector that is the vector sum of three space vector components, one for each of the stator windings.

The three-phase voltage, currents and fluxes of AC motors can be analyzed in terms of complex space vectors.

For instance, with regard to the currents in the stator windings, the current space vector can be define as follows.

Figure 3. Stator current space vector and its components in (a,b,c)



Assuming that i_{s1},i_{s2},i_{s3} are the instantaneous currents in the stator phases, then the complex stator current vector i_s is defined by:

$$\bar{i_s} = \frac{2}{3} \cdot (i_{s1} + i_{s2} \cdot \alpha + i_{s3} \cdot \alpha^2) \quad [2.3]$$

where $\alpha = e^{j(2\pi/3)}$ and $\alpha^2 = e^{j(4\pi/3)}$ represent spatial operators. The *Figure 3* shows the stator current complex space vector.

That being stated, it is possible to write the mathematical model of an AC brushless machine in a stator frame in terms of space vectors, as follows:

$$\bar{u_s} = R_s \cdot \bar{i_s} + L_s \cdot \frac{d\bar{i_s}}{dt} + \frac{d}{dt}(\psi_f \cdot e^{jp\theta_r}) \quad [2.4]$$

where:

ψ_f	Modulus of the magnetizing flux-linkage vector
R_s	Stator resistance
L_s	Total three phase stator inductance
ω_r	Rotor angular speed
p	Number of pole pairs
θ_r	Mechanical position

and the space vectors:

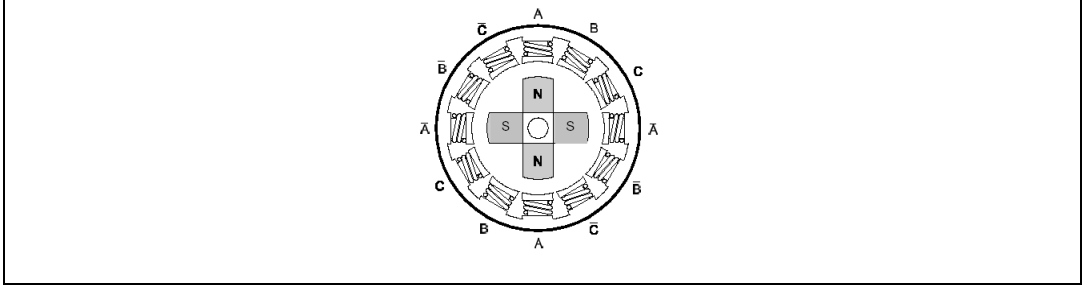
$\bar{u_s} = \frac{2}{3} \cdot \left(u_1(t) + u_2(t) \cdot e^{j\frac{2\pi}{3}} + u_3(t) \cdot e^{j\frac{4\pi}{3}} \right)$ Space vector of the stator voltage

$\bar{i_s} = \frac{2}{3} \cdot \left(i_{s1}(t) + i_{s2}(t) \cdot e^{j\frac{2\pi}{3}} + i_{s3}(t) \cdot e^{j\frac{4\pi}{3}} \right)$ Space vector of the stator current

Note that the mechanical position of the electric motors is related to the rotation of the shaft while the electrical position is relate to the rotation of the rotor magnetic field.

So being the motor with ***p*** pole pairs, its rotor needs only to move *360/p* mechanical degrees to obtain an identical magnetic configuration as when it started.

Figure 4. Phase motor with 2 pole pair



Consequently the electric position of the rotor is linked to the mechanical position by the relation:

$$\theta_e = \theta_r \cdot p$$

To complete the mathematical model of the motor, we include the equation of mechanical equilibrium:

$$J\frac{d\omega_r}{dt} = T_e - (T_m + T_v) \quad [2.5]$$

and substituting the expression of the electric torque (2.2), it yields:

$$\frac{d\omega_r}{dt} = \frac{1}{J} \cdot (T_e - (T_m + T_v)) = \frac{1}{J} \cdot \left(\frac{3}{2} \cdot p\psi_f \left|\bar{i}_s\right| \sin((\alpha_s - p \cdot \theta_r) - (T_m + T_v))\right) \quad [2.6]$$

where:

T_e	Electromagnetic torque
T_m	Mechanical torque
T_v	Viscose friction torque
α_s	Phase of the current space vector respect
J	Inertia momentum of the machine

Note: in Matlab-Simulink environment, the PMSM discrete model has been implemented using a model of the machine in a stationary stator reference *(D,Q)* frame.

2.3 FOC control structure of PMSM

In Flux Oriented Control, motor currents and voltages are manipulated in the *d-q* reference frame of the rotor. This means that measured motor currents must be mathematically transformed from the three-phase stationary reference frame *(a,b,c)* of the stator windings to the two axis rotating *d-q* reference frame, prior to processing, for example by PI controllers (it is possible to use a different controller). Similarly, the voltages to be applied to the motor are mathematically transformed from *d-q* frame of the rotor to the three phases reference frame of stator before they can be used to produce the voltage control signals for the output inverter that feeds the motor.

These transformations are the core of Flux Oriented Control.

Simplifying the expression of the electrical model of the machine, the projection from the three-phase stationary reference frame of the stator windings to the two axis rotating reference frame can be executed into two subsequent steps:

- **(a,b,c) => (*D,Q*)** (the Clarke transformation) which outputs a two co-ordinate time variant system;
- **(*D,Q*) => (*d,q*)** (the Park transformation) which outputs a two co-ordinate time invariant system;

where it is necessary to know in any time the current values in the stator phases and the rotor position to execute the projections from a frame to other one, how explained in details in the following.

Figure 5 is showing the block diagram of the FOC control library, where two motor phase currents,(i.e. i_{s1}, i_{s2}), are measured with two current sensors (e.g. by phase shunts or current transducers) calculating the current in the third winding like the negative sum of the other two windings, ($i_{s3} = -(i_{s1} + i_{s2})$), and then sending them to the Clarke transformation module, (Forward Clarke).

Outputs of this block are the two current components (i_{sD}, i_{sQ}) in the *D,Q* stator fixed frame.

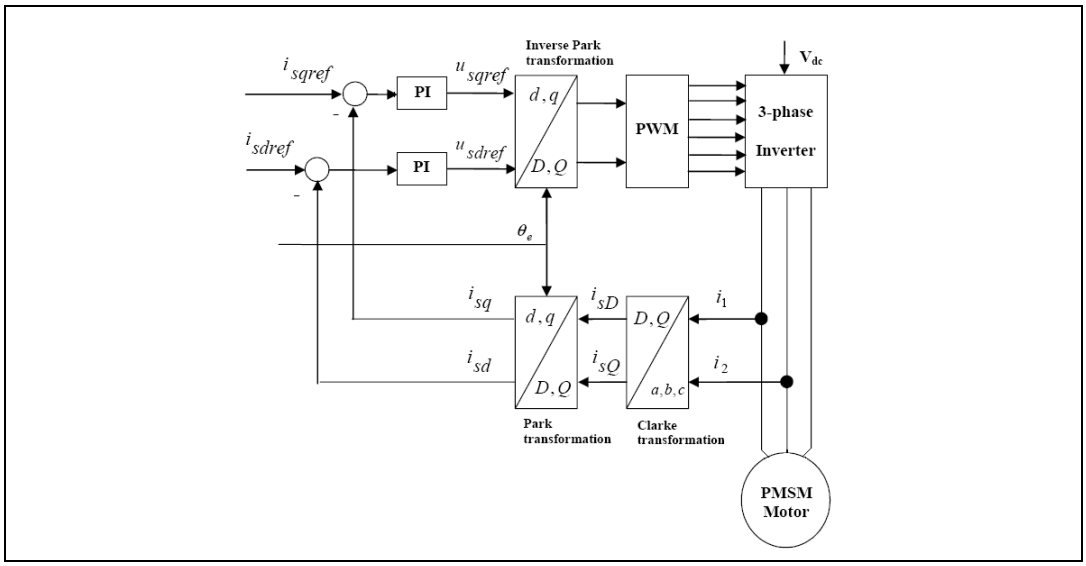
These components are used as inputs of the Park transformation module, (Forward Park), that gives in output the current components (i_{sd}, i_{sq}) in the *d,q* rotating reference frame.

The i_{sd} and i_{sq} measured current components are compared to the references i_{sdref} (the flux reference) and i_{sqref} (the torque reference) and corrected by mean of two PI controllers.

As in brushless synchronous permanent magnet motor the magnet flux is fixed (depending on magnets), in the PMSM control, i_{sdref} should be set to zero, being the only current component in able to weak the flux, while the torque command i_{sqref} could be the output of the speed regulator, e.g. for a speed-FOC. That forces the current space vector $\bar{i}_s$ to be exclusively in the quadrature direction, respect on the magnet flux vector. Since only i_{sq} produces useful torque, this maximizes the torque efficiency of the system.

Then the outputs of two PI, u_{sd} and u_{sq}, are sent to the Inverse Park transformation module, (Reverse Park), from which we get the new components of the stator voltage vector in the (u_{sD}, u_{sQ}) non-rotating stator frame.

Figure 5. Block diagram of the flux oriented control library



These signals are then appropriately processed to produce voltage signals for the output bridge.

In our case, it is chosen to use the **Space Vector Modulation (SVM)** technique to impress the new voltage vector to the motor.

2.4 The space vector modulation theory

The SVM technique is a sophisticated continuous modulation method used, independently from the type of implemented control on the motor, to generate a desired voltage space vector at the output of the inverter that feeds the AC motor, in our case a PMSM. It uses a special scheme to switch the power transistors generating pseudo-sinusoidal currents in the stator windings.

This strategy offers the following advantages to the application:

- Higher performance to control mid/high dynamical motors;
- higher efficiency (86%);
- improved torque management;
- better start up performance;
- constant torque, less torque ripple;
- improved dynamical reaction.

To better understand the space vector modulation algorithm, it is before explained an other fundamental component of the control system: the 3 phase inverter.

2.4.1 The 3-phase inverter

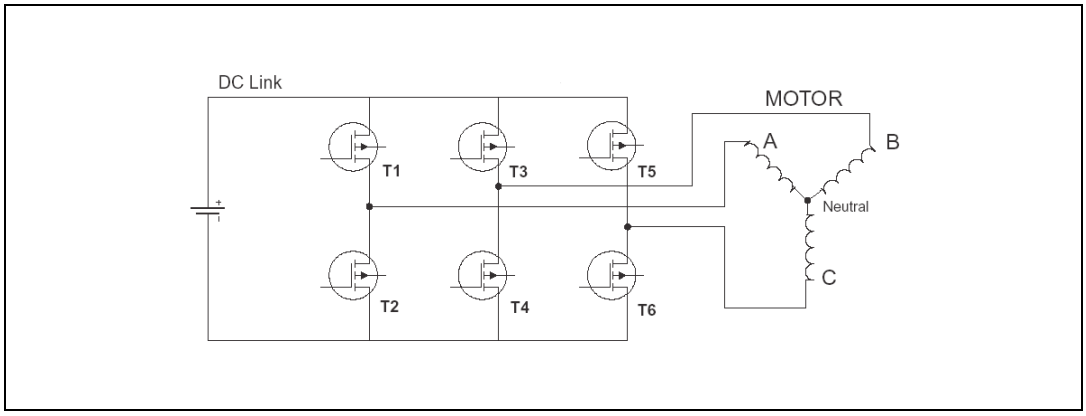
The inverter is a d.c. to a.c. converter. The *Figure 6* shows the structure of a typical 3-phase power inverter connected to the star motor windings, where V_{dc} is the DC Link voltage.

The six switches can be power BJT, GTO, IGBT etc. The state-of-the-art solution for the inverter power stages uses MOSFETs in low-voltage applications (i.e. automotive field).

The ON-OFF sequence of all these devices must respect the following conditions, so as to feed in any time all three stator windings:

– three of the switches must always be ON and three always OFF.
– to avoid shortcut, the upper and lower switches of the same leg are driven with two complementary pulsed signals

Figure 6. 3-phase power inverter scheme



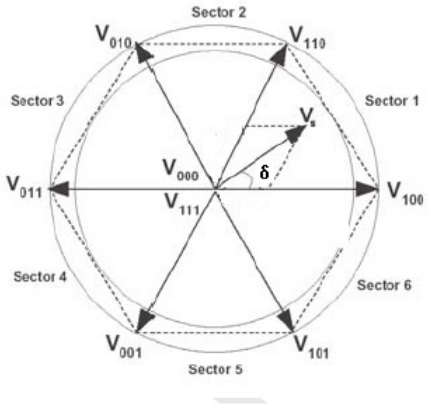
On base of the aforementioned conditions, the inverter has only eight permissible switching states of which six states apply a no-zero voltage to the motor windings and two states with (V_0 and V_7) zero volts when the motor is shorted through the upper or lower transistors.

It is useful to express the eight states of the inverter as space vectors: $V_{0\text{-}7}$ expresses the three voltages V_{An}, V_{Bn}, V_{Cn}, that are spatially separated 120° apart, as a space vector for each of the switching states 0-7.

The six vectors including the zero voltage vectors can be expressed geometrically on the complex plane as shown in the following *Figure 7*.

In order to generate a rotating field into the machine to produce a useful torque, the inverter has to be switched in all the possible eight states. A way to use the inverter is the operating mode called **“six-step mode”**, that generates high magnitude low order harmonics which cannot be filtered by motor inductance.

Figure 7. Space vector diagram



2.4.2 The Space vector pulse width modulation

The inverter is able to apply only eight space vector positions to the stator winding of an electric machine, while the control imposes a voltage space vector that vary in all the inner cycle of the hexagon in order to create a smooth rotating field (see *Figure 7*).

The SVM block, here implemented in Simulink, allows to generate the appropriate PWM patter to impulse the inverter so that any voltage vector inside the space vector hexagon can be produced by “time weighting“.

It is based on the fact that a reference voltage vector $\overline{V}_s$, can be realized by a combination of the two adjacent active vectors and the zero vectors inside of sector where it lies. The output space vector voltage (choosing an appropriate PWM period, T_{PWM}, so as to suppose steady the vector $\overline{V}_s$ in this period), can be computed by the integral:

$$\int_0^{T_{PWM}} (\overline{V_s} \cdot dt) = \left(\int_0^{\delta} \overline{V_{0/7}} + \int_0^{\alpha} \overline{V_k} + \int_0^{\beta} \overline{V_{k+1}} \right) \quad [2.7]$$

from which it yields:

$$\overline{V_s} = \overline{V_{0/7}} \cdot \frac{\delta}{T_{PWM}} + \overline{V_k} \cdot \frac{\alpha}{T_{PWM}} + \overline{V_{k+1}} \cdot \frac{\beta}{T_{PWM}}$$

where:

$$T_{PWM} = \alpha + \beta + \delta \quad [2.8]$$

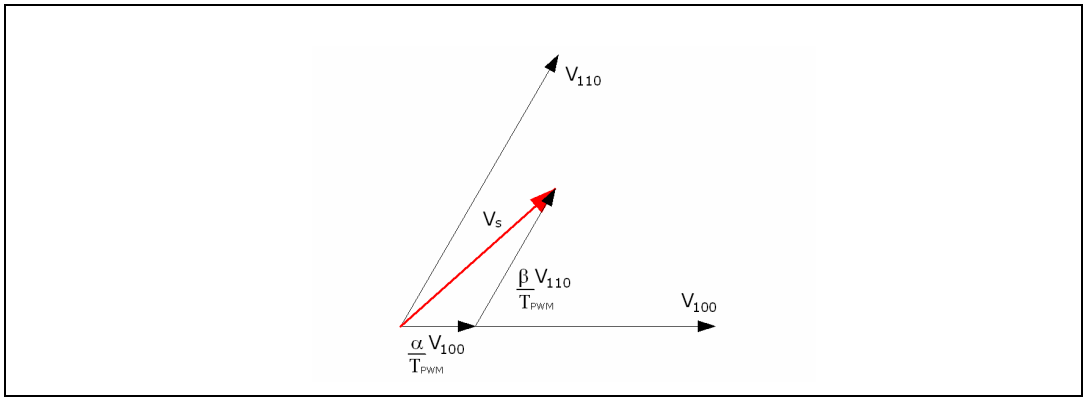
V_k and V_{k+1} (k=0,..7) are the vectors that bound the sector in which the reference vector is included, T_{PWM} is the switching period and α, β and δ are the time frames of V_k, V_{k+1} and V_0/V_7 vectors in that sector.

The resulting equation is:

$$|\overline{V_S}| = \frac{|\overline{V_k} \cdot \alpha + \overline{V_{k+1}} \cdot \beta|}{T_{PWM}} \quad [2.9]$$

For example, assuming that the vector $\overline{V_s}$ is in the 1st sector, we have the following situation:

Figure 8. SVM in the 1st sector



in which α and β are the times during which the vectors V_{100} and V_{110} are applied.

2.4.3 Sector finder

To impose the $\overline{V}_s$ voltage vector at the motor windings, it is fundamental the knowledge of the sector where the reference vector is included for a correct behavior of SVM.

In the classic approach, in order to find the sector, you need to know the phase of the complex reference vector that is calculated with the known formula:

$$\gamma = \arctan\left(\frac{\Im m(\overline{V}_s)}{\Re e(\overline{V}_s)}\right) \quad [2.10]$$

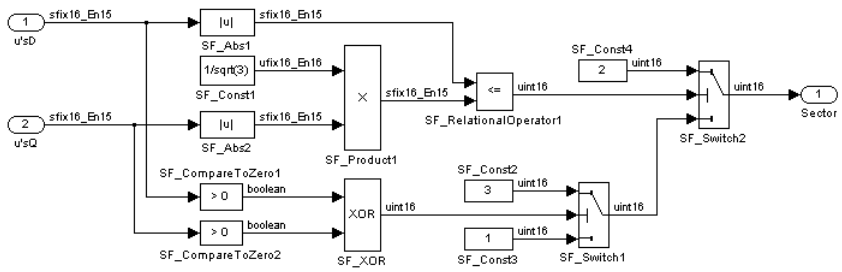
The main problem of this formula is the calculation of *arctan* implemented on a 16bit microcontroller. One solution can be to calculate the *arctan* making a look-up table of the *arctan* function.

Another solution (here chosen) is to recognize the sector of reference vector $\overline{V_s}$, starting from the knowledge of the sign of the imaginary (quadrature component) and real (direct component) parts of the reference vector (u_{sDref}, u_{sQref}) written in a stator non-rotating frame and the comparison of the their magnitudes in order to avoid the division in the equation [2.10].If the vector phase obtained from the formula [2.11] is bigger than $\pi/3$ the vector lies in sector 2 or 5. The control on sign of the components discriminates between the sector 1-4 and 6-3 if the angle is less than $\pi/3$.

$$\left|\frac{\Im m(\overline{V}_s)}{\Re e(V_s)}\right| \geq \arctan\left(\frac{\pi}{3}\right) \Rightarrow |\Im m(V_s)| \geq \arctan\left(\frac{\pi}{3}\right) \cdot |\Re e(\overline{V}_s)| \quad [2.11]$$

i*Figure 9* shows the exploded Sector Finder block:

Figure 9. Sector finder schematic



2.4.4 SVM formulation

To create reference vector $\overline{V}_s$ inside one of the six sectors, the reference vectors which bound that sector, have to be “time weighted”, like shown in the equation [2.9]. It is possible to modulate the reference voltage vector and to apply the better switching pattern in term of power dissipation on the power switches of the inverter and to make that, it is necessary to choose the strategy to apply the vector V_k and V_{k+1} (active vectors) in each sector.

We have some freedom degrees to choose the modulation algorithm as:

- The choice of the zero vector- whether $V_0(000)$ or $V_7(111)$ or both;
- Sequencing of the vector;
- Splitting of the duty cycle of the vectors without introducing additional commutations.

Literature demonstrates that, in order to reduce the number of commutation and switching losses, it is preferable to utilize a states sequence where the states are adjacent.

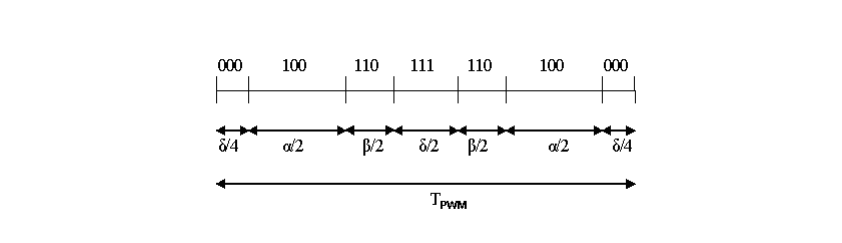
This means that passing from a state to the successive one should occur with only one switch commutation.

According to that, the scheme chosen is a symmetric sequence in which there are seven conduction states, so called “Seven states Space Vector Modulation”.

Dividing the conduction time of every component of inverter in opportune time frames, as shown below for a vector into Sector 1

:

Figure 10. Example of a switching pattern in sector 1



every component switches two times for every PWM period.

Analyzing the classical approach of SVM, it starts from the following equation:

$$\overline{V_s} \cdot T_{PWM} = \overline{V_k} \cdot \alpha + \overline{V}_{k+1} \cdot \beta + \overline{V}_0 \cdot \delta \quad [2.12]$$

$$\alpha + \beta + \delta = T_{PWM} \quad [2.13]$$

in which we are supposing steady in every period of PWM the vectors V_k, V_{k+1}, V_0 and $\overline{V}_s$.

Projecting the vector equation [2.12] on the real and imaginary axis (*D,Q*) in the stator non-rotating frame, it yields:

$$\alpha \cdot (V_k)_D + \beta \cdot (V_{k+1})_D = u_{sD} \cdot T_{PWM} \quad [2.14]$$

$$\alpha \cdot (V_k)_Q + \beta \cdot (V_{k+1})_Q = u_{sQ} \cdot T_{PWM} \quad [2.15]$$

So solving the above said equations system for every sector, the following table will be obtained:

Table 1. Time frames of application of V_k,V_{k+1} and V_0

	Sector 1	Sector 2	Sector 3
$\frac{\alpha}{T_{PWM}}$	$\frac{u_{sD}}{V} - \left(\frac{\sqrt{3}}{3} \cdot \frac{u_{sQ}}{V}\right)$	$\frac{u_{sD}}{V} + \left(\frac{\sqrt{3}}{3} \cdot \frac{u_{sQ}}{V}\right)$	$\frac{u_{sD}}{V} \cdot \frac{2}{3} \cdot \sqrt{3}$
$\frac{\beta}{T_{PWM}}$	$\frac{u_{sQ}}{V} \cdot \frac{2}{3} \cdot \sqrt{3}$	$\left(-\frac{u_{sD}}{V}\right) + \left(\frac{\sqrt{3}}{3} \cdot \frac{u_{sQ}}{V}\right)$	$-\frac{u_{sD}}{V} - \left(\frac{\sqrt{3}}{3} \cdot \frac{u_{sQ}}{V}\right)$
δ	$T_{PWM} - \alpha - \beta$	$T_{PWM} - \alpha - \beta$	$T_{PWM} - \alpha - \beta$

	Sector 4	Sector 5	Sector 6
$\frac{\alpha}{T_{PWM}}$	$\left(-\frac{u_{sD}}{V}\right) + \left(\frac{\sqrt{3}}{3} \cdot \frac{u_{sQ}}{V}\right)$	$-\frac{u_{sD}}{V} - \left(\frac{\sqrt{3}}{3} \cdot \frac{u_{sQ}}{V}\right)$	$-\frac{u_{sD}}{V} \cdot \frac{2}{3} \cdot \sqrt{3}$
$\frac{\beta}{T_{PWM}}$	$-\frac{u_{sQ}}{V} \cdot \frac{2}{3} \cdot \sqrt{3}$	$\left(\frac{u_{sD}}{V}\right) - \left(\frac{\sqrt{3}}{3} \cdot \frac{u_{sQ}}{V}\right)$	$\left(\frac{u_{sD}}{V}\right) + \left(\frac{\sqrt{3}}{3} \cdot \frac{u_{sQ}}{V}\right)$
δ	$T_{PWM} - \alpha - \beta$	$T_{PWM} - \alpha - \beta$	$T_{PWM} - \alpha - \beta$

where V is $\frac{2}{3} \cdot V_{dc}$.

In this SVM algorithm, there isn't the need of calculation of trigonometric functions to obtain the time frames (α,β,δ), as it happens in a classical approach, and for every PWM period only multiplications and divisions are needed.

The time frames so calculated from this algorithm must be processed with a look up table in order to establish on which phase must be applied.

In the following table there are the time calculations to be done:

	Sector 1	Sector 2	Sector 3	Sector 4	Sector 5	Sector 6
t1	$\frac{\delta}{4}$	$\frac{\beta}{2}+\frac{\delta}{4}$	$\frac{T_{PWM}}{2}-\frac{\delta}{4}$	$\frac{T_{PWM}}{2}-\frac{\delta}{4}$	$\frac{\alpha}{2}+\frac{\delta}{4}$	$\frac{\delta}{4}$
t2	$\frac{\alpha}{2}+\frac{\delta}{4}$	$\frac{\delta}{4}$	$\frac{\delta}{4}$	$\frac{\beta}{2}+\frac{\delta}{4}$	$\frac{T_{PWM}}{2}-\frac{\delta}{4}$	$\frac{T_{PWM}}{2}-\frac{\delta}{4}$
t3	$\frac{T_{PWM}}{2}-\frac{\delta}{4}$	$\frac{T_{PWM}}{2}-\frac{\delta}{4}$	$\frac{\alpha}{2}+\frac{\delta}{4}$	$\frac{\delta}{4}$	$\frac{\delta}{4}$	$\frac{\beta}{2}+\frac{\delta}{4}$

3 Flux control simulink library

3.1 Description

The implementation of the FOC control needs of some peculiar functions. The Simulink library implements all needed functions to built a FOC based electric motor control application using the following blocks, here listed:

- Forward Clarke;
- Forward Park;
- Reverse Park;
- Sin_cos;
- PI;
- SVM.

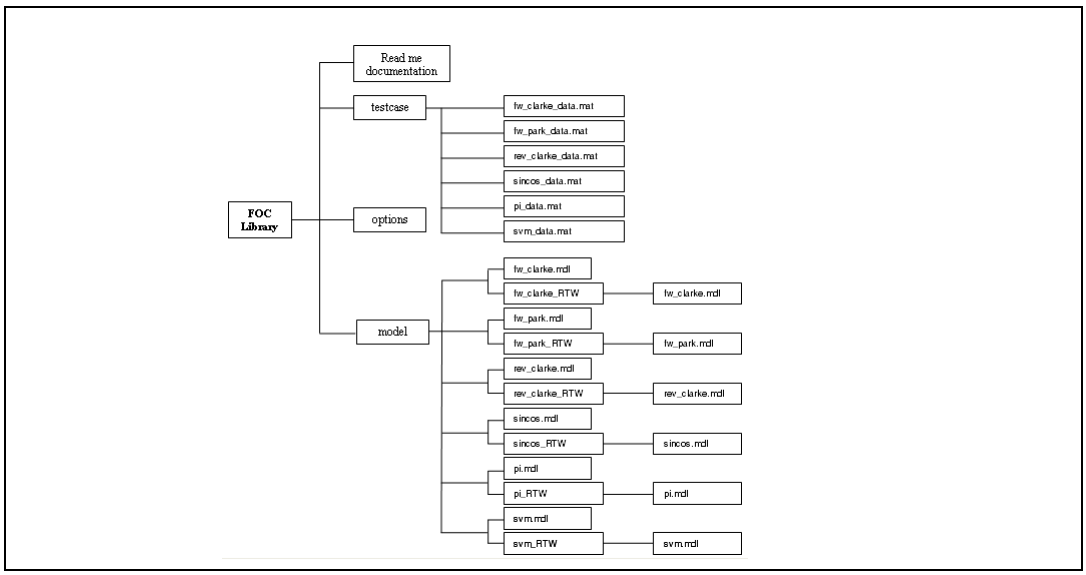
3.2 Using the simulink library

The 2 main directories of the library package are:

- 1 directory for all test cases: 1 subdirectory per library function;
- 1 directory for all .mdl files.

The file structure is the following:

Figure 11. Simulink library structure



3.2.1 How to install simulink library

The Simulink Library is delivered as an archive file with .zip extension. To install one you need to unzip the file in the (C:) directory for a correct use.

Note: you must have a 7.0.0 Matlab version or upward installed on your system to use this library, plus a licence for Fixed-Point-Precision Toolbox to use the “convert block“in each scheme block and a licence of RTW Embedded Coder Toolbox.

Please, read the README.txt file in the archive file for using the library.

3.2.2 Test environment

.mat: the inputs and outputs data obtained by Simulink in the double format are stored. When the mdl file is opened data is loaded in Workspace of Matlab.

The name of each test-file begins with the (yyy) function name that it refers, followed by underscore and the suffix “data”.

3.3 Parameters format

The FOC control system in Simulink has the same behavior of one implemented on micro where it was necessary to use a different fixed point precision number representation in every block. In the *Table 2* the variables and their representations are listed:

Table 2. Data representation

variable	representation	description
i_{s1}	sfix(16,8)	phase current
i_{s2}	sfix(16,8)	phase current
i_{s3}	sfix(16,8)	phase current
i_{sD}	sfix(16,8)	direct-axis current component in stator fixed frame
i_{sQ}	sfix(16,8)	quadrature-axis current component in stator fixed frame
theta_el	ufix(16,16)	electrical angle
cos_t	sfix(16,14)	$\cos(\theta_e)$
sin_t	sfix(16,14)	$\sin(\theta_e)$
i_{sd}	sfix(16,8)	direct-axis current component in rotor no-fixed frame
i_{sq}	sfix(16,8)	quadrature-axis current component in rotor no-fixed frame
u_{sd}	sfix(16,6)	direct-axis voltage component in rotor no-fixed frame
u_{sq}	sfix(16,6)	quadrature-axis voltage component in rotor no-fixed frame
u_{sQ}	sfix(16,6)	quadrature-axis voltage component in stator fixed frame
t_1	int16	time frames
t_2	int16	time frames
t_3	int16	time frames

In the following, the Simulink implemented blocks are described in details.

3.4 Clarke transformation

Description

The Clarke Transformation projects the motor currents (i_{s1},i_{s2},i_{s3}) from the 120° degrees physical frame to a two co-ordinate stator non-rotating frame (i_D,i_Q).

Arguments

i_{s1} phase current;
i_{s2} phase current;
i_{s3} phase current;
i_{sD} direct-axis current component in stator fixed frame;
i_{sQ} quadrature-axis current component in stator fixed frame.

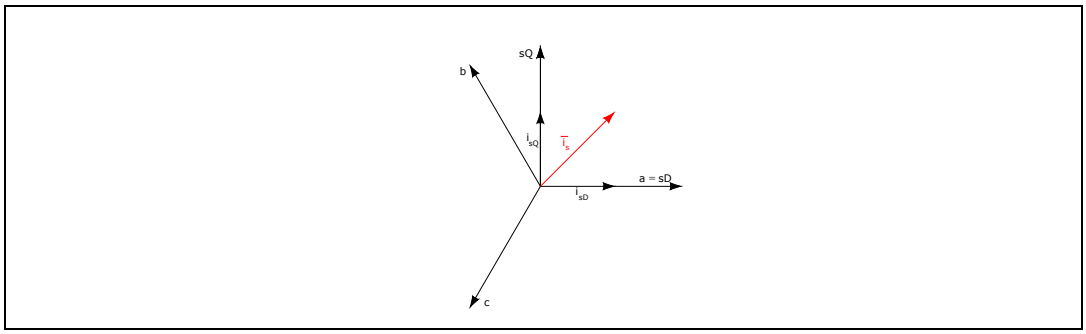
Algorithm

The following equations are implemented:

$$\begin{bmatrix} i_{sD} \\ i_{sQ} \end{bmatrix} = \begin{bmatrix} i_{s1} \\ \frac{1}{\sqrt{3}} \cdot (i_{s2} - i_{s3}) \end{bmatrix} \quad [3.1]$$

where assuming that the axis *a* (axis of the first phase) and the axis *sD* (stands for stator direct axis) are in the same direction, we have the following vector diagram:

Figure 12. Stator current space vector

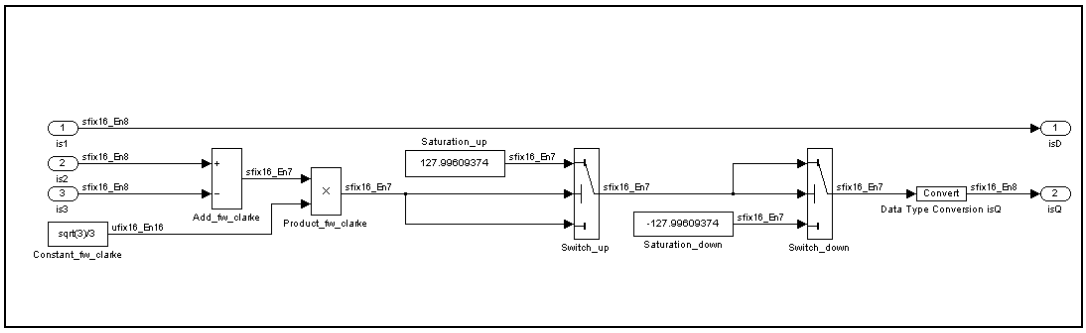


We have so obtained a two co-ordinate system that still depends on time and speed.

Simulink block

As shown here below in the *Figure 13*, the Forward Clark block, implemented in Simulink, receives in input the three current signals, here represented in sfix(16,8) fixed-point format, returning in output the two current components, (i_{sD},i_{sQ}), in the stator fixed frame, in the same fixed-point format.

Figure 13. Forward Clarke block



Test case

In **fw_clarke_data.mat** file the inputs and outputs data to test this function are stored.

3.5 Park transformation

Description

The currents (i_{sD},i_{sQ}) in the stator fixed frame are projected in the (*d*,*q)* rotor rotating frame where the flux vector direction is chosen as the *direct-axis d.*

Arguments

i_{sD} direct-axis current component in stator fixed frame;

i_{sQ} quadrature-axis current component in stator fixed frame;

$\sin(\theta_e)$;

$\cos(\theta_e)$;

i_{sd} direct-axis current component in rotor no-fixed frame;

i_{sq} quadrature-axis current component in rotor no-fixed frame.

Algorithm

The following equations are implemented:

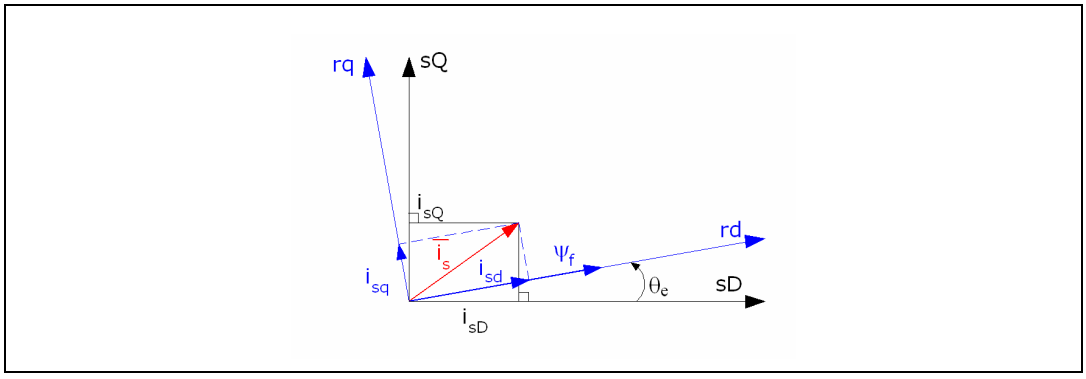
$$\begin{bmatrix} i_{sd} \\ i_{sq} \end{bmatrix} = \begin{bmatrix} \cos(p\theta_e) & \sin(p\theta_e) \\ -\sin(p\theta_e) & \cos(p\theta_e) \end{bmatrix} \cdot \begin{bmatrix} i_{sD} \\ i_{sQ} \end{bmatrix} \quad [3.2]$$

where ($p\theta_r = \theta_e$) represents the electric position of the rotor flux. Substituting in the equation [3.2] the expressions of (i_{sD},i_{sQ}), it yields:

$$\begin{bmatrix} i_{sd} \\ i_{sq} \end{bmatrix} = \begin{bmatrix} \cos(p\theta_e) & \sin(p\theta_r) \\ -\sin(p\theta_e) & \cos(p\theta_e) \end{bmatrix} \cdot \begin{bmatrix} i_{s1} \\ \frac{1}{\sqrt{3}} \cdot (i_{s2} - i_{s3}) \end{bmatrix} \quad [3.3]$$

here represented in the *Figure 14*:

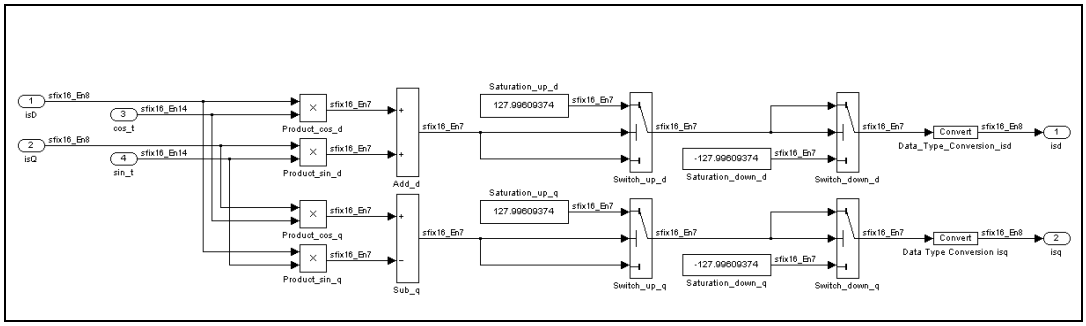
Figure 14. Stator space vector into rotor frame



Simulink block

As shown in the following, the Forward Park block, implemented in Simulink, receives in input the two current components (i_{sD},i_{sQ}), here represented in sfix(16,8) fixed-point format, returning in output the two current components (i_{sd},i_{sq}) in the rotor rotating frame, in the same format.

Figure 15. Forward Park block



Test case

In **fw_park_data.mat** file the inputs and outputs data to test this function are stored.

3.6 Inverse Park transformation

Description

With this transformation, the voltage vectors outputs of PI controllers are projected from rotor rotating frame in the stator fixed frame.

Arguments

u_{sd} direct-axis voltage component in rotor no-fixed frame;

u_{sq} quadrature-axis voltage component in rotor no-fixed frame.

$\sin(\theta_e)$;

$\cos(\theta_e)$;

u_{sD} direct-axis voltage component in stator fixed frame;

u_{sQ} quadrature-axis voltage component in stator fixed frame;

Algorithm

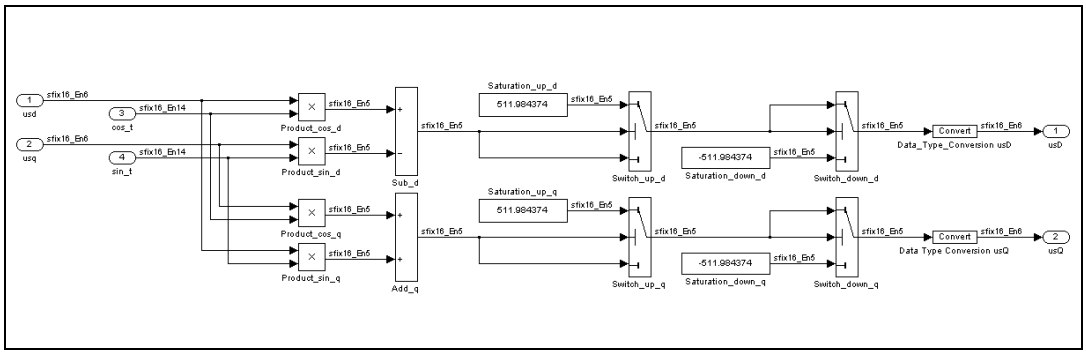
The following equations are implemented:

$$\begin{bmatrix} u_{sD} \\ u_{sQ} \end{bmatrix} = \begin{bmatrix} \cos(p\theta_r) & -\sin(p\theta_r) \\ \sin(p\theta_r) & \cos(p\theta_r) \end{bmatrix} \cdot \begin{bmatrix} u_{sd} \\ u_{sq} \end{bmatrix} \quad [3.4]$$

Simulink block

As shown here below in the *Figure 16*, the Reverse Park block, implemented in Simulink, receives in input the two voltage components (u_{sd},u_{sq}), here represented in sfix(16,6) fixed-point format, returning in output the two current components (u_{sD},u_{sQ}) in the rotor rotating frame, in the same format.

Figure 16. Reverse Park block



Test case

In **rev_clarke_data.mat** file the inputs and outputs data to test this function are stored.

3.7 Sin_cos

Description

Known the electrical position of the rotor, the functions sin(θ_e) and cos(θ_e) are calculated to project the space vectors from a frame to other one.

Arguments

θ_e electrical position;

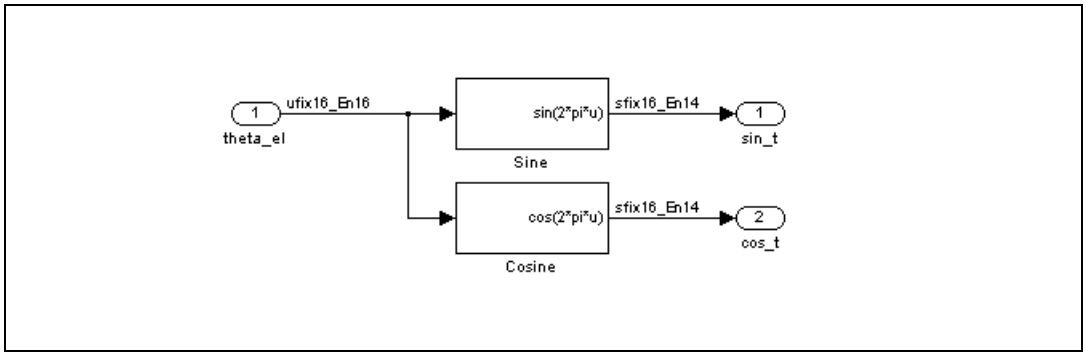
sin(θ_e);

cos(θ_e).

Algorithm

Simulink block

Figure 17. Sin_cos block



Test case

In **sincos_data.mat** file the inputs and outputs data to test this function are stored.

3.8 PI block

Description

An electrical driver based on the FOC control needs of some controllers. In our case two PI controllers: one for the torque component reference i_{sqref}, one for the flux component reference i_{sdref}.

Arguments

i_{sdref} (o i_{sqref})	reference signal
i_{sd} (o i_{sq})	measured signal
u_{sd} (o u_{sq})	command signal

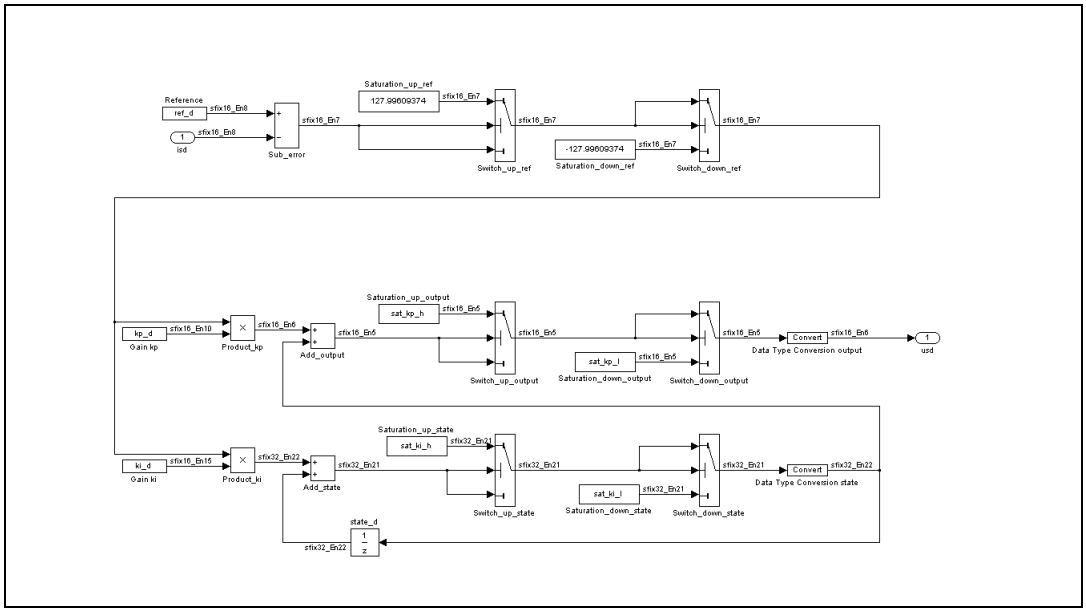
Algorithm

$$U_k = K_p \cdot e_k + K_i \cdot e_k + \sum_{n=0}^{k-1} e_n \qquad [3.6]$$

Simulink block

The structure of the PI controller, in the discrete format, used in the Simulink model is shown in *Figure 18*:

Figure 18. PI structure



Test case

In **pi_d_q_data.mat** file the inputs and outputs data to test this function are stored.

The configuration parameters of PI are in **pi_d_q_conf.mat** file.

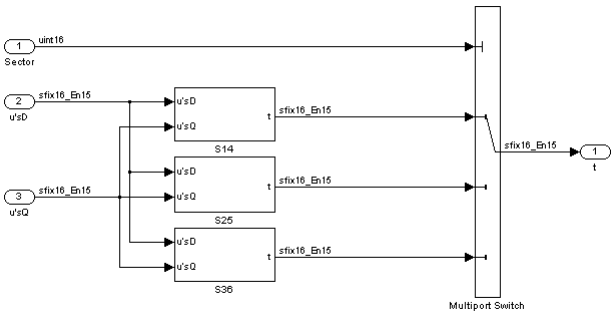
3.9 SVM

Description

The goal of Space Vector Modulation is to generate three appropriate PWM signals to pulse the inverter, that feeds the motor, so that three voltage vectors shifted (by 120° between each other) can be produced on the phases of the motor.

Given a voltage space vector of module $\overline{V}_s$ and angle γ, the implemented algorithm modulates this vector in output applying on the inverter a switching pattern in order to reduce the power dissipation on the electronic switches.

Figure 19. SVM scheme



It was possible to develop a Simulink block based on an optimized SVM algorithm, that receives the outputs of the two PI controllers and the voltage of the DC link of the Inverter, to produce the control signals.

How it is possible to view in the *Figure 19*, to implement the time frames of *Table 1*,we use only three blocks for six sectors so to generate the appropriate switching pattern, choosing on base of the actual sector where the vector $\overline{V}_s$ lies in. This simplification, from six to three blocks, it is possible observing that the six sectors are symmetric.

In this way we can calculate the time frames (t_1 ,t_2,t_3) for each pair of sectors (1-4, 2-5 and 3-6) and with a Selector, from Sector Finder block, choose the right time frames.

Moreover the algorithm implements the dc-ripple-compensation by recalculating the voltage components (u_{sD} ,u_{sQ}) into relative voltages compared to measured dc-bus voltage:

$$U_s' = \frac{U_s}{U_{DC}}$$

where:

U_s is absolute voltage

U_s' is relative voltage

U_{DC} is dc-link voltage

Arguments

u_{sD}	direct-axis voltage component in stator fixed frame;
u_{sQ}	quadrature-axis voltage component in stator fixed frame;
V_{DC}	battery (or DC link) Voltage;
R_{PWM}	PWM resolution;
t_1, t_2, t_3	time frames.

Algorithm

We can calculate the time frames for each pair of sectors (1-4, 2-5 and 3-6) and with a Selector, from Sector Finder block, choose the right time frames.

For instance, substituting the values of the time frames, α, β, δ, supposing a reference vector in the Sector 1:

$$\alpha = T_{PWM} \cdot \frac{2}{3} \cdot \left(u_{sD} - \frac{\sqrt{3}}{3} \cdot u_{sQ}\right) \cdot \left(\frac{3}{2} \cdot \frac{1}{V_{dc}}\right)$$

$$\beta = T_{PWM} \cdot \frac{2}{3} \cdot \sqrt{3} \cdot u_{sQ} \cdot \left(\frac{3}{2} \cdot \frac{1}{V_{dc}}\right)$$

$$\delta = T_{PWM} - \alpha - \beta$$

substituting in the expressions of t_1,t_2,t_3, it yields:

$$t_1 = \frac{\delta}{4} = T_{PWM} \cdot \left(\frac{1}{4} - u'_{sD} - \left(\frac{\sqrt{3}}{3} \cdot u'_{sQ}\right)\right)$$

$$t_2 = \frac{\alpha}{2} + \frac{\delta}{4} = T_{PWM} \cdot \left(\frac{1}{4} + u'_{sD} - \sqrt{3} \cdot u'_{sQ}\right) \quad [3.7]$$

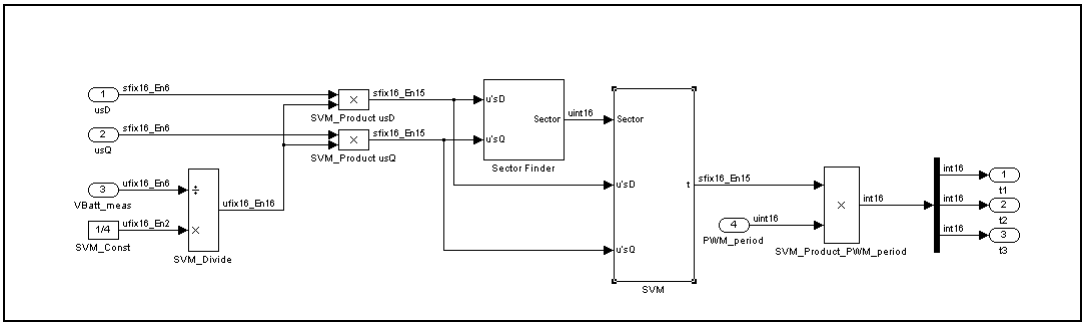
$$t_3 = \frac{T_{PWM}}{2} - \frac{\delta}{4} = T_{PWM} \cdot \left(\frac{1}{4} + u'_{sD} + \frac{\sqrt{3}}{3} \cdot u'_{sQ}\right)$$

In the Sector 4 the result is the same, changing only the sign of the reference voltage vector (u'_{sD} ,u'_{sQ}).

Simulink block

In the following figure, the schema of the SVM is described:

Figure 20. SVM implementation block



Here below it are exploded the blocks calculating in Simulink the time frames for a reference vector inside the sector 1-4, 2-5 and 3-6.

Figure 21. Sector 1-4 implementation

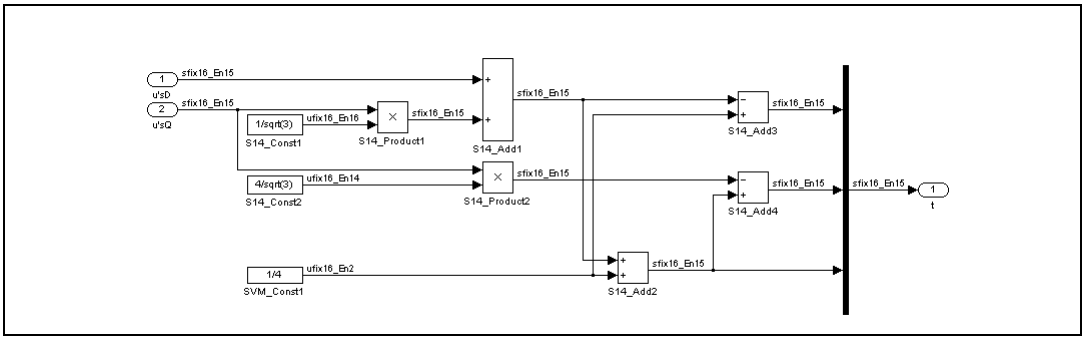


Figure 22. Sector 2-5 implementation

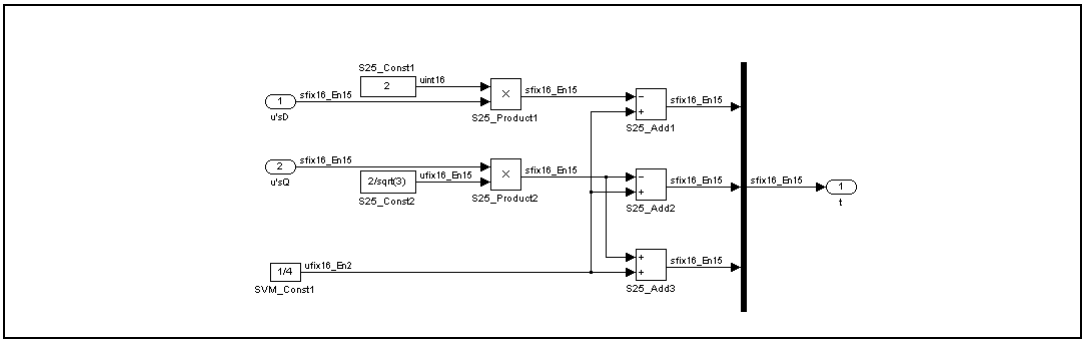
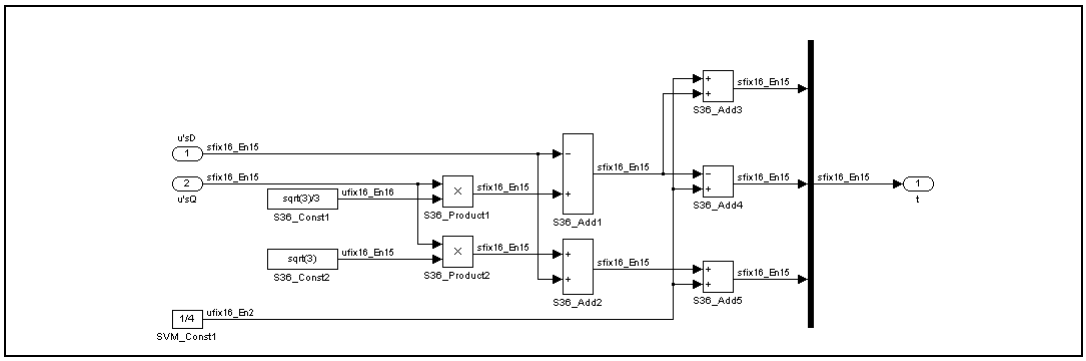


Figure 23. Sector 6-3 implementation



The results of the SVM block are in int16 format.

Test Case

In **svm_data.mat** file the inputs and outputs data to test this function are stored.

4 Flux control software library

4.1 Description

The Flux Control Software library provides the functions for mixed “C” and Assembly programmers on ST10 microcontrollers necessary to implement an (FOC) electric motor control.

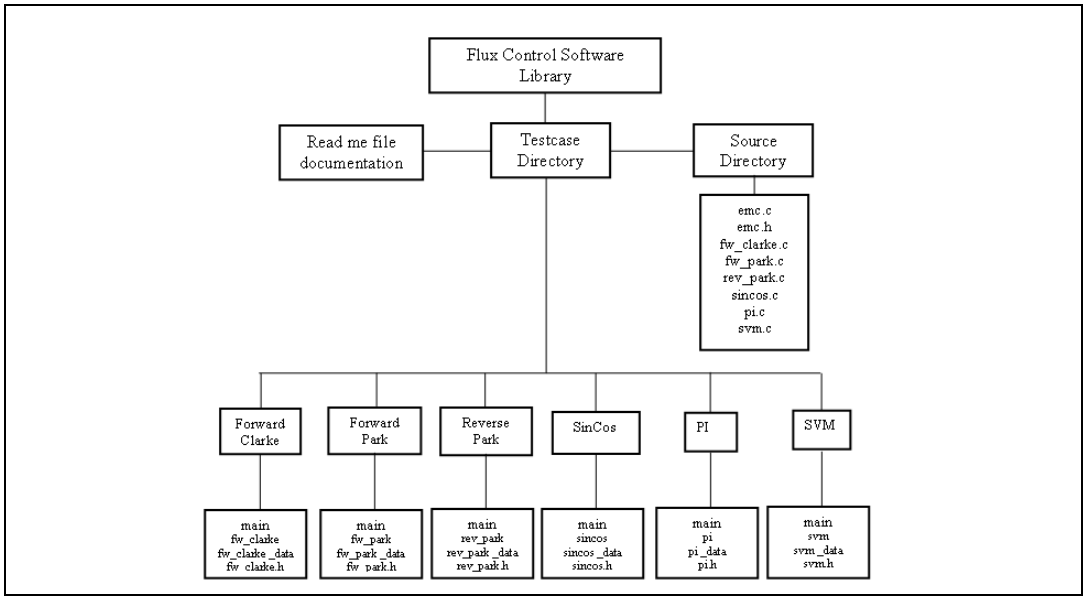
4.2 Using the software library

The 2 main directories of the library package are:

- 1 directory for all test cases: 1 subdirectory per library function.
- 1 directory for all .c sources file: all functions

The file structure is the following:

Figure 24. File structure



4.2.1 How to install Software library

The Software Library is delivered as an archive file with .zip extension. To install the Software Library you need to unzip the file in the directory where you want the library to be copied into.

Note: Please, read the README.txt file in the archive file for specific details on the release.

4.2.2 Tool chain compatibility

FOC library is compatible with Tasking tool chain (V7.5r2 and upward).

4.2.3 Calling a function

The functions have been written to be called by a C language program.

To include a function in a C language program, it is needed to:

- include the **“emc.h”**
- You find this .h file in the Source directory of the library package.

4.2.4 ST10 MAC configuration

This library has been done for implementing electric motor control functions (FOC control), using 16-bit data in fixed point precision with different representations (i.e. sfix(16,8), sfix(16,6), etc.). The implemented functions have been optimized with MAC commands using the default configuration (the user have not to change the configuration registers of MAC).

4.2.5 Real time aspects

Any DSP code developed for ST10 can be interrupted at any time and execution resumed after the interrupt routine. There is no added latency when the DSP library is used.

Interrupt routine requirements: the only requirements are only when the DSP unit is used by other tasks that have different priorities: the interrupting task that may interrupt another task using the DSP should save and restore the MAC registers at the entry point and exit point of the routine. (use **#pragma savemac** in Tasking tool chain).

4.2.6 Naming convention

The name of each functions coincides with the name of the Simulink equivalent block, that implements it on micro.

Example: fw_park

The fw label represents the direction of the projection, from a (D,Q) frame to a (d,q) frame.

rev_park

The rev label represents the direction of the projection, from a (d,q) frame to a (D,Q) frame.

4.2.7 Test environment

yyy_data.c : you find the input data vectors and the output data vectors, obtained by Simulink for the same function block, in int16 format.

The name of each test-file begins with the (yyy) function name that it refers, followed by underscore and the suffix “data”.

4.2.8 Flux control library benchmark

The following table gives the characteristics of the main functions of the library:

Table 3. FOC library capabilities

Function	Code size (bytes)	Nb cycles
Forward Clark	70	21
PID	226	57
Reverse park	538	48
SVM	822	74
SINCOS	532	56

4.3 Library functions

4.3.1 Forward Clarke

FCLARKE_c_step

FCLARKE_c_step(ExternalInputs_fw_clarke *fw_clarke_U, ExternalOutputs_fw_clarke *fw_clarke_Y);

Data types and structures:

ExternalInputs_fw_clarke

This structure contains the motor phases currents.

```
typedef struct _ExternalInputs_fw_clarke_tag {
int16_T is1;                    phase current;
int16_T is2                     phase current;
int16_T is3;                    phase current;
} ExternalInputs_fw_clarke;
```

ExternalOutputs_fw_clarke

This structure contains the current components in a fixed (D,Q) stator frame.

```
typedef struct _ExternalOutputs_fw_clarke_tag {
int16_T isD;                    current component in a fixed (D,Q) stator frame;
int16_T isQ;                    current component in a fixed (D,Q) stator frame;
} ExternalOutputs_fw_clarke;
```

Description:

It projects the motor currents (i_{s1},i_{s2},i_{s3}) from the 120° degrees physical frame to a two co-ordinate stator non-rotanting frame (i_D,i_Q), using 16-bit operands.

Arguments:

fw_clarke_U	pointer to the inputs structure
fw_clarke_Y	pointer to the outputs structure

Algorithm:

$$\text{isD} = \frac{1}{\sqrt{3}} \cdot (\text{is2} - \text{is3})$$

$$\text{isQ} = \left(\frac{1}{\sqrt{3}} \cdot (\text{is2} - \text{is3})\right)$$

Notes:

Test:

To test this function, include the **fw_clarke_data.c** file in the current directory. In the .c file you find the inputs and outputs vectors defined as const.

4.3.2 Forward Park

fw_park

FPARK_c_step(ExternalInputs_fw_park *fw_park_U, ExternalOutputs_fw_park *fw_park_Y);

Description:

It projects the current components (i_{sD},i_{sQ}) from the fixed stator frame in the (*d*,*q*) rotor rotanting frame, using 16-bit operands.

Data types and structures:

ExternalInputs_fw_park

This structure contains the current components and the sin() and cos() functions of the eletrical angle.

```
typedef struct _ExternalInputs_fw_park_tag {
  int16_T isD;        direct-axis current component in (D,Q) stator frame;
  int16_T isQ;        quadrature-axis current component in (D,Q) stator frame;
  int16_T cos_t;      cos(θe)
  int16_T sin_t;      sin(θe)
} ExternalInputs_fw_park;
```

ExternalOutputs_fw_park

This structure contains the current components in a no-fixed rotor frame.

```
typedef struct _ExternalOutputs_fw_park_tag {
  int16_T isd;        current component in a no-fixed rotor frame
  int16_T isq;        current component in a no-fixed rotor frame
} ExternalOutputs_fw_park;
```

Arguments:

fw_park_U	pointer to the inputs structure
fw_park_Y	pointer to the outputs structure

Algorithm:

$$i_{sd} = i_{sD} \times \cos(\theta_e) + i_{sQ} \times \sin(\theta_e)$$

$$i_{sq} = -i_{sD} \times \sin(\theta_e) + i_{sQ} \times \cos(\theta_e)$$

Notes:

Test:

To test this function, include the **fw_park_data.c** file in the current directory. In the .c file you find the inputs and outputs vectors defined as const.

4.3.3 Reverse Park

rev_park

RPARK_c_step(ExternalInputs_rev_park *rev_park_U, ExternalOutputs_rev_park *rev_park_Y);

Description:

It projects the outputs of PI controllers (u_{sd},u_{sq}), from rotor rotating frame in the stator fixed frame (u_{sD},u_{sQ}), using 16-bit operands.

Data types and structures:

ExternalInputs_rev_park

This structure contains the direct-axis and quadrature-axis voltage components in a no-fixed (d,q) rotor frame and the sin() and cos() functions of the electrical angle.

typedef struct _ExternalInputs_rev_park_tag {
int16_T usd; direct-axis voltage component in a no-fixed (d,q) rotor frame
int16_T usq; quadrature-axis voltage component in a no-fixed (d,q) rotor frame
int16_T cos_t; $\cos(\theta_e)$
int16_T sin_t; $\sin(\theta_e)$
} ExternalInputs_rev_park;

ExternalOutputs_rev_park

This structure contains the current components in a *(D,Q)* stator frame.

typedef struct _ExternalOutputs_rev_park_tag {
int16_T usD; direct-axis voltage component in *(D,Q)* stator frame
int16_T usQ; quadrature-axis voltage component in *(D,Q)* stator frame
} ExternalOutputs_rev_park;

Arguments:

rev_park_U pointer to the inputs structure
rev_park_Y pointer to the outputs structure

Algorithm:

$$u_{sD} = u_{sd} \times \cos(\theta_e) - u_{sq} \times \sin(\theta_e)$$

$$u_{sqQ} = u_{sd} \times \sin(\theta_e) + u_{sq} \times \cos(\theta_e)$$

Notes:

Test:

To test this function, include the **rev_park_data.c** file in the current directory. In the .c file you find the inputs and outputs vectors defined as const.

4.3.4 Sin_Cos

sincos

SINCOS_c_step(ExternalInputs_sin_cos *sin_cos_U, ExternalOutputs_sin_cos *sin_cos_Y);

Description:

Data types and structures:

ExternalInputs_sin_cos

This structure contains the current electrical angle.

typedef struct _ExternalInputs_sin_cos_tag {
uint16_T theta_el; θ_e electrical position
} ExternalInputs_sin_cos;

ExternalOutputs_sin_cos

This structure contains the sin() and cos() functions of the electrical angle.

typedef struct _ExternalOutputs_sin_cos_tag {
int16_T sin_t;; sin(θ_e)
int16_T cos_t; cos(θ_e)
} ExternalOutputs_sin_cos;

Arguments:

sin_cos_U	pointer to the inputs structure
sin_cos_Y	pointer to the outputs structure

Algorithm:

Notes:

Test:

To test this function, include the **sincos_data.c** file in the current directory.

In the .c file you find the inputs and outputs vectors defined as const.

4.3.5 PI controller

pi

pi_d_c_step(D_Work_pi_d *pi_d_DWork, ExternalInputs_pi_d *pi_d_U, ExternalOutputs_pi_d *pi_d_Y);

pi_q_c_step(D_Work_pi_q *pi_q_DWork, ExternalInputs_pi_q *pi_q_U, ExternalOutputs_pi_q *pi_q_Y);

Description:

It implements classical PI scheme for each control component (i_{sd},i_{sq}).The error and the proportional and integral terms are forced to be in a range of values so as to calculate the reference voltage signals, (u_{sd},u_{sq}), using 16-bit operands.

Data types and structures:

ExternalInputs_sin_cos

This structure contains the controlled signal (isd) of pi_d.

```
typedef struct _ExternalInputs_pi_d_tag {
int16_T isd;
} ExternalInputs_pi_d;
```

This structure contains the state of pi_d.

```
typedef struct D_Work_pi_d_tag {
int32_T state_d_DSTATE;
} D_Work_pi_d;
```

This structure contains the output signal (usd) of pi_d.

```
typedef struct _ExternalOutputs_pi_d_tag {
int16_T usd;
} ExternalOutputs_pi_d;
```

This structure contains the controlled signal (isq) of pi_q.

```
typedef struct _ExternalInputs_pi_q_tag {
int16_T isq;
} ExternalInputs_pi_q;
```

This structure contains the state of pi_q.

```
typedef struct D_Work_pi_q_tag {
int32_T state_q_DSTATE;
} D_Work_pi_q;
```

This structure contains the output signal (usq) of pi_q.

```
typedef struct _ExternalOutputs_pi_q_tag {
int16_T usq;
} ExternalOutputs_pi_q;
```

Arguments:

pi_d_DWork	pointer to the state structure
pi_d_U	pointer to the inputs structure
pi_d_U	pointer to the outputs structure

Algorithm:

Notes:

Test:

To test this function, include the **pi_data.c** file in the current directory. In the .c file you find the inputs and outputs vectors defined as const.

4.3.6 SVM

svm

SVM_c_step(ExternalInputs_svm *svm_U, ExternalOutputs_svm *svm_Y);

Description:

It calculates three signals (t_1,t_2,t_3) to impose the correct switching pattern on the Inverter, using 16-bit operands.

Data types and structures:

ExternalInputs_svm

This structure contains (u_{sD} , u_{sQ}) the voltage components in *(D,Q)* stator frame, V_{dc} DC link voltage,T_{PWM} period of timer of PWM unit.

```
typedef struct _ExternalInputs_svm_tag {
int16_T usD;          direct-axis voltage component in (D,Q) stator frame
int16_T usQ;          quadrature-axis voltage component in (D,Q) stator frame
uint16_T VBatt_meas;  measured DC link voltage
uint16_T PWM_period;  period of timer of PWM unit
} ExternalInputs_svm;
```

ExternalOutputs_svm

This structure contains the duty- cycles.

```
typedef struct _ExternalOutputs_svm_tag {
int16_T t1;           duty cycle applied on first phase
int16_T t2;           duty cycle applied on second phase
int16_T t3            duty cycle applied on third phase
} ExternalOutputs_svm;
```

Arguments:

svm_U	pointer to the inputs structure
svm_Y	pointer to the outputs structure

Algorithm:

The algorithm implements a different set of equations according to the actual position of voltage vector, represented by its components (u_{sD},u_{sQ}).

For example if the reference vector is in the Sector1:

$$t_1 = T_{PWM} \cdot \left(\frac{1}{4} - u'_{sD} - \left(\frac{\sqrt{3}}{3} \cdot u'_{sQ}\right)\right)$$

$$t_2 = T_{PWM} \cdot \left(\frac{1}{4} + u'_{sD} - \sqrt{3} \cdot u'_{sQ}\right)$$

$$t_3 = T_{PWM} \cdot \left(\frac{1}{4} + u'_{sD} + \frac{\sqrt{3}}{3} \cdot u'_{sQ} \right)$$

Notes:

Test:

To test this function, include the **svm_data.c** file in the current directory. In the .c file you find the inputs and outputs vectors defined as const.

5 C code auto generation

5.1 Overview

When the Simulink schematics are done, converted to fixed point precision and tested, the last step is to generate C code downloadable on the microcontroller. This step is done using two toolboxes of Matlab:

- The Real Time Workshop
- The Real Time Workshop Embedded Coder

The Real Time Workshop is an essential tool used in rapid prototyping with Simulink. Automatic program building allows you to make design changes directly to the block diagram, putting algorithm development (including coding, compiling, linking, and downloading to target hardware) under control of a single process.

In this part, a set of signal processing functions for C programmers on ST10 are presented.

5.2 Steps to generate optimized C code

- Design a model in Simulink

The rapid prototyping process begins with the development of a model in Simulink. Using principles of control engineering, it's possible to model plant dynamics and other dynamic components that constitute a controller and/or an observer.

- Simulate the Model in Simulink

Using MATLAB-Simulink, and toolboxes it's possible to develop algorithms and analyze the results.If the results are not satisfactory, it's possible to iterate the modelling and analysis process until results are acceptable.

- Generate Source Code with Real-Time Workshop

Once simulation results are acceptable, it's possible to generate downloadable C code that implements the appropriate portions of the model. Simulink could be used in external mode to monitor signals, tune parameters, and further validate and refine the model, quickly iterating through solutions.

- Implement a Production Prototype

At this stage, the rapid prototyping process is complete.

5.3 Real-Time Workshop

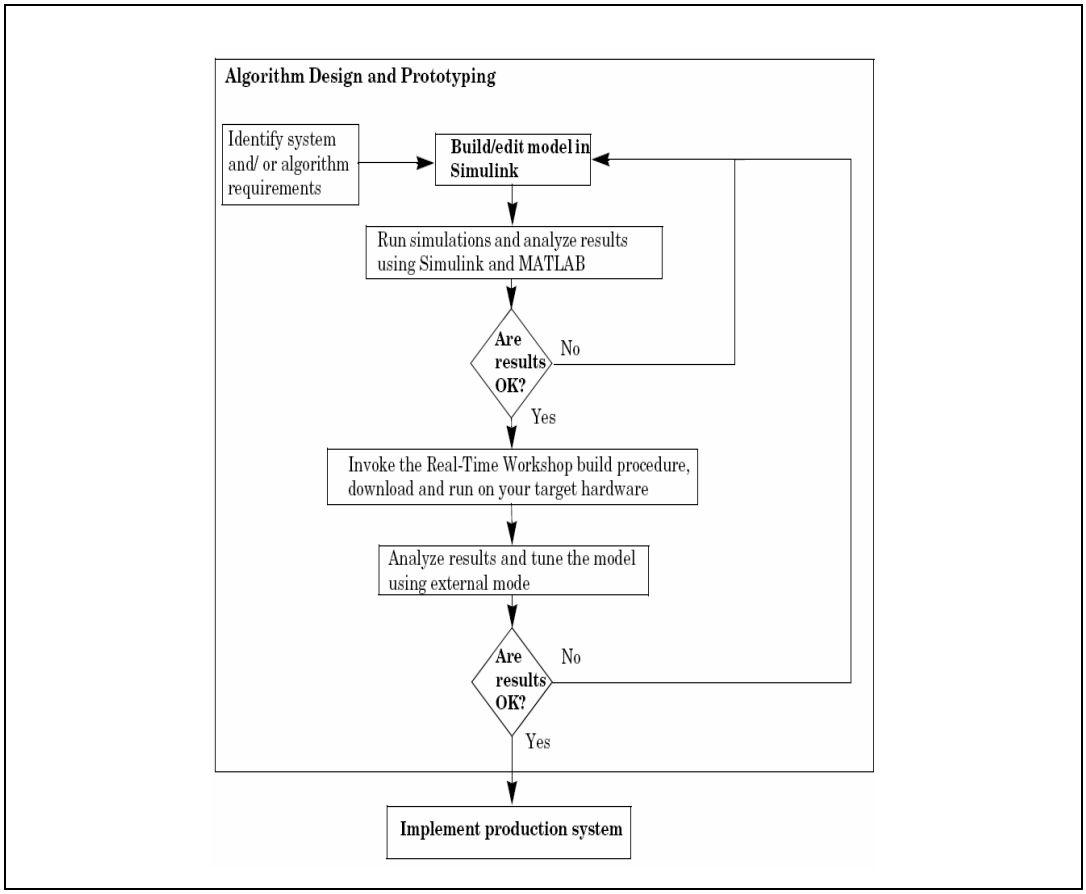
The Real-Time Workshop Embedded Coder is a separate, add-on product for use with Real-Time Workshop.

It is intended for use in embedded systems development to generate code that is easy to read, trace, and customize for all production environment. The Real-Time Workshop Embedded Coder provides a framework for the development of production code that is optimized for speed, memory usage, and simplicity. It generates optimized ANSI-C or ISO-C code for fixed point and floating point microprocessors. It extends the capabilities provided by the Real-Time Workshop to support specification, integration, deployment, and testing of production applications on embedded targets. The Real Time Workshop Embedded Coder

addresses targeting considerations such as RAM, ROM, and CPU constraints, code configuration, and code verification.

The *Embedded Real-Time* (***ERT***) target, provided by the Real Time Workshop Embedded Coder, is designed for customization.

Figure 25. Flow chart



In our applications we use the ERT target with optimization for fixed point systems. Correct specification of target-specific characteristics of generated code (such as word sizes for char, int, and long data types, or desiderated rounding behaviors in integer operations) can be critical in embedded systems development. The Hardware Implementation category of options in the settings menu provides a simple and flexible way to control such characteristics in both simulation and code generation.

5.4 How to generate C code using Real Time Workshop

Starting from a model in fixed point precision it is described step by step how to generate C code.

For the example, the inner loop of FOC control will be considered and starting from the Simulink schematic the C code will be automatically generate.

Step 1 - Simulink schematic constructor

The first step is the construction of the Simulink schematic implementing the considered function.

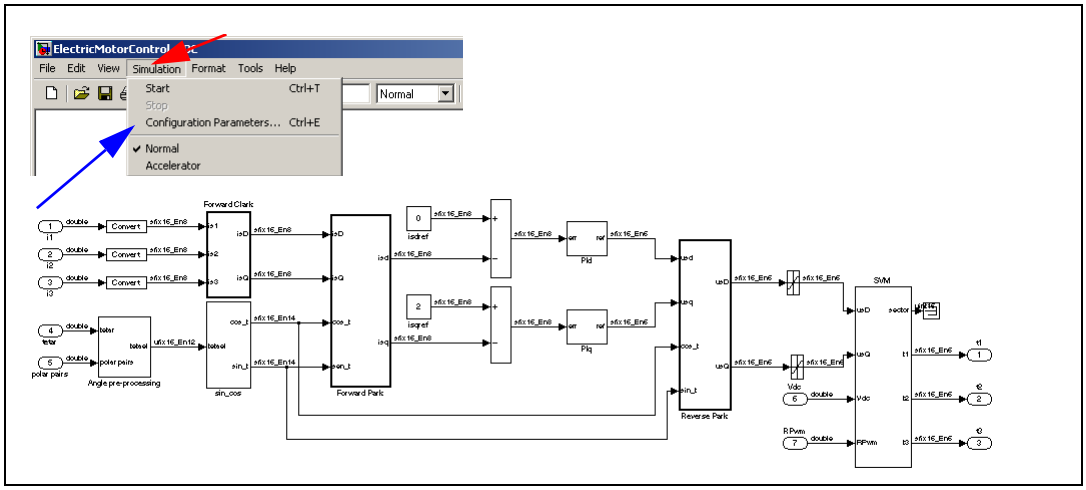
Note: for improving the readability of the auto generated C-code it's useful to include the schematic of each single function in a single subsystem.

In *Figure 26*, it's possible to see also the signal format. The model is now ready to be compiled in order to generate C code.

Step 2 - Real Time Workshop options configuration

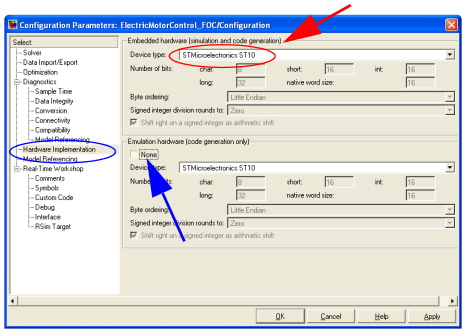
Selecting from the *Simulation* menu the "*Configuration Parameter"* pane all the options are shown:

Figure 26. Configuration parameter



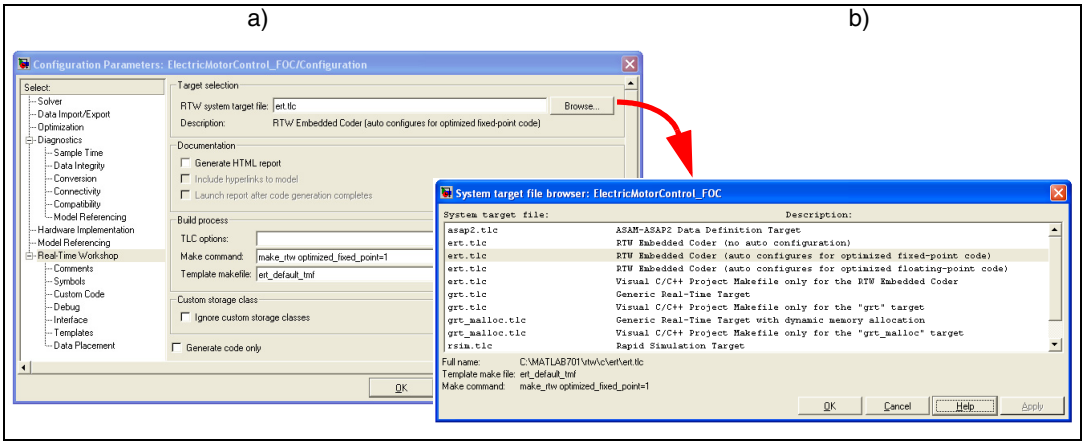
The first thing to do is selecting the *Hardware Implementation*, in our case ST10. In this way the format of data are chosen.

Figure 27. Hardware implementation



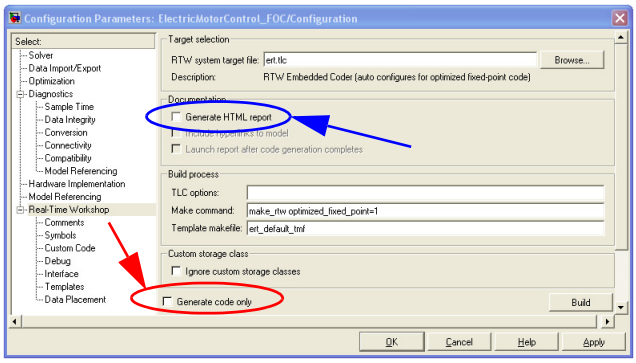
After the Real time Workshop options must be chosen. The first one is the *RTW system target.* As before said we choose the ERT optimum for fixed point precision (*Figure 28*).

Figure 28. RTW system target file



If only the code is needed (as in our case), the *Generate code only* box must be checked, (*Figure 29*), furthermore you could auto generate the *Generate HTML report* checking the apposite box.

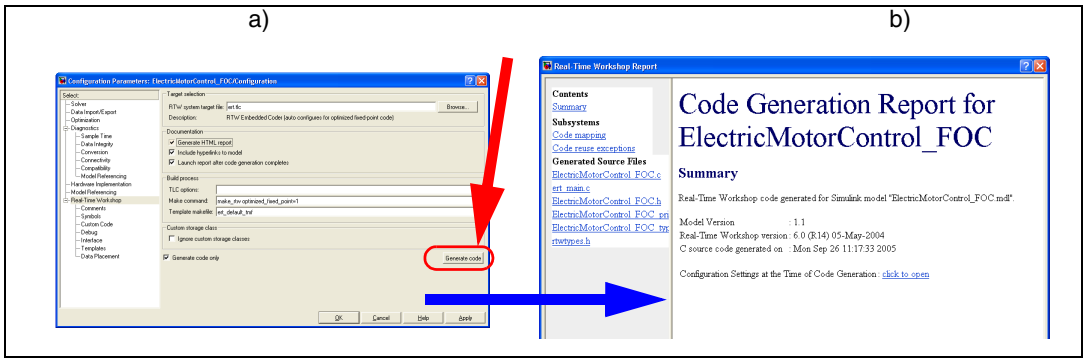
Figure 29. Generate HTML



In the *Comments* pane it's possible to define the verbosity level of the compiler and the comments that are automatically included in the generated C code.

Now everything is ready for generating code. Pushing the *"Generate code"* button the code generation starts with some verbose comments in the Matlab command windows, (*Figure 30*).

Figure 30. Generate code



When the process is completed the HTML report windows will appear generating the files:

- ElectricMotorControl_FOC.c
- ElectricMotorControl_FOC.h
- rtwtypes.h

Not all of them are useful for the next step code download.

5.5 Automatic configuration of RTW

Running RTWconfiguration.m file in the Command Window of Matlab available in the folder C:\FOC_Library2.0\options, a set of parameters is loaded to configure the RTW and associated with *a* given *filename.mdl*. For Automatic configuration the following steps has to be followed:

- Open *filename.mdl* file;
- Copy RTWconfiguration.m and config_RTW.mat in the actual working directory;
- Run “RTWconfiguration” from Command Window of Matlab.

In this way you’ll get active the “RTW_configuration” set to run and auto generate the C code.

6 Revision history

Table 4. Document revision history

Date	Revision	Changes
9-Mar-2007	1	Initial release.

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZED ST REPRESENTATIVE, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER’S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2007 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com