



Sinusoidal control simulink and software library of a PMSM

Introduction

This application note describes a software library for the electric motor control implementing a (SC) Sinusoidal Control on ST10 Microcontrollers.

Contents

1	Introduction	7
2	Sinusoidal control	8
2.1	PMSM	9
2.1.1	Mathematical model of the machine	10
2.2	Sinusoidal control structure	12
2.3	The Space vector modulation theory	13
2.3.1	The 3-phase inverter	13
2.3.2	The Space vector pulse width modulation	14
3	EMSC simulink library	17
3.1	Description	17
3.2	Using the simulink library	17
3.2.1	How to install simulink library	17
3.2.2	Test environment	17
3.3	Parameters format	18
3.4	PI	19
3.4.1	Description	19
3.4.2	Arguments	19
3.4.3	Algorithm	19
3.4.4	Simulink block	19
3.4.5	Test case	19
3.5	Phase advance	20
3.5.1	Description	20
3.5.2	Arguments	20
3.5.3	Algorithm	20
3.5.4	Simulink block	20
3.5.5	Test case	20
3.6	Space vector modulation look up table	21
3.6.1	Description	21
3.6.2	Arguments	21
3.6.3	Algorithm	21
3.6.4	Simulink block	21

	3.6.5	Test case	21
3.7		Rescaling	22
	3.7.1	Description	22
	3.7.2	Arguments	22
	3.7.3	Algorithm	22
	3.7.4	Simulink block	22
	3.7.5	Test case	22
4		Sinusoidal control software library	23
4.1		Description	23
4.2		Using the software library	23
	4.2.1	How to install software library	23
	4.2.2	Tool chain compatibility	23
	4.2.3	Calling a function	24
	4.2.4	ST10 MAC configuration	24
	4.2.5	Real time aspects	24
	4.2.6	Naming convention	24
	4.2.7	Test environment	24
	4.2.8	Sinusoidal control library benchmark	24
4.3		Library functions	26
	4.3.1	PI	26
	4.3.2	Description:	26
	4.3.3	Arguments:	26
	4.3.4	Algorithm:	26
	4.3.5	Notes:	26
	4.3.6	Test:	26
4.4		Phase advance	27
	4.4.1	phase_advance	27
	4.4.2	Description:	27
	4.4.3	Arguments:	27
	4.4.4	Algorithm:	27
	4.4.5	Notes:	27
	4.4.6	Test:	27
4.5		SVM modulation look up table	28
	4.5.1	SVM	28
	4.5.2	Description:	28

4.5.3	Arguments:	28
4.5.4	Algorithm:	28
4.5.5	Notes:	28
4.5.6	Test:	28
4.6	Rescaling	29
4.6.1	rescaling	29
4.6.2	Description:	29
4.6.3	Arguments:	29
4.6.4	Algorithm:	29
4.6.5	Notes:	29
4.6.6	Test:	29
5	C code auto generation	30
5.1	Overview	30
5.2	Steps to generate optimized C code	30
5.3	Real-Time Workshop	30
5.4	How to generate C code using real time workshop	31
5.4.1	Step 1 - Simulink schematic constructor	31
5.4.2	Step 2 - Real Time Workshop options configuration	31
5.5	Automatic configuration of RTW	33
6	Revision history	35

List of tables

Table 1. Time frames 16

Table 2. Data representation 18

Table 3. Sinusoidal Control library capabilities 25

Table 4. Document revision history 35

List of figures

Figure 1.	Vector diagram	8
Figure 2.	Cross-section of PMSM	9
Figure 3.	Stator current space vector and its components in (a,b,c)	10
Figure 4.	Phase motor with 2 pole pair	11
Figure 5.	Schema of sinusoidal control for PMSM-motor.	12
Figure 6.	3-phase power inverter scheme.	14
Figure 7.	Space vector diagram.	14
Figure 8.	SVM in the 1 st sector	15
Figure 9.	Example of a switching pattern in Sector 1.	16
Figure 10.	Simulink library structure	17
Figure 11.	PI block.	19
Figure 12.	Phase advance block	20
Figure 13.	SVM block.	21
Figure 14.	Rescaling block.	22
Figure 15.	File structure	23
Figure 16.	Flow chart	31
Figure 17.	Configuration parameter.	32
Figure 18.	Hardware implementation.	32
Figure 19.	RTW system target file	32
Figure 20.	Generate HTML	33
Figure 21.	Generate code	33

1 Introduction

This document describes a simulink and software library for the electric motor control implementing a (EMSC) Sinusoidal Control targeted to ST10 Microcontrollers.

The library consists of:

- Simulink Library;
- C optimized Software Library;
- ASM optimized Software Library.

The EMSC 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, by Real Time Workshop Embedded Coder, starting from EMSC Simulink library blocks, and then optimized in C and Assembler.

The SC software library for ST10 is fully compatible with SC simulink library.

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

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

2 Sinusoidal control

The Sinusoidal Control is a control strategy, characterized by sinusoidal waveforms so as to drive the three motor windings with currents that vary smoothly and sinusoidally, obtaining a negligible torque ripple and improved high-torque characteristics at both low and high speeds compared with other control strategies.

In order to generate a smooth sinusoidal modulation as the motor works, it is necessary an accurate rotor position feedback from a sensor, as encoder or similar devices.

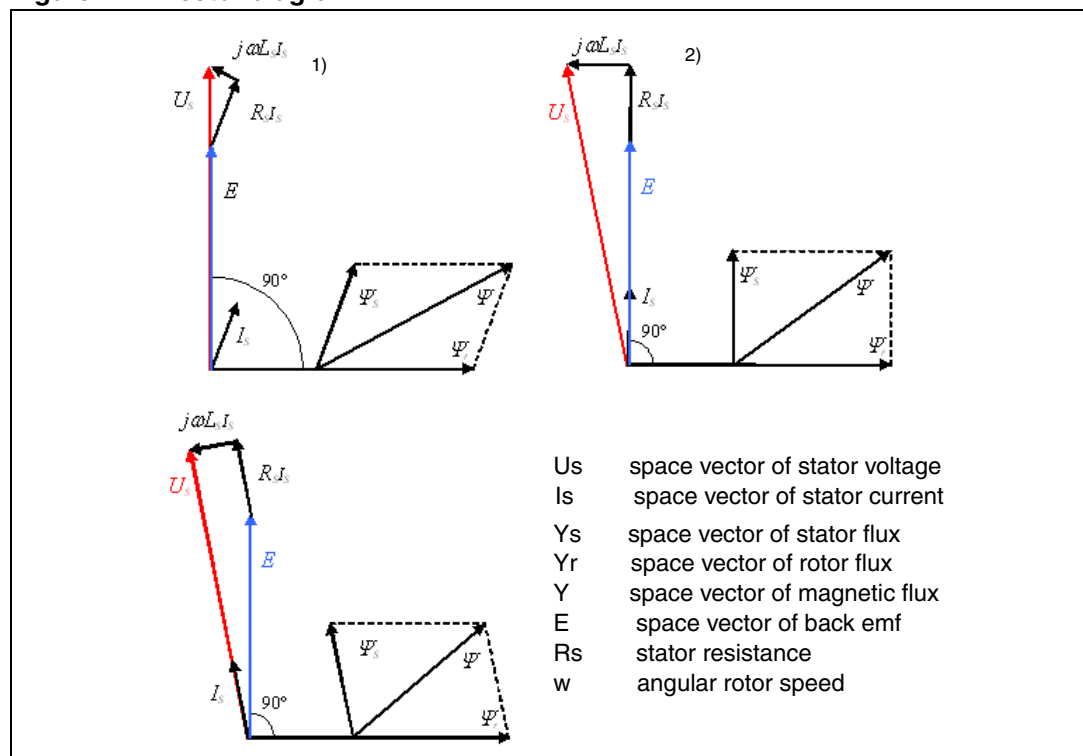
The position information is used to synthesize three modulation signals shifted by 120° degrees each other then multiplied by the command signal provided as output of a PI controller. These signals are used to feed the motor windings with the three voltages, proportional to desired command signal and appropriately phased, through a voltage source inverter (VSI).

There are three basic methods to control the voltage vector direction, each with different advantages and requests of measurement:

1. Vector of stator voltage is placed 90° relative to the vector of rotor permanent magnet flux;
2. Vector of stator current is placed 90° relative to the vector of rotor permanent magnet flux;
3. Voltage vector is kept in the direction of the current vector.

Figure 1 shows the vector diagram of each control strategy:

Figure 1. Vector diagram



In [Figure 5](#), the control of the motor speed forces the amplitude of the voltage vector obtained in output from the PI controller starting from the speed error while an offset is added to the actual rotor position in order to keep the voltage space vector in a desired position respect to the rotor flux space vector.

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.

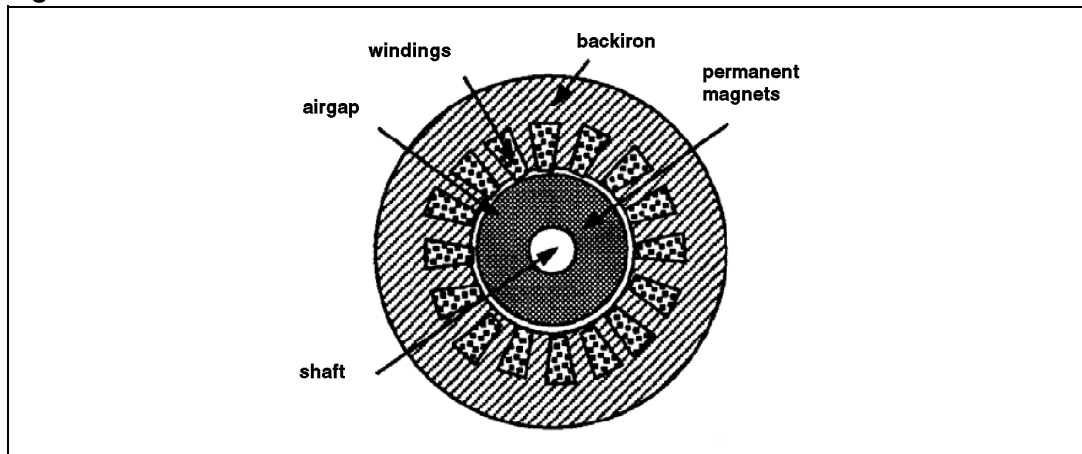
The DC machine fulfils these requirements but needs periodic maintenance.

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 below [Figure 2](#), they have a wound stator, similar to an induction machine, and a rotor with some permanent magnets instead of a wound rotor fed with DC current like the one that is used in the classical synchronous machines. 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



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 each kind, the high reliability control makes this type of machine a powerful system for electric vehicle application.

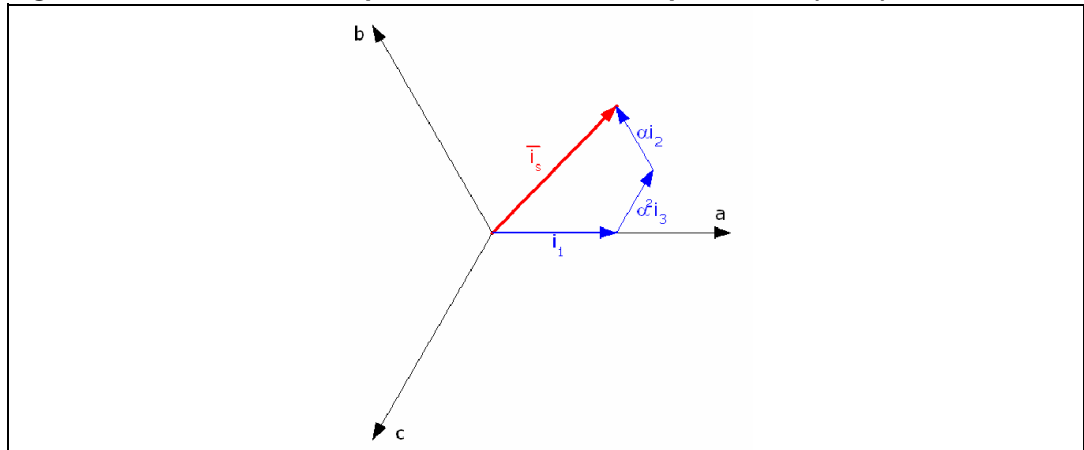
2.1.1 Mathematical model of the machine

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 defined 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 \bar{i}_s is defined by:

Equation 1
$$\bar{i}_s = \frac{2}{3} \cdot (i_{s1} + i_{s2} \cdot \alpha + i_{s3} \cdot \alpha^2)$$

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.

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

Equation 2

$$\bar{u}_s = R_s \cdot \bar{i}_s + L_s \cdot \frac{d\bar{i}_s}{dt} + \frac{d}{dt}(\psi_f \cdot e^{jp\theta_r})$$

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:

Equation 3

$$\bar{u}_s = \frac{2}{3} \cdot \left(u_1(t) + u_2(t) \cdot e^{j\frac{2\pi}{3}} + u_3(t) \cdot e^{j\frac{4\pi}{3}} \right)$$

[Equation 3](#) space vector of the stator voltage

Equation 4

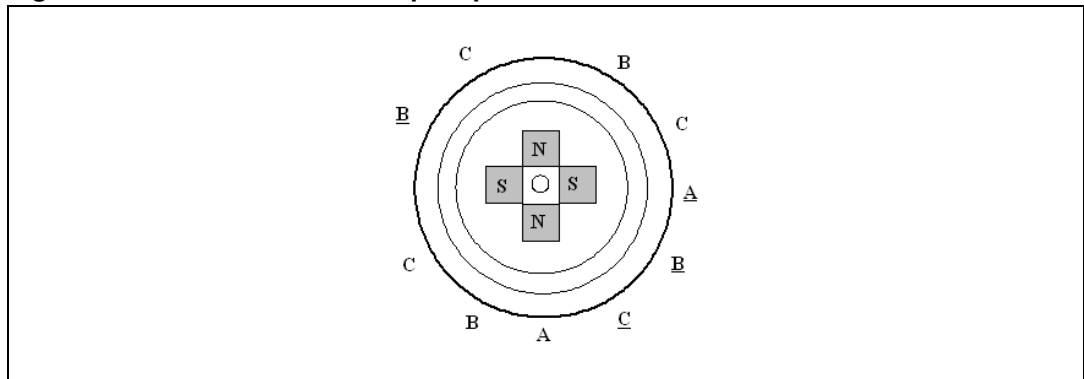
$$\bar{i}_s = \frac{2}{3} \cdot \left(i_{s1}(t) + i_{s2}(t) \cdot e^{j\frac{2\pi}{3}} + i_{s3}(t) \cdot e^{j\frac{4\pi}{3}} \right)$$

[Equation 4](#) shows space vector of the stator current

Note: *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:

Equation 5
$$\theta_e = \theta_r \cdot p$$

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

Equation 6
$$J \frac{d\omega_r}{dt} = T_e - (T_m + T_v)$$

and substituting the expression of the electric torque, it yields:

Equation 7
$$\frac{d\omega_r}{dt} = \frac{1}{J} \cdot (T_e - (T_m + T_v)) = \frac{1}{J} \cdot \left(\frac{3}{2} \cdot p \psi_f |i_s| \sin((\alpha_s - p \cdot \theta_r) - (T_m + T_v)) \right)$$

where:

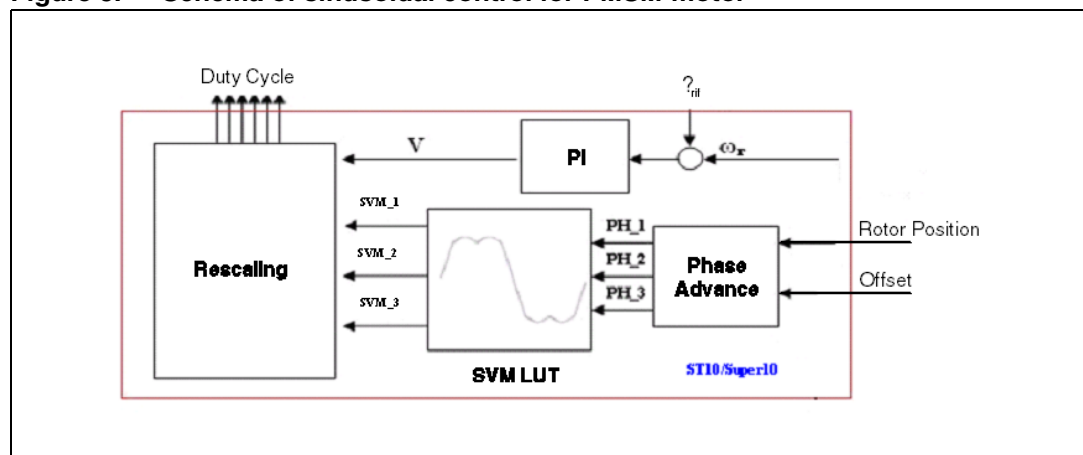
T_e	Electromagnetic torque
T_m	Mechanical torque
T_v	Viscose friction torque
α_s	Phase of the current space vector respect
J	Inertia momentum of the machine

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

2.2 Sinusoidal control structure

This application runs closed-loop control based on the difference between the reference speed ω_r^* and actual speed ω_r of the motor.

Figure 5. Schema of sinusoidal control for PMSM-motor



- The speed set point is established **via software** while the actual speed and the rotor position in electrical degrees are calculated by a Rotor Position Block using the signals coming from sensors (Hall Effect sensors, resolver or encoder).
- PI interpretations of the speed error determine the amplitude of the voltage signal wave, which in turn controls the motor speed.
- To impose the phase of the voltage space vector respect to the phase of the rotor flux space vector is used the 'Phase advance calculation Block' that adds an offset at the rotor position. In this way, it is possible to generate three signals, one for each motor phase, shifted by an angle in electrical degrees compared with calculated phase.
- Space Vector Modulation Look Up Table, is a table in which is stored an entire period of modulation. It uses the values of the Phases, calculated from the Phase Advance, to generate the three duty-cycle signals scaled between 2^{-15} and 2^{15} . The PWM module converts the duty cycle inputs into modulated square waves, which drive an external inverter.
- Rescaling, according to the duty-cycles from the SVM Look Up Table and the Amplitude of the voltage from the PI, re-scales the duty-cycles in order to let them to be compliant with the PWM peripherals of the microcontroller.

The chosen modulating signal, stored in the look up table, implements the SVM algorithm described in the following.

2.3 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 dynamic motors;
- higher efficiency (86%);
- improved torque management;
- better start up performance;
- constant torque, less torque ripple;
- improved dynamic 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.

2.3.1 The 3-phase inverter

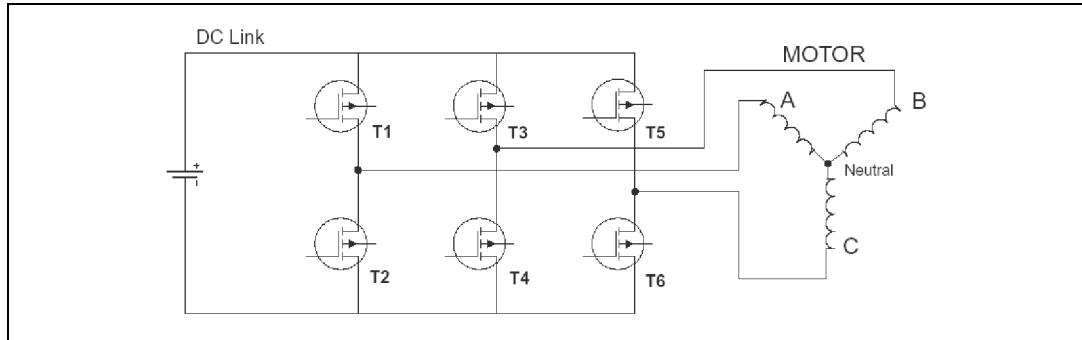
The inverter is a d.c. to a.c. converter. The [Figure 6](#) shows the structure of a typical 3-phase power inverter connected to the star motor windings, where V_{dc} is the DC Link voltage.

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.

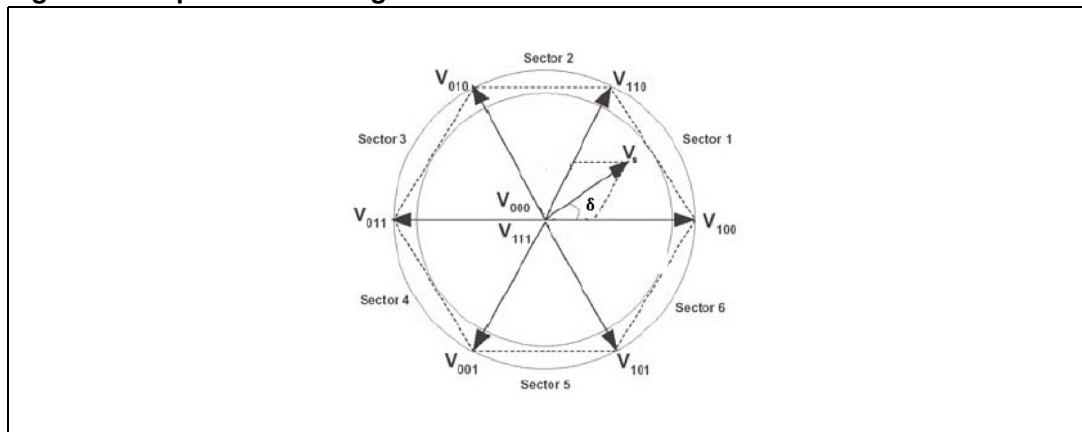


On base of the aforementioned conditions, the inverter has only eight permissible switching states of which six states apply a no-zero voltage to the motor windings and two states with (V0 and V7) zero volts when the motor is shorted through the upper or lower transistors.

It is useful to express the eight states of the inverter as space vectors: V_{0-7} 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](#):

Figure 7. Space vector diagram



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.

2.3.2 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](#)).

A reference voltage vector \bar{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 \bar{V}_s in this period), can be computed by the integral:

$$\text{Equation 8} \quad \int_0^{T_{PWM}} (\bar{V}_s \cdot dt) = \left(\int_0^{\delta} \bar{V}_{0/7} + \int_0^{\alpha} \bar{V}_k + \int_0^{\beta} \bar{V}_{k+1} \right)$$

from which it yields:

$$\text{Equation 9} \quad \bar{V}_s = \bar{V}_{0/7} \cdot \frac{\delta}{T_{PWM}} + \bar{V}_k \cdot \frac{\alpha}{T_{PWM}} + \bar{V}_{k+1} \cdot \frac{\beta}{T_{PWM}}$$

where:

$$\text{Equation 10} \quad T_{PWM} = \alpha + \beta + \delta$$

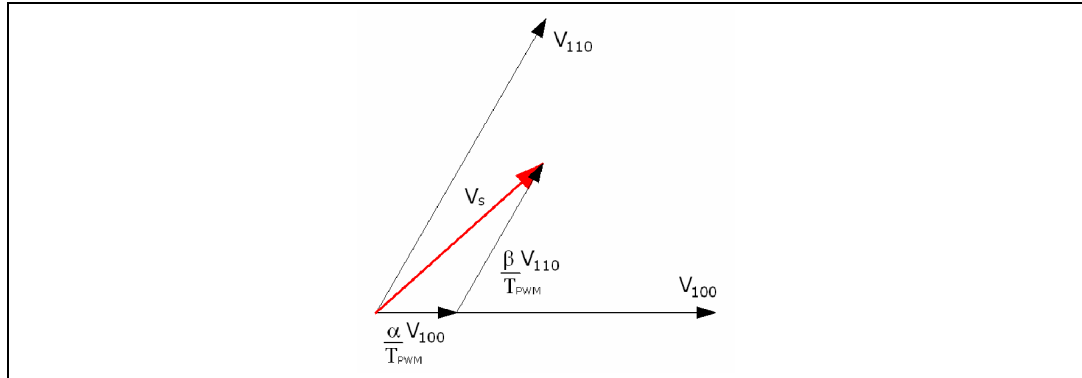
V_k and V_{k+1} ($k=0,..7$) are the vectors that bound the sector in which the reference vector is included, T_{PWM} is the switching period and α , β and δ are the time frames of V_k , V_{k+1} and V_0/V_7 vectors in that sector.

The resulting equation is:

$$\text{Equation 11} \quad |\bar{V}_s| = \frac{|\bar{V}_k \cdot \alpha + \bar{V}_{k+1} \cdot \beta|}{T_{PWM}}$$

For example, assuming that the vector \bar{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.

To create reference vector \bar{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 [Equation 11](#). It is possible to modulate the reference voltage vector and to apply the better switching pattern in terms of power dissipation, on the power switches of the inverter and to make it necessary to choose the strategy to apply the vector V_k and V_{k+1} (active vectors) in each sector.

We have same 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.

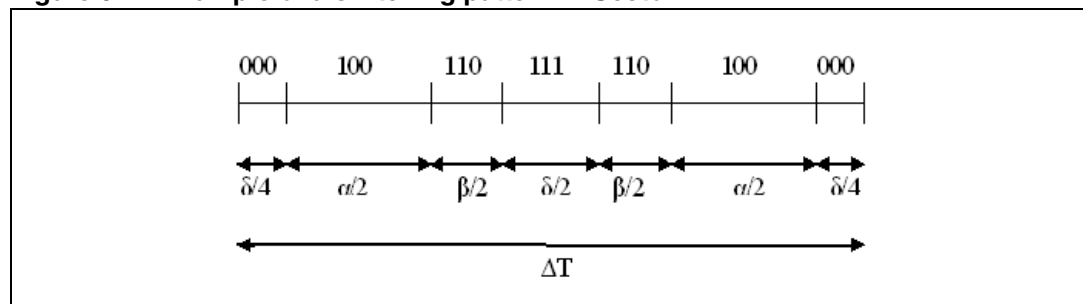
Literature demonstrates that, in order to reduce the number of commutation and switching losses, it is preferable to utilize a states sequence where the states are adjacent.

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

According to that, the chosen scheme 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 9. Example of a switching pattern in Sector 1



every component switches two times for every PWM period.

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

[Table 1](#) shows the time calculations to be done:

Table 1. Time frames

	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 EMSC simulink library

3.1 Description

The EMSC Simulink Library implements all needed functions for electric motor control using the following blocks, here listed:

- PI;
- Space Vector Modulation Look Up Table;
- Phase Advance;
- Rescaling;

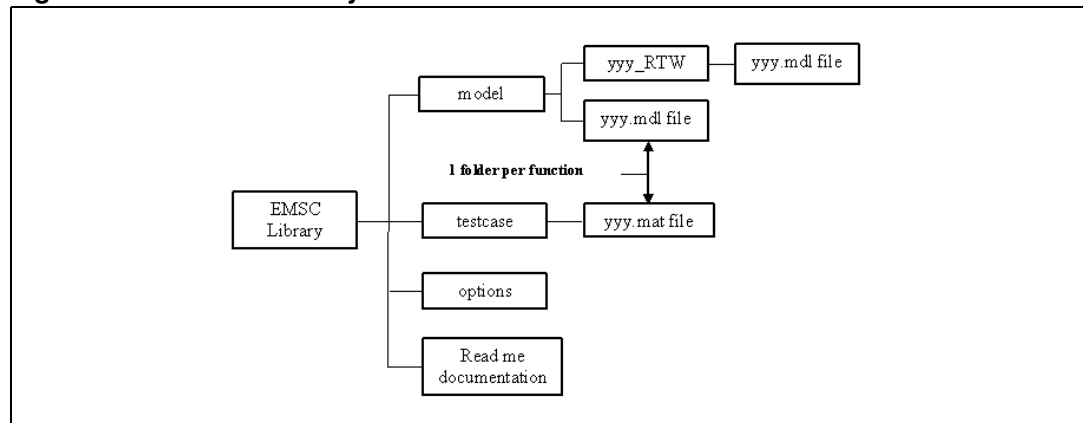
3.2 Using the simulink library

The library package consists of 2 main directories:

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

The file structure is the following:

Figure 10. 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 directory where you want the library to be copied into.

Note: it is required a license of Matlab version 7.0.0 or upward 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.

3.2.2 Test environment

.mat: it is possible to find the input and output data obtained by Simulink in the double format, stored in vectors that you preload in the Workspace of Matlab, opening the **.mdl** file.

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 EMSC library in Simulink and the optimized C-code have the same inputs/outputs so keeping the compatibility between them, using for the data coding a fixed point precision number representation.

In the [Table 2](#) the variables and their representations are listed:

Table 2. Data representation

variable	representation	description
speed_ref	sfix(16,5)	reference speed
speed_fdk	sfix(16,5)	measured speed
err	sfix(16,5)	error
PI_stato	sfix(32,26)	
kp	sfix(16,5)	proportional constant
ki	sfix(16,15)	integral constant
theta	ufix(16,16)	electrical angle
offset	ufix(16,16)	
PH_1	ufix(16,16)	theta 1
PH_2	ufix(16,16)	theta 2
PH_3	ufix(16,16)	theta 3
SVM1	sfix(16,15)	duty-cycle
SVM2	sfix(16,15)	duty-cycle
SVM3	sfix(16,15)	duty-cycle
Module	sfix(16,5)	Amplitude of the voltage signal wave
PWM_per1	uint16	duty-cycle
PWM_per2	uint16	duty-cycle
PWM_per3	uint16	duty-cycle

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

3.4 PI

3.4.1 Description

From the difference between the (ω_r^*) reference speed and (ω_r) actual speed (estimated by Hall sensors) the PI determines the amplitude of the voltage to feed the motor.

3.4.2 Arguments

err	(reference signal)-(measured signal);
speed_ref	reference speed;
speed_fdk	speed;
kp	proportional constant;
ki	integral constant;
Module	Amplitude of the voltage signal wave;

3.4.3 Algorithm

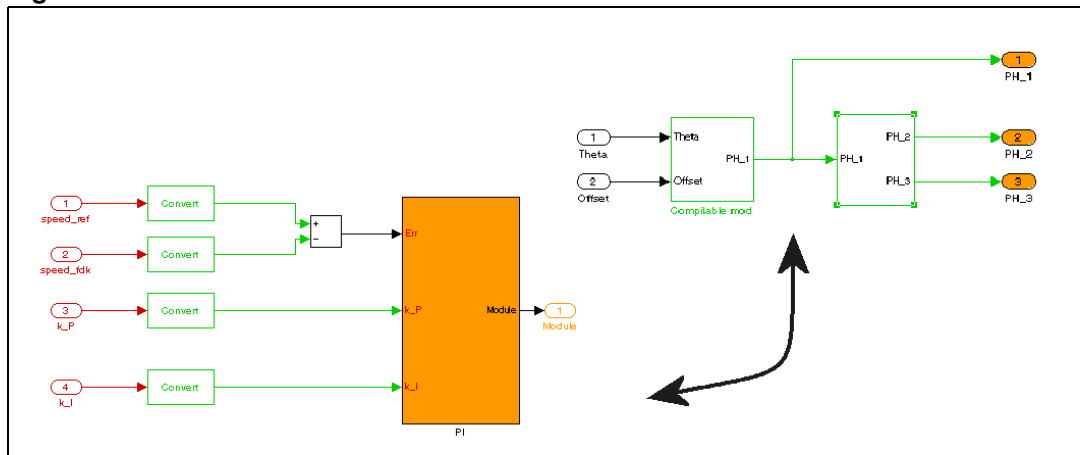
The following equations are implemented:

$$\text{Equation 12} \quad U_k = K_p \cdot e_k + K_i \cdot e_k + \sum_{n=0}^{k-1} e_n$$

3.4.4 Simulink block

As shown here below in the [Figure 11](#), the PI block, implemented in Simulink, receives in input the reference speed and calculated speed signals, in sfix(16,5) format and returns in output the voltage module in the same format.

Figure 11. PI block



3.4.5 Test case

You find in **pi_data.mat** file the inputs and outputs data to test this function.

3.5 Phase advance

3.5.1 Description

To impose the phase of the voltage space vector respect to the phase of the rotor flux space vector it adds an offset at the actual rotor position.

3.5.2 Arguments

$\theta_e = p\theta_r$	rotor position in electrical degrees;
offset	offset;
PH ₁	theta1;
PH ₂	theta2;
PH ₃	theta3.

3.5.3 Algorithm

The following equations are implemented:

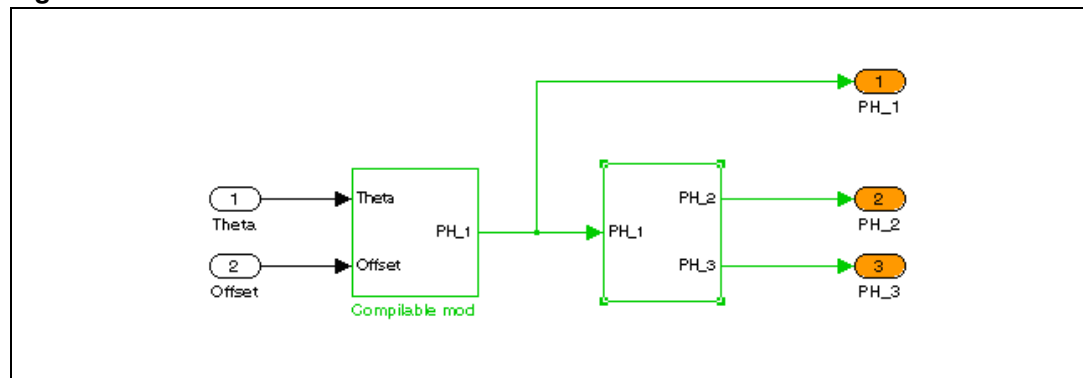
Equation 13
$$PH1 = \theta_e + \text{offset}$$

Equation 14
$$PH2 = PH1 - \frac{1}{3}$$

Equation 15
$$PH3 = PH1 + \frac{1}{3}$$

3.5.4 Simulink block

Figure 12. Phase advance block



3.5.5 Test case

You find in **phase_advance_data.mat** file the inputs and outputs data to test this function.

3.6 Space vector modulation look up table

3.6.1 Description

An entire period of modulation is stored in a look up table whose input is the value calculated from the Phase Advance Block, so an electrical angle, and in output it generates a duty-cycle signal, represented by a number scaled between 2^{-15} and 2^{15} .

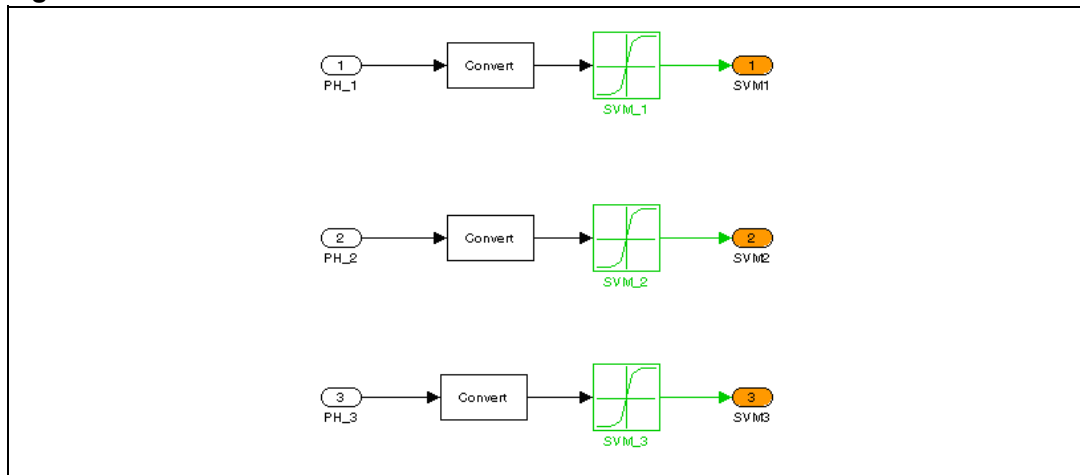
3.6.2 Arguments

PH_1	theta1;
PH_2	theta2;
PH_3	theta3;
SVM_1	duty-cycle;
SVM_2	duty-cycle;
SVM_3	duty-cycle;

3.6.3 Algorithm

3.6.4 Simulink block

Figure 13. SVM block



3.6.5 Test case

You find in **svm_lut_data.mat** file the inputs and outputs data to test this function.

3.7 Rescaling

3.7.1 Description

Rescale the duty-cycles from the SVM Look Up Table and the Amplitude of the voltage from the PI in order to let them to be compliant with the PWM peripherals of the microcontroller according to the duty-cycles

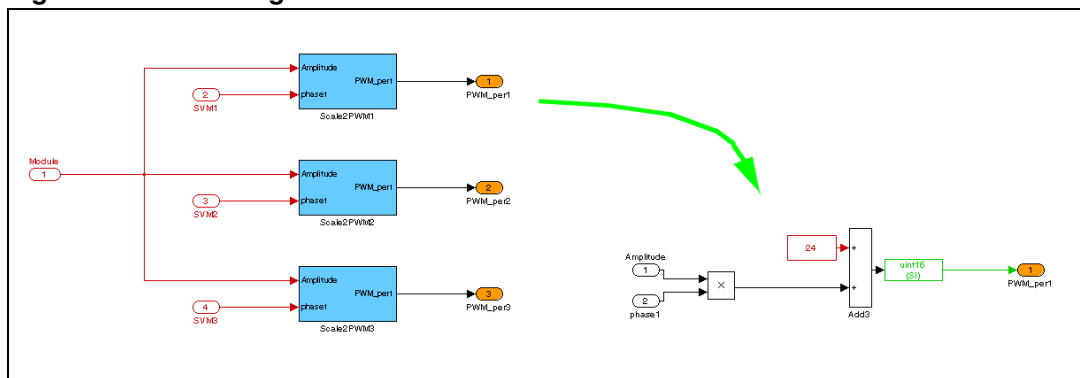
3.7.2 Arguments

Module	Amplitude of the voltage signal wave;
SVM ₁	duty-cycle;
SVM ₂	duty-cycle;
SVM ₃	duty-cycle;
PWM _{1_per1}	duty-cycle;
PWM _{2_per2}	duty-cycle;
PWM _{3_per3}	duty-cycle;

3.7.3 Algorithm

3.7.4 Simulink block

Figure 14. Rescaling block



3.7.5 Test case

You find in **rescaling_data.mat** file the inputs and outputs data to test this function.

4 Sinusoidal control software library

4.1 Description

The Sinusoidal Control Software library provides the functions for mixed “C” and Assembly programmers targeted to ST10 necessary to implement the (EMSC) electric motor control.

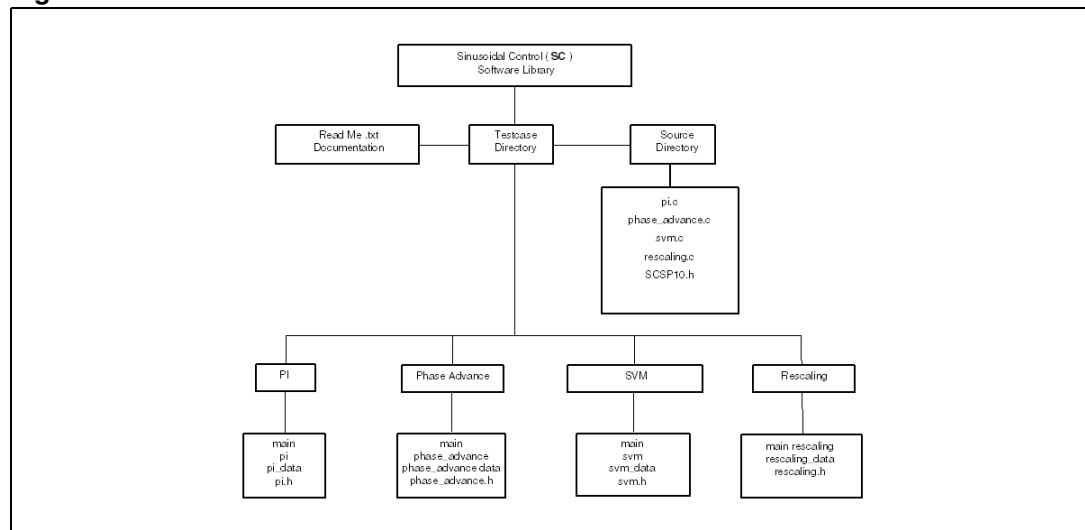
4.2 Using the software library

The two main directories of the library package are:

- Testcase directory;
- Source directory.

The file structure is the following one:

Figure 15. 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

EMSC library is compatible with Altium/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 “**SCSP10.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 (EMSC control). The implemented functions have been optimized with the use of MAC commands using the default configuration (the user have not to change the configuration registers of MAC) in order to improve the execution time performance.

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 Altium/Tasking tool chain).

4.2.6 Naming convention

The name of each functions refers to the name of the Simulink equivalent block, that it implements on micro, followed by underscore “step”.

Example: phase_advance_step

It translates the function of the Simulink block “Phase_Advance”.

For the input/output variables, the convention is:

name function underscore “U” underscore variable name for the inputs;

Example: Phase_Advance_U_Theta

name function underscore “Y” underscore variable name for the outputs

Example: Phase_Advance_Y_PH1.

4.2.7 Test environment

yyy_data.c : you find the input data vectors in **int** format and the output data vectors obtained by Simulink for the same function block in **int** 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 Sinusoidal control library benchmark

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

Table 3. Sinusoidal Control library capabilities

Function	Code size (bytes)	Nb cycles
PI	128	21
SVM	40	20
Phase Advance	22	16
Rescaling	152	20

4.3 Library functions

4.3.1 PI

```
pi_step(int32_T * PI_state, int16_T speed_ref, int16_T speed_fdk, int16_T kp, int16_T  
ki, int16_T* amplitude)
```

4.3.2 Description:

It implements classical PI scheme receiving in input the reference signal, the feedback signal and the state, calculating the error, defined like reference signal - calculated signal, and produces in output the reference voltage signal.

4.3.3 Arguments:

PI_state	PI_state;
speed_ref	Reference speed;
speed_fdk	Speed;
kp	Proportional constant;
ki	Integral constant;
amplitude	Amplitude of the voltage wave.

4.3.4 Algorithm:

4.3.5 Notes:

4.3.6 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.

4.4 Phase advance

4.4.1 phase_advance

```
phase_advance_step (uint16_T Phase_Advance_U_Theta, uint16_T  
                  Phase_Advance_U_Offset, uint16_T* Phase_Advance_Y_PH1,  
                  uint16_T* Phase_Advance_Y_PH2, uint16_T*  
                  Phase_Advance_Y_PH3)
```

4.4.2 Description:

The input signal waves are shifted by an angle that aims to optimize the efficiency of the motor.

4.4.3 Arguments:

Phase_Advance_U_Theta	Rotor position in electrical degrees;
Phase_Advance_U_Offset	Offset;
Phase_Advance_Y_PH1	Angle on phase1:Theta1;
Phase_Advance_Y_PH2	Angle on phase2:Theta2;
Phase_Advance_Y_PH3	Angle on phase3:Theta3.

4.4.4 Algorithm:

4.4.5 Notes:

4.4.6 Test:

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

4.5 SVM modulation look up table

4.5.1 SVM

SVM_step (uint16_T LUT_U_PH1, uint16_T LUT_U_PH2, uint16_T LUT_U_PH3,
int16_T* LUT_U_SVM1, int16_T* LUT_U_SVM2, int16_T* LUT_U_SVM3)

4.5.2 Description:

It uses the motor phases calculated from Phase_Advance Block as index to enter in a look up table and calculate the duty cycle.

4.5.3 Arguments:

LUT_U_PH1	Angle on phase1:Theta1;
LUT_U_PH2	Angle on phase2:Theta2;
LUT_U_PH3	Angle on phase3:Theta3;
LUT_U_SVM1	Duty-cycle on Phase1;
LUT_U_SVM2	Duty-cycle on Phase2;
LUT_U_SVM3	Duty-cycle on Phase3.

4.5.4 Algorithm:

4.5.5 Notes:

4.5.6 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.

4.6 Rescaling

4.6.1 rescaling

```
rescaling_step(int16_T Rescaling_U_Module, int16_T Rescaling_U_SVM1, int16_T  
              Rescaling_U_SVM2, int16_T Rescaling_U_SVM3, uint16_T*  
              Rescaling_Y_PWM_per1, uint16_T* Rescaling_Y_PWM_per2,  
uint16_T*      Rescaling_Y_PWM_per3)
```

4.6.2 Description:

Rescale the duty-cycles in order to let them to be compliant with the PWM peripherals of the microcontroller.

4.6.3 Arguments:

Rescaling_U_Module	Amplitude of the voltage wave;
Rescaling_U_SVM1	Duty-cycle on Phase1;
Rescaling_U_SVM2	Duty-cycle on Phase2;
Rescaling_U_SVM3	Duty-cycle on Phase3;
Rescaling_Y_PWM_per1	Duty-cycle on Phase1;
Rescaling_Y_PWM_per2	Duty-cycle on Phase2;
Rescaling_Y_PWM_per3	Duty-cycle on Phase3.

4.6.4 Algorithm:

4.6.5 Notes:

4.6.6 Test:

To test this function, include the **rescaling_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

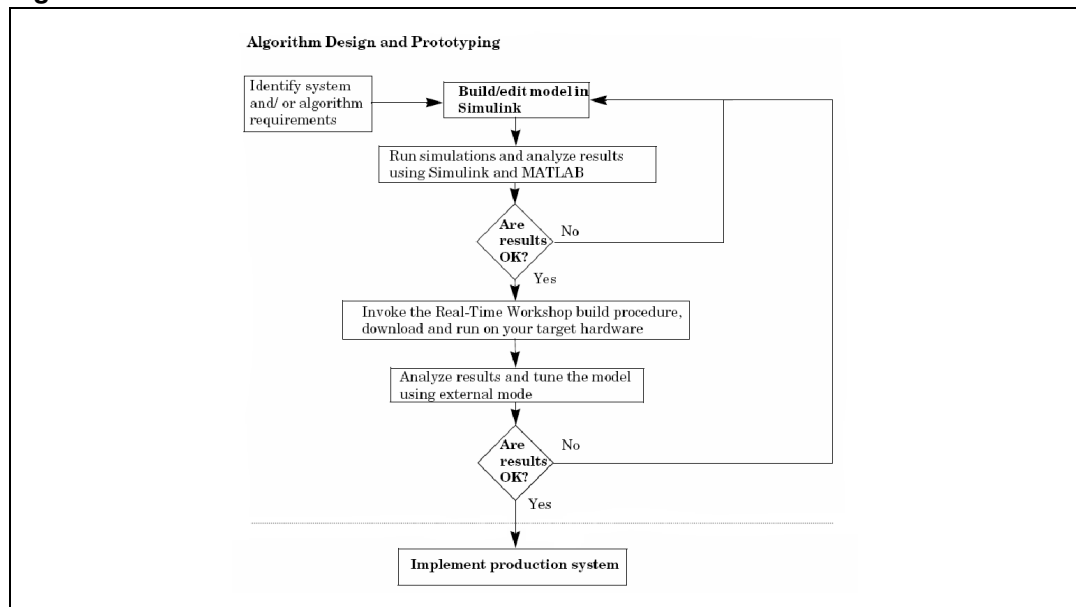
The Real-Time Workshop Embedded Coder is a separate, add-on product for use with Real-Time Workshop.

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 16. 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.

5.4.1 Step 1 - Simulink schematic constructor

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

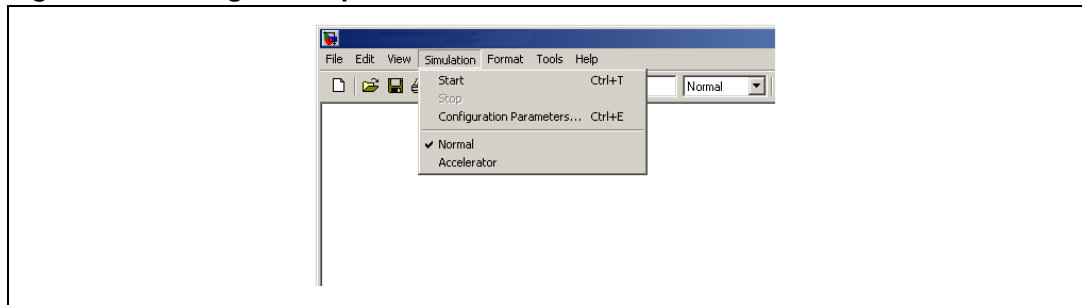
Note: for improving the readability of the auto generated C-code it's useful to include the schematic of each single function in a single subsystem.

The model is now ready to be compiled in order to generate C code.

5.4.2 Step 2 - Real Time Workshop options configuration

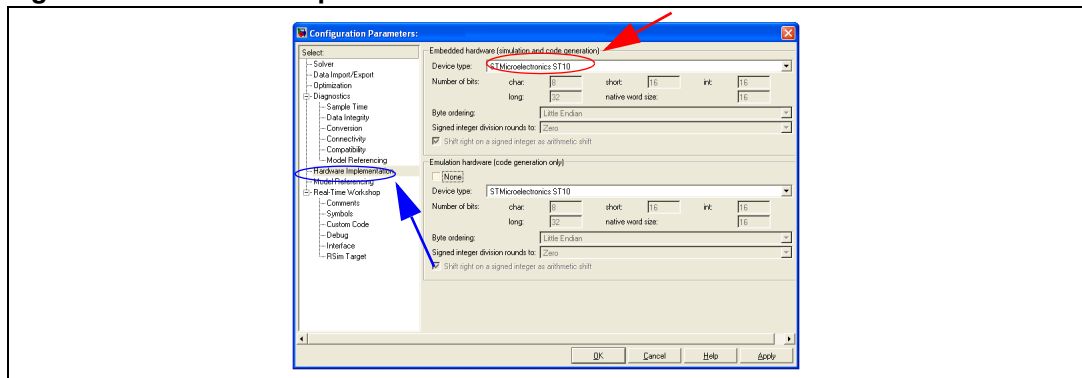
Selecting from the *Simulation* menu the "Configuration Parameter" pane all the options are shown:

Figure 17. Configuration parameter



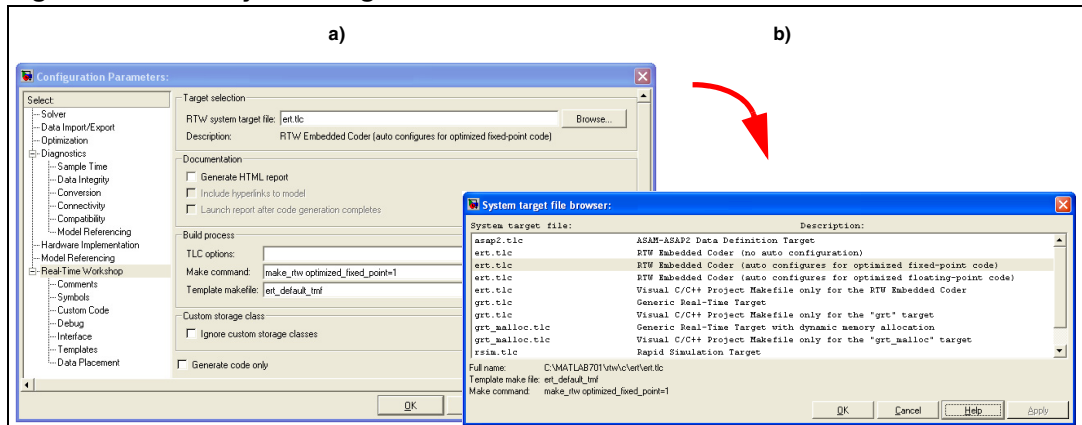
The first thing to do is selecting the *Hardware Implementation*, in our case ST10. In this way the format of data are chosen.

Figure 18. Hardware implementation



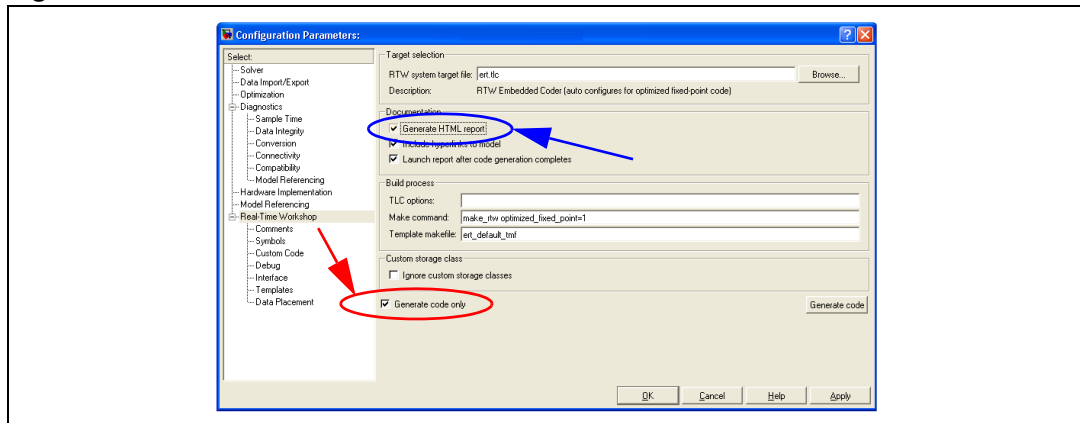
After the Real time Workshop options must be chosen. The first one is the *RTW system target*. As before said we choose the ERT optimum for fixed point precision (Figure 19).

Figure 19. RTW system target file



If only the code is needed (as in our case), the *Generate code only* box must be checked, (Figure 20), furthermore you could auto generate the *Generate HTML report* checking the apposite box.

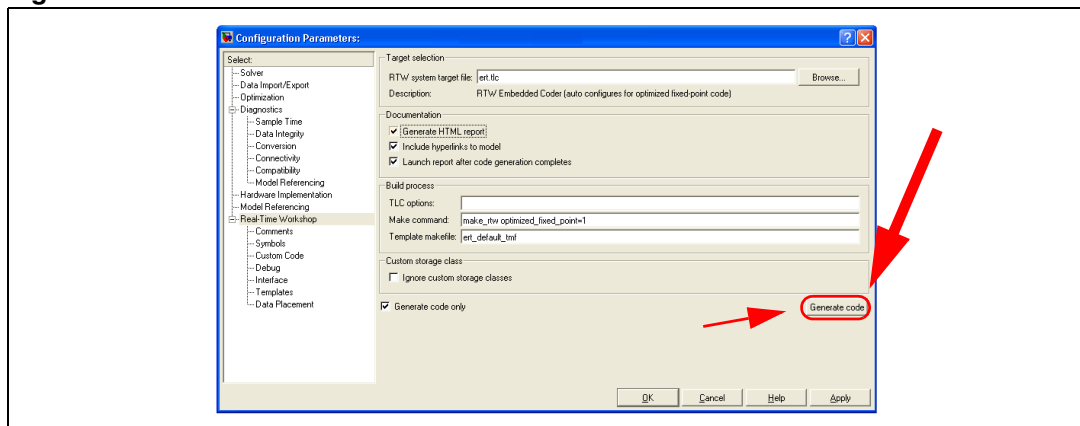
Figure 20. Generate HTML



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 generates some verbose comments in the Matlab command windows, as in [Figure 21](#).

Figure 21. Generate code



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

- ert_main.c
- filename.c
- 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:\EMSC_Library\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
09-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