UM0312
User manual

STR750

Sensored PMSM field-oriented control software library V1.0

Introduction

The STR750 microcontroller comes with a dedicated set of peripherals designed for field oriented control (FOC) of both permanent magnet DC/AC motors (PMDC/PMAC also called BLDC) and AC induction motors. It is delivered with two software libraries that allow you to develop applications to control these motors.

- The PMSM software library
- The AC induction motor software library

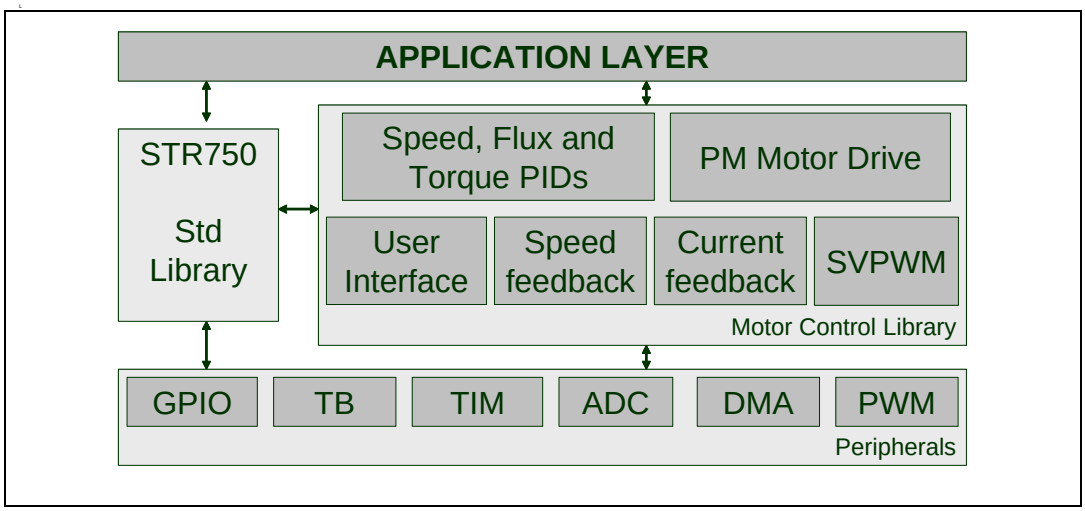
The complete library source files are delivered with the STR750-MCKIT, and are also available for free on the ST website www.stmcu.com, in the *Support* section. Check this site for the latest version of the library.

This user manual describes the PMSM software library required to control a permanent magnet synchronous motor with an encoder and a sinewave drive, open or closed loop. The AC induction motor software library designed to control an AC induction motor in sinewave mode with sensors is described in the UM0324 User Manual.

The PMSM software library is composed of several C modules, compatible with the IAR EWARM toolchain. The functions are grouped into several families, making this library an easy way to go through any PM sensored motor project development. Used in conjunction with the STR750-MCKIT starter kit, evaluation can be achieved in a very short time, because the software library spares you the trouble of studying the MCU in detail.

Overall software architecture

The figure below shows the architecture of the firmware. It uses the STR750 Standard Library extensively but it also acts directly on hardware peripherals when optimizations in terms of execution speed or code size are required.



PMSM software library 1.0 features (CPU running at 60MHz)

- Permanent magnet motor with encoder:
  - Open loop operation
  - Closed loop operation, PID regulation with 0.5ms to 127ms sampling time
- Current sampling method:
  - 2 isolated current sensors
  - 3 shunt resistors
- Current regulation for torque and flux control:

  PIDs, sampling time adjustable up to the PWM frequency
- 16-bit space vector PWM generation frequencies:
  - PWM frequency can be manually adjusted
  - Centered PWM pattern type
  - 11-bit resolution at 14.6kHz
- Free C source code and spreadsheet for look-up tables
- Motor control modules developed in accordance with MISRA C rules

Note: These figures are for information only; this software library may be subject to changes depending on the final application and peripheral resources. Note that it was built using robustness-oriented structures, thus preventing the speed or code size from being fully optimized.

The table below summarizes the memory required by the software library, as it is delivered (three-shunt topology for the current reading, encoder for speed feedback). These metrics include non motor control related code, implemented for demo purposes (such as user interface via LCD and joystick). Therefore, the figures provided should be considered as an estimation, which would be lower in the final application.

	ROM	RAM
Size (Kbyte)	27.5	3

Related documents:

Available on www.st.com:

- STR750 User Manual,
- STR750 Datasheet,
- STR750 Standard Library User Manual,
- STR7 Flash Programming Manual

Available on www.arm.com:
ARM7TDMI-S Rev.4 Technical Reference Manual ARM DDI 0234A

Contents

List of tables

List of figures

1 Getting started with tools

To develop an application for a PM sensored motor using the PMSM software library, you must set up a complete development environment, as described in the following sections. A PC running Windows 2000 (SP4) or Windows XP (SP2) is necessary.

1.1 Working environment

The PMSM software library was fully validated using the main hardware board included in the STR750-MCKIT starter kit (a complete inverter and control board). The STR750-MCKIT starter kit provides an ideal toolset for starting a project and using the library. Therefore, for rapid implementation and evaluation of the software described in this user manual, it is recommended to acquire this starter kit.

It is also recommended to install the IAR EWARM C toolchain which was used to compile the PMSM software library. With this toolchain, you do not need to configure your workspace. You can set up your workspace manually for any other toolchain. A free “kickstart editio”n of the IAR EWARM C toolchain with a 32Kb limitation, which is enough to compile and evaluate this library, can be downloaded from http://www.iar.com.

1.2 Software tools

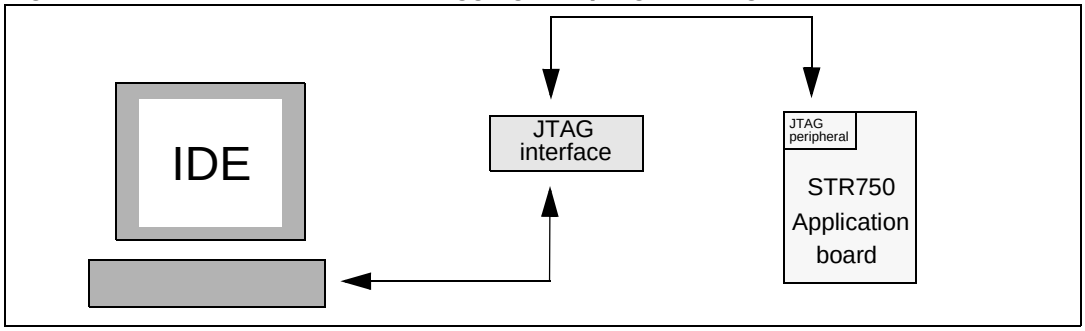
A complete software package consists of:

- A third-party integrated development environment (IDE).
- A third-party C-compiler: the EWARM development environment from IAR is pre-configured (30 days time-limited or 32Kb kickstart versions can be obtained upon request for evaluation).

  This library was compiled using the third party IAR EWARM C-toolchain. However, the choice of the C toolchain is left to your own appreciation. An IAR dedicated workspace can be directly opened in the root of the library installation folder (See *Section 1.3*).
- JTAG interface for debugging and programming

  Using the JTAG interface of the MCU, you can enter in-circuit debugging sessions with most toolchains. Each toolchain can be provided with an interface connected between the PC and the target application.

Figure 1. JTAG interface for debugging and programming



The JTAG interface can also be used for in-circuit programming of the MCU. Other production programmers can be obtained from third-parties.

1.3 Library source code

1.3.1 Download

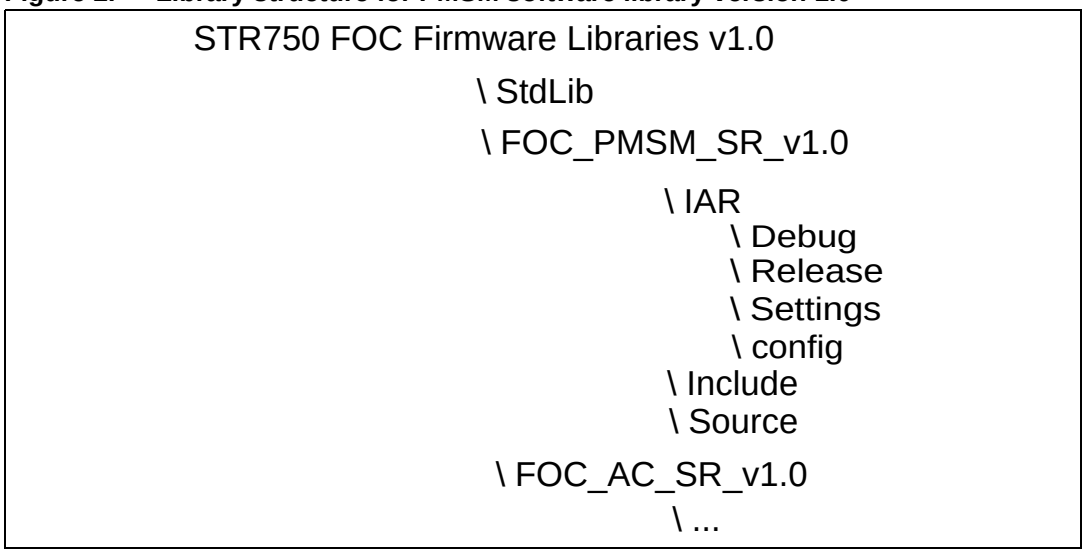
The complete library source files are available for free download on the ST web site www.stmcu.com, in the *Support* section.

Note: It is highly recommended to check for the latest releases of the library before starting any new development, and then to verify from time to time all release notes to be aware of any new features that might be of interest for your project. Registration mechanisms are available on ST web sites to automatically obtain updates.

1.3.2 File structure

The PMSM software library contains the workspace for the IAR toolchain. Once the download files are unzipped, the following library structure appears as shown in *Figure 2*.

Figure 2. Library structure for PMSM software library version 1.0



The **STR750 FOC Firmware Libraries v1.0** folder contains the firmware libraries for running both PMSM and AC induction three-phase sensored motors.

The **Stdlib** folder contains the standard library for the STR750. It provides standard routines for the configuration of the STR750 peripherals. This library is described in the STR75x Software Library User Manual UM0218.

The **Include** folder contains all the standard library header files: function prototypes, global variables, compiler directives.

The **Source** folder contains all library C source files and dedicated routines for the control of a permanent magnet motor with an encoder. These functions are described in this manual in *Section 4: Motor control library routines on page 31*.

The **IAR** folder contains the configuration file for the IAR C toolchain (kickstart, evaluation or standard edition).

1.3.3 Starting the IAR toolchain

When you have installed the toolchain, you can open the workspace directly from the dedicated folder, by double-clicking on the `PMSM Sensored.eww` file.

The file location is:

`\ FOC_PMSM_SR_v1.0\ IAR\ PMSM Sensored.eww`

1.4 Customizing the workspace for your STR75x device

The library described in this manual was written for the STR750FVT2. However, it works equally successfully with all the products in the STR75x family.

Using a STR750 sales type different from the STR750FVT2 may require some modifications to the library, according to the available features (some of the I/O ports are not present on low-pin count packages). Refer to the datasheet for details.

Depending on the memory size, the workspace may have to be configured to suit your STR75x MCU.

Table 1. Device summary

Features	STR755FRx	STR751FRx	STR752FRx	STR755FVx	STR750FVx
Flash - Bank 0 (bytes)	64K/128K/256K				
Flash - Bank 1 (bytes)	16K RWW				
RAM (bytes)	16K				
Operating Temp.	-40 to +85°C / -40 to +105°C (see *Table 41*)				
Common Peripherals	2 UARTs, 2 SSPs, 1 I2C, 3 timers 1 PWM timer, 38 I/Os 13 Wake-up lines, 11 A/D Channels			3 UARTs, 2 SSPs, 1 I^2C, 3 timers 1 PWM timer, 72 I/Os 15 Wake-up lines, 16 A/D Channels	
USB/CAN peripherals	None	USB	CAN	None	USB+CAN
Operating Voltage	3.3V or 5V	3.3V	3.3V or 5V		
Packages (**x**)	**T**=LQFP64 10x10, **H**=LFBGA64			**T**=LQFP100 14x14, **H**=LFBGA100	

1.4.1 Inkarm_xxx.xcl file (internal/external flash or RAM based project)

The `IAR\ config` folder contains 3 files:

- `Inkarm_flash.xcl`
- `Inkarm_smi.xcl`
- `Inkarm_ram.xcl`

These files are used as an extended command linker file and contain linker options. Memory areas, start address, size, and other parameters are declared here. It also contains the value assigned to the stack size for each ARM operating mode (for example, USER or FIQ. Refer to the ARM7TDMI-S Technical Reference Manual for more information).

The default extended linker file used in the standard library to configure the device for internal flash based resident firmware is `Inkarm_flash.xcl`. An extract of this file showing the definitions of heap and stack size is provided below. Depending on the project requirements, it may be necessary to manually edit the segment sizes.

```
// Embedded Flash (256/128/64Kbytes)
// The user has to change the flash memory length depending STR75xFxx
devices

// Code memory in flash
-DROMSTART=0x20000000
-DROMEND=0x2003FFFF   //0x2001FFFF;0x200FFFF

// Data memory
-DRAMSTART=0x40000000
-DRAMEND=0x40003FFF

........

//*****************************************************************************
*
// Stack and heap segments.
//*****************************************************************************

// Add size >0 for  ABT_Stack, UND_Stack if you need them.
// size must be 8 byte aligned.

-D_CSTACK_SIZE=0x200
-D_SVC_STACK_SIZE=0x20
-D_IRQ_STACK_SIZE=0x100
-D_FIQ_STACK_SIZE=0x40
-D_ABT_STACK_SIZE=0x0
-D_UND_STACK_SIZE=0x0
-D_HEAP_SIZE=0x400
```

Memory size modifications must be done according to the MCU specifications. Default settings are done for a 256Kb embedded flash memory. Depending on the project requirements, stack and heap segment size might also need to be changed.

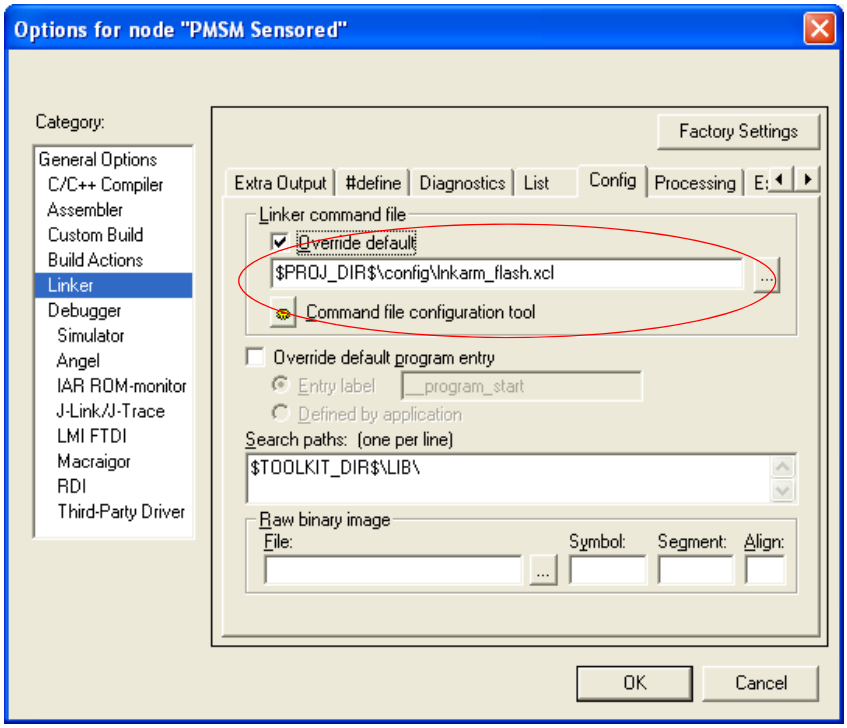
1.4.2 Extended linker file setting

As mentioned in the previous section, the IAR workspace is provided by default with the internal flash extended linker file `Inkarm_flash.xcl`. To modify the workspace to use a different file:

1. Open the IAR workspace by double-clicking on the `\ FOC_PMSM_SR_v1.0\ IAR \ PMSM Sensored.eww` file.
2. Go to the **Project** menu, select **Options...** then **Linker**, and select the **Config** sub-menu.

   The dialog box shown in *Figure 3* is displayed.
3. In the **Linker command file** section, tick the **Override default** checkbox, select the linker file you want to use, and then click **OK**.

Figure 3. Linker file selection



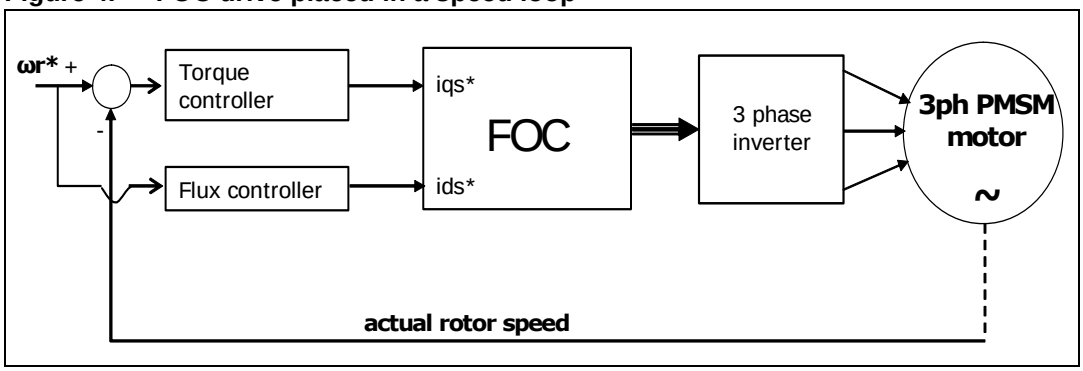
2 Getting started with the library

2.1 PMSM FOC drive quick introduction

The PMSM software library is designed to achieve the high dynamic performance in permanent magnet motor control offered by the field oriented control (FOC) strategy.

Through complex machine electrical quantity transformations, this well-established drive system optimizes the system efficiency, to the extent that it is able to offer decoupled torque (T_e) and magnetic flux (λ) control.

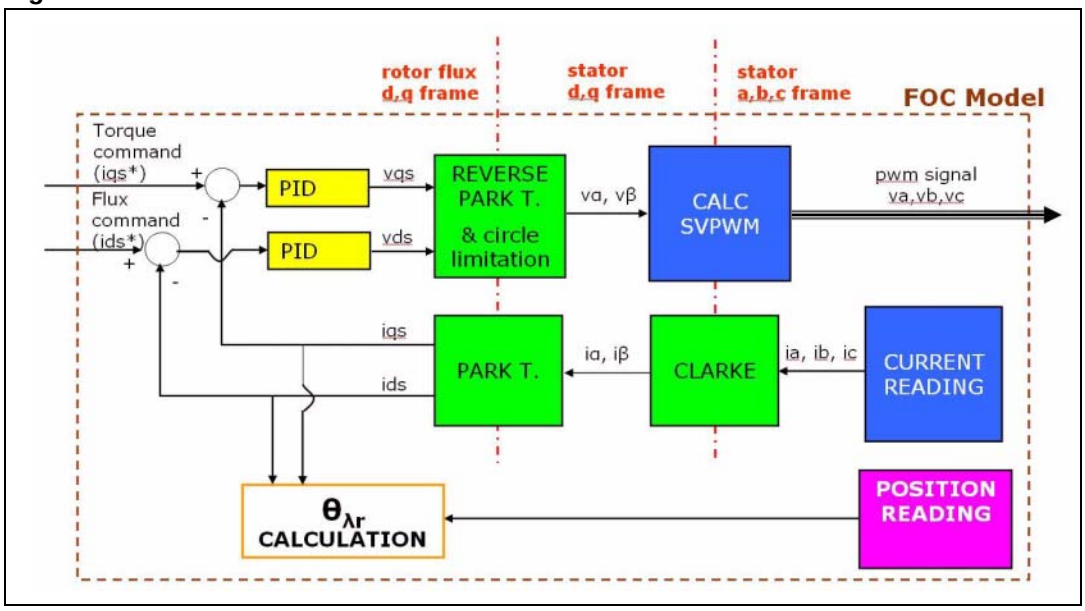
Figure 4. FOC drive placed in a speed loop



Basic information on field oriented structure and library functions is represented in *Figure 5*.

- The $\theta_{\lambda r}$ calculation block estimates the rotor position, which is essential to transformation blocks (Park, Reverse Park) for performing field orientation, so that the currents supplied to the machine can be oriented in quadrature to the rotor flux vector. More in depth information about reference frame theory is available in *Section 4.2.8 on page 54*.
- The space vector PWM block (SVPWM) implements an advanced modulation method that reduces current harmonics, thus optimizing DC bus exploitation.
- The current reading block allows you to measure stator currents correctly, using either shunt resistors or market-available isolated current Hall sensors (ICS).
- The speed-reading block handles the encoder signals in order to acquire properly rotor angular velocity or position.
- The PID-controller block implements a proportional, integral and derivative feedback controller, to achieve speed, torque and flux regulation.

Figure 5. PMSM FOC structure



2.2 Pre-checks and library configuration

It is quite easy to set up an operational evaluation platform with a drive system that includes the STR750-MCKIT (featuring the STR750 microcontroller, where this software runs) and a permanent magnet sensored motor.

This section explains how to quickly configure your system and, if necessary, customize the library accordingly.

Follow these steps to accomplish this task:

1. Gather all the information that is needed regarding the hardware in use (motor parameters, power devices features, speed/position sensor parameters, current sensors characteristics);
2. Edit, using an IDE, the `75x_MCconf.h` configuration header file (as explained in more detail in *Section 2.2.1*), and the following parameter header files,
   - `MC_PMSM_motor_param.h` (see *Section 2.2.2*);
   - `MC_encoder_param.h` (see *Section 2.2.3*).
   - `MC_Control_Param.h` (see *Section 2.2.4*),
3. Re-build the project and download it on the STR750 microcontroller.

2.2.1 Library configuration file: `75x_MCconf.h` file

The purpose of this file is to declare the compiler conditional compilation keys that are used throughout the entire library compilation process to:

1. Select which current measurement technique is actually in use (the choice is between three-shunt or ICS sensors, according to availability).
2. Enable or disable the derivative action in the speed controller or in the current controllers in accordance with expected performance and code size.

If this header file is not edited appropriately (no choice or undefined choice), you will receive an error message when building the project. Note that you will not receive an error message if the configuration described in this header file does not match the hardware that is actually in use, or in case of wrong wiring.

More specifically:

- `#define ICS_SENSORS`

  To be uncommented when current sampling is done with isolated current sensors (commented by default).

- `#define THREE_SHUNT`

  To be uncommented when current sampling is done with 3 shunt resistors (uncommented by default).

- `#define ENCODER`

  To be uncommented when the encoder is connected to the starter kit. Mandatory (uncommented by default) for the PMSM software library 1.0.

- `#define TACHO`

  To be uncommented when the tacho signal is connected to the starter kit. Not used (commented by default) in the PMSM software library.

- `#define Id_Iq_DIFFERENTIAL_TERM_ENABLED`

  To be uncommented when derivative terms for torque and flux control loop regulation (PID) are enabled (uncommented by default).

- `#define SPEED_DIFFERENTIAL_TERM_ENABLED`

  To be uncommented when the derivative term for speed control loop regulation (PID) is enabled (uncommented by default).

Once these settings have been done, only the required blocks will be linked in the project; this means that you do not need to exclude any C source files from the build.

Caution: When using shunt resistors for current measurement, ensure that the `REP_RATE` parameter (in `MC_Control_Param.h`) is set properly (see *Section 2.2.4* and *Section A.2: Selecting PWM frequency for 3 shunt resistor configuration on page 80* for details).

2.2.2 Permanent magnet motor parameters: `MC_PMSM_motor_param.h` file

The `MC_PMSM_motor_param.h` header file holds motor parameters which are essential to properly operate the vector drive. It provides the compiler with the number of pole pairs of the motor, and with the alignment settings (see *Figure 6*).

The following parameters must be defined in all cases:

- `#define POLE_PAIR_NUM`

  Specify here the number of pole pairs of your motor. For the SHINANO PM Sensored motor provided with the starter-kit, the default value is 2.

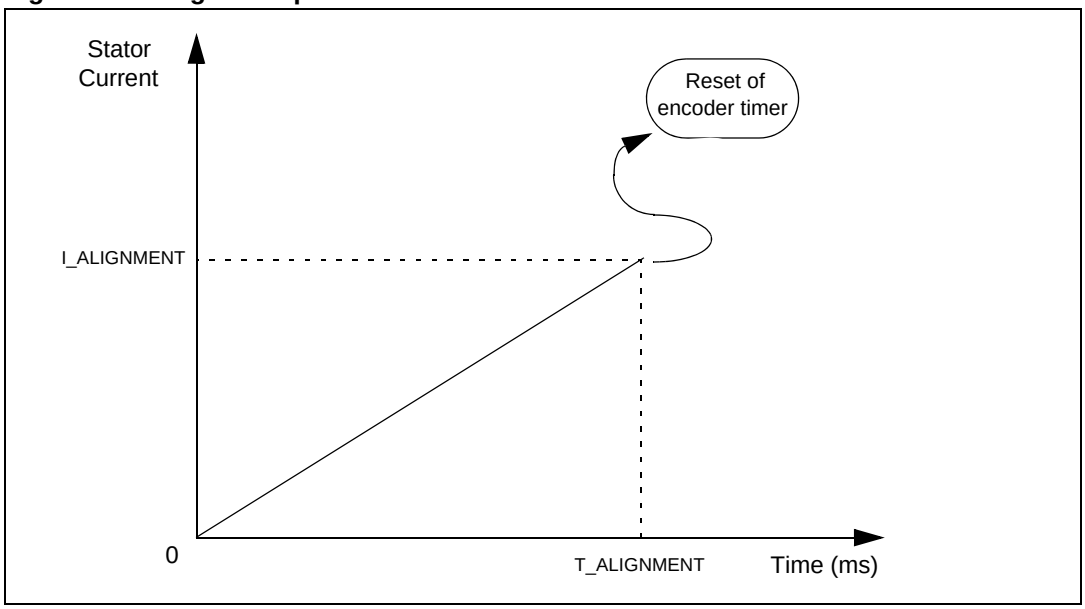
- #define T_ALIGNMENT

  Define here the time in ms for the rotor alignment.

- #define I_ALIGNMENT

  Define here the maximum current value at the end of the rotor alignment (the value is hardware dependent).

Figure 6. Alignment procedure



After the alignment procedure, a reset of the encoder timer is done in order to record the '0 degree' rotor angle reference.

2.2.3 Encoder parameters: `MC_ENCODER_param.h` file

The purpose of this header file is to provide the compiler with the encoder parameters.

- `#define TIMER0_HANDLES_TACHO`

  To be uncommented if the encoder outputs are connected to the TIMER0 input.
- `#define TIMER1_HANDLES_TACHO`

  To be uncommented if the encoder outputs are connected to the TIMER1 input.
- `#define TIMER2_HANDLES_TACHO`

  To be uncommented if the encoder outputs are connected to the TIMER2 input. (This is the default setting when using the STR750-MCKIT).
- `#define ENCODER_PPR`

  This statement contains the number of pulses per revolution of the motor encoder. For the SHINANO PM Sensored motor provided with the starter-kit, the default value is '400'.

2.2.4 Drive control parameters: `MC_Control_Param.h` file

The `MC_Control_Param.h header` file gathers parameters related to:

- Maximum modulation index, see *page 19*
- Power device control parameters, see *page 19*
- Flux and torque PID regulators sampling rate, see *page 19*
- Speed regulation loop frequency, see *page 19*
- Speed controller setpoint and PID constants, see *page 20*
- Torque and flux controller setpoints and PID constants, see *page 20*
- Linear variation of PID constants according to mechanical speed, see *page 21*

Maximum modulation index

The maximum modulation index for space vector PWM modulation guarantees a maximum limit to the PWM duty cycle output to the power stage in order to sample the current in the 3 phases correctly. This value depends on both the PWM switching frequency and the time required to compute the PID control loop for torque and flux regulation.

Only one definition out of the 9 declared must be uncommented. Refer to *Section 4.2.9: Circle limitation on page 56* for explanations on maximum modulation index.

```
/* DEFINE ONLY ONE max modulation index in the following list */
//#define MAX_MODULATION_100_PER_CENT // 100% max modulation index
//#define MAX_MODULATION_99_PER_CENT // 99% max modulation index
//#define MAX_MODULATION_98_PER_CENT // 98% max modulation index
//#define MAX_MODULATION_97_PER_CENT // 97% max modulation index
#define MAX_MODULATION_96_PER_CENT // 96% max modulation index
//#define MAX_MODULATION_95_PER_CENT // 95% max modulation index
//#define MAX_MODULATION_94_PER_CENT // 94% max modulation index
//#define MAX_MODULATION_93_PER_CENT // 93% max modulation index
//#define MAX_MODULATION_92_PER_CENT // 92% max modulation index
```

Power device control parameters

- `#define PWM_FREQ`

  PWM switching frequency. The value must be expressed in Hz (default 14000, 14Khz).

- `#define DEADTIME_NS`

  Define here, in ns, the dead time, in order to avoid shoot-through conditions.

Flux and torque PID regulators sampling rate

`#define REP_RATE`

Value to be fed into the repetition counter of the synchronizable PWM timer peripheral. The value (default 1) is 8-bit long, and provides the period frequency for current sampling and regulation. For more information, refer to the STR750 Datasheet Synchronizable Standard Timer, Repetition Counter Register section.

$$\text{Flux and torque PIDs sampling rate} = \frac{2 \cdot PWM\_FREQ}{REP\_RATE + 1}$$

In fact, because there is no reason for either executing the PMSM FOC algorithm without updating the stator currents values or for performing stator current conversions without running the PMSM FOC algorithm, in the proposed implementation the stator current sampling frequency and the PMSM FOC algorithm execution rate coincide.

Note: REP_RATE must be an odd number if currents are measured by shunt resistors (see Section A.2: Selecting PWM frequency for 3 shunt resistor configuration on page 80 for details); its value is 8-bit long;

Speed regulation loop frequency

`#define PID_SPEED_SAMPLING_TIME`

Sampling time for the PID speed control loop. This value is 8-bit long, and the sampling period is adjustable from 0 (500ms) to 255 (127ms). The default sampling time is 2ms.

Speed controller setpoint and PID constants (initial values)

- `#define PID_SPEED_REFERENCE`

  Mechanical speed reference in closed loop at start up. This is a signed 16-bit value, expressed with 0.1Hz resolution ('150' means 15Hz which is the default value). The sign gives the direction of the motor rotation.

- `#define PID_SPEED_KP_DEFAULT`

  Proportional coefficient gain for the speed control loop regulation. This is a signed 16-bit value, adjustable from **0 to 32767** (default 5400).

- `#define PID_SPEED_KI_DEFAULT`

  Integral coefficient gain for the speed control loop regulation. This is a signed 16-bit value, adjustable from **0 to 32767** (default 2000).

- `#define PID_SPEED_KD_DEFAULT`

  Derivative coefficient gain for the speed control loop regulation. This is a signed 16-bit value, adjustable from **0 to 32767** (default 7400).

Torque and flux controller setpoints and PID constants:

- `#define PID_TORQUE_REFERENCE`

  Torque reference value in open loop at start up. This is a signed 16-bit value (default 2500). The sign gives the direction of rotation of the motor. In closed loop, this value is computed automatically by the speed regulation loop.

- `#define PID_TORQUE_KP_DEFAULT`

  Proportional coefficient gain for the torque control loop regulation. This is a signed 16-bit value, adjustable from **0 to 32767** (default 15000).

- `#define PID_TORQUE_KI_DEFAULT`

  Integral coefficient gain for the torque control loop regulation. This is a signed 16-bit value, adjustable from **0 to 32767** (default 1000).

- `#define PID_TORQUE_KD_DEFAULT`

  Derivative coefficient gain for the torque control loop regulation. This is a signed 16-bit value, adjustable from **0 to 32767** (default 1400).

- `#define PID_FLUX_REFERENCE`

  Flux reference value. This is a signed 16 bits value (default 0). The modification of flux reference can help to increase the maximum speed of the motor, while the efficiency will be slightly decreased.

- `#define PID_FLUX_KP_DEFAULT`

  Proportional coefficient gain for the flux control loop regulation. This is a signed 16-bit value, adjustable from **0 to 32767** (default 10000).

- `#define PID_FLUX_KI_DEFAULT`

  Integral coefficient gain for the flux control loop regulation. This is a signed 16-bit value, adjustable from **0 to 32767** (default 1200).

- `#define PID_FLUX_KD_DEFAULT`

  Derivative coefficient gain for the flux control loop regulation. This is a signed 16-bit value, adjustable from **0 to 32767** (default 1000).

Linear variation of PID constants according to mechanical speed

When the linear coefficient computation is enabled in speed closed loop (see *Figure 40 on page 73*), the following set-points must be fed with appropriate values (coefficient computation is disabled by default in the library). Refer to *Section 5.2.2: Adjusting the speed regulation loop Ki, Kp and Kd vs the motor frequency on page 72*.

- `#define Freq_Min`
  Linear curve for speed control loop, minimum frequency, set-point number 1.
- `#define Ki_Fmin`
  Linear curve for speed control loop, integral coefficient gain, set-point number 1.
- `#define Kp_Fmin`
  Linear curve for speed control loop, proportional coefficient gain, set-point number 1.
- `#define Kd_Fmin`
  Linear curve for speed control loop, derivative coefficient gain, set-point number 1.
- `#define F_1`
  Linear curve for speed control loop, intermediate frequency, set-point number 2.
- `#define Ki_F_1`
  Linear curve for speed control loop, integral coefficient gain, set-point number 2.
- `#define Kp_F_1`
  Linear curve for speed control loop, proportional coefficient gain, set-point number 2.
- `#define Kd_F_1`
  Linear curve for speed control loop, derivative coefficient gain, set-point number 2.
- `#define F_2`
  Linear curve for speed control loop, intermediate frequency, set-point number 3.
- `#define Ki_F_2`
  Linear curve for speed control loop, integral coefficient gain, set-point number 3.
- `#define Kp_F_2`
  Linear curve for speed control loop, proportional coefficient gain, set-point number 3.
- `#define Kd_F_2`
  Linear curve for speed control loop, derivative coefficient gain, set-point number 3.
- `#define Freq_Max`
  Linear curve for speed control loop, maximum frequency, set-point number 4.
- `#define Ki_Fmax`
  Linear curve for speed control loop, integral coefficient gain, set-point number 4.
- `#define Kp_Fmax`
  Linear curve for speed control loop, proportional coefficient gain, set-point number 4.
- `#define Kd_Fmax`
  Linear curve for speed control loop, derivative coefficient gain, set-point number 4.

2.3 Running your own motor

As a starting point, the open loop mode should be used for the first trials (fixed speed Ki, Kp and Kd coefficients are applied). Low-current alignment values should be used also and then increased smoothly step by step. Sufficient alignment time should be applied in order to avoid any oscillations while resetting the encoder timer at the end of the ramp (see `I_ALIGNMENT` and `T_ALIGNMENT` settings in *Section 2.2.2: Permanent magnet motor parameters: MC_PMSM_motor_param.h file on page 17*).

2.4 Closed loop operation and PID settings

To run a motor in standalone closed loop, the first step should be to run the system with fixed PID speed parameters and modify them while the motor is running in order to define a working range (linear coefficient computation disabled, see *Disabling the linear curve computation routine, 75x_it.c module on page 73*).

The second step is, for a given target mechanical speed, to fine-tune all the PID speed parameters most adequate for this speed. For each target speed, these values should be recorded in the form of a table, which will be used by the STR750 standalone firmware.

You should collect data for 4 speeds: the min and max speeds, and 2 intermediate speeds of your choice. The STR750 firmware will then make a linear extrapolation of these parameters between the 4 specified speeds to ensure smooth operation (linear coefficient computation enabled, see *Figure 40 on page 73*).

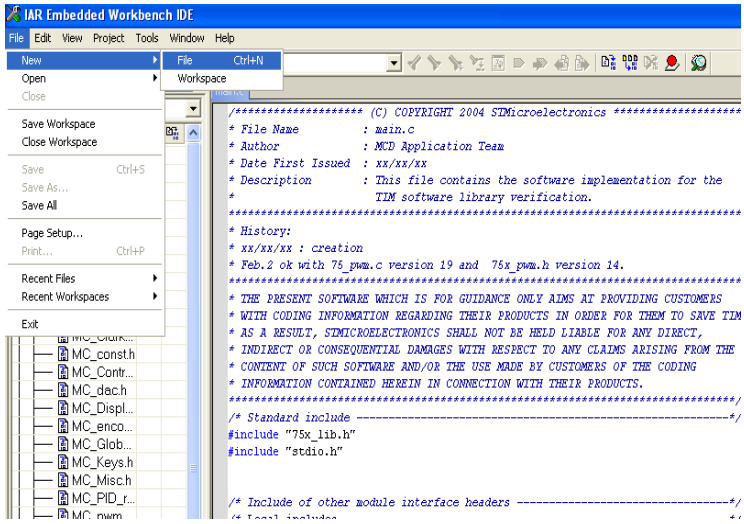
2.5 How to define and add a module

This section explains how you can create your own library modules to enhance the functionalities offered by the PMSM software library.

1. Create a new file.

   You can either copy and paste an existing file and rename it, or in the **File** menu, choose **New**, then click the **File** icon and save it in the right format (`*.c`, `*.h` extension), as shown in *Figure 7*.

Figure 7. Creating a new file

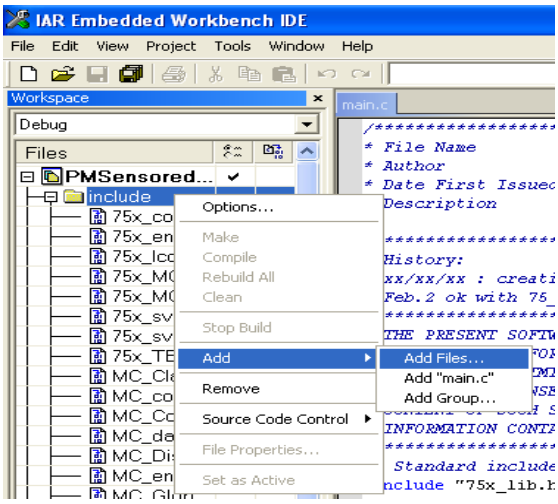


2. Declare the new file containing your code in the toolchain workspace.

   To do this, simply right-click in the workspace folder, then choose the **Add Files** sub-menu. The new file is automatically added to the workspace and taken into account for the compilation of the whole project.

The procedure of adding the module to the project is very easy with the IAR Embedded Workbench, as the makefile and linking command files are automatically generated. When rebuilding the library, the configuration files are updated accordingly.

Figure 8. Adding a file to the workspace



3 Running the demo program

This section assumes that you are using the STR750-MCKITmotor control starter kit.

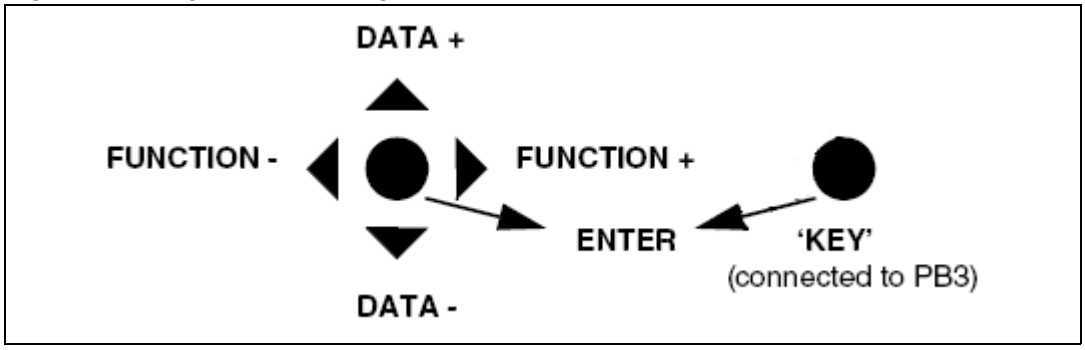
The demo program is intended to provide examples on how to use the software library functions; it includes both open speed loop and closed speed loop operations (hereafter simply referred to as Open Loop and Closed Loop), with the possibility of changing different parameters on the fly.

The default configuration allows the use of three shunt resistors for current reading and encoder for speed feedback. Refer to *Section 3.3 on page 27* for setting up the system when using ICS.

When the application is started, it first shows a welcome message and switches shortly to the main screen. Use the joystick and the button labelled **KEY** to navigate between the menus.

Key assignments are shown in *Figure 9*.

Figure 9. Key function assignments



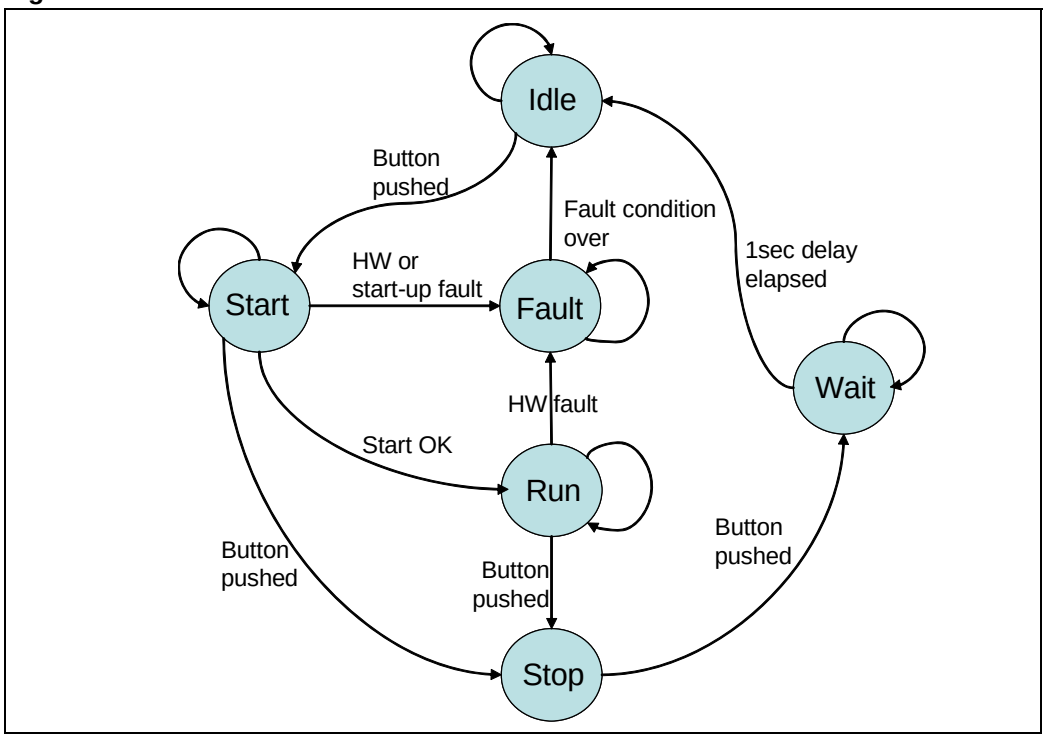
A simple state machine handles the motor control tasks in the main loop, as well as basic monitoring of the power stage. This state machine does not differentiate open loop from closed loop control. It is described in *Figure 10*.

The power stage is monitored using the ADC peripheral and the PWM peripheral Emergency Stop (ES) input to watch the following conditions:

- Heatsink over-temperature (ADC channel AIN6 and ES input),
- DC bus over/under-voltage (on ADC channel AIN7),
- Over-current protection (ES input).

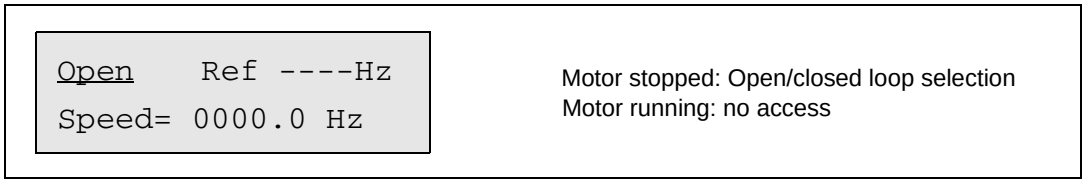
Any of these three conditions will cause the PWM to be stopped and the state machine to go into FAULT state before coming back to IDLE state. Depending on the source of the fault, an error message is also displayed on the LCD during FAULT state.

Figure 10. `Main.c` state machine

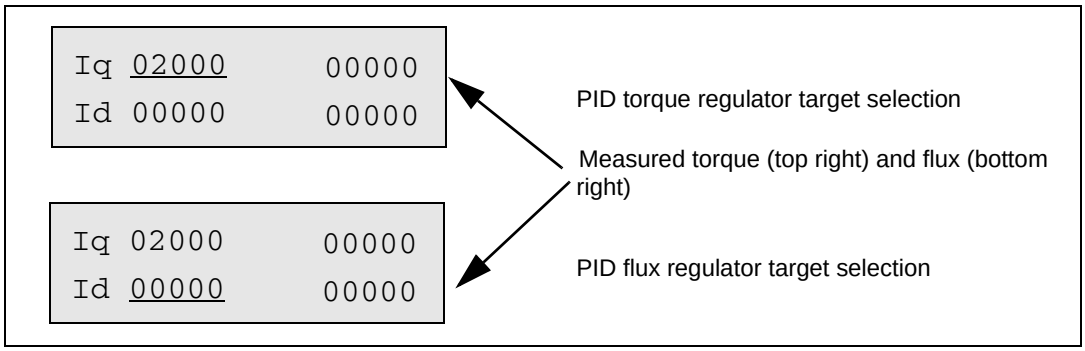


3.1 LCD display in OPEN loop mode

The following sections provide a summary of the screen access and settings in open loop; blinking items are shown underlined.



Switching from open to closed loop operation and vice versa is done by moving the joystick up or down while the first menu shown in the above figure is displayed and the motor is stopped. Moving the joystick left or right in these circumstances allows changing the context into the second menu where it is possible to modify both the torque and flux reference.

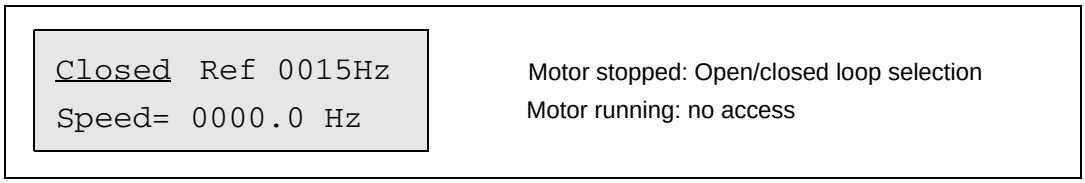


Finally press either the KEY button or the joystick to start the motor (main state machine will move from IDLE to START state).

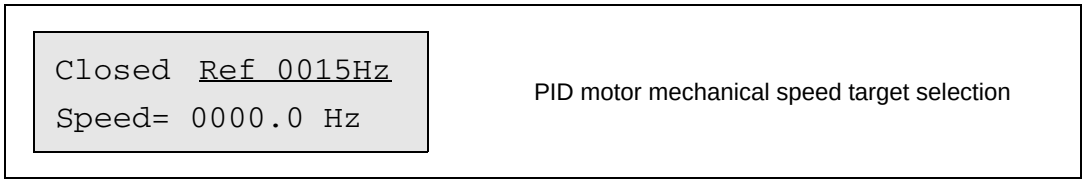
The ramp up strategy is illustrated in *Figure 6: Alignment procedure on page 18*. Basically, the applied stator current reference reaches the I_ALIGNMENT value that is defined in 'MC_Control_Param.h' following a linear ramp. After a programmed delay (T_ALIGNMENT), the torque and flux references become adjustable on the fly from the joystick.

3.2 LCD display in CLOSED loop mode

The following sections provide a summary of the screen access and settings in closed loop; blinking items are shown underlined.

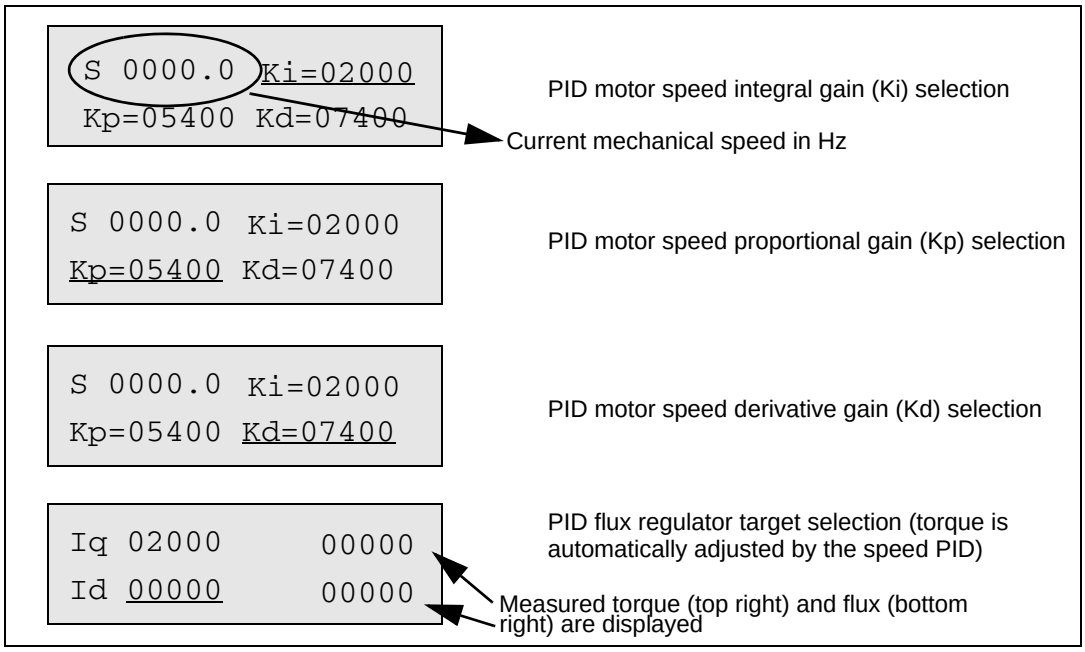


Switching from open to closed loop operation and back is done by moving the joystick up or down while the first menu shown in the above figure is displayed and the motor is stopped.



In closed loop operation, you can vary the target speed by moving the joystick up or down while the PID motor speed target selection menu is displayed.

The demo program also allows real-time tuning of the speed PID regulator coefficients:

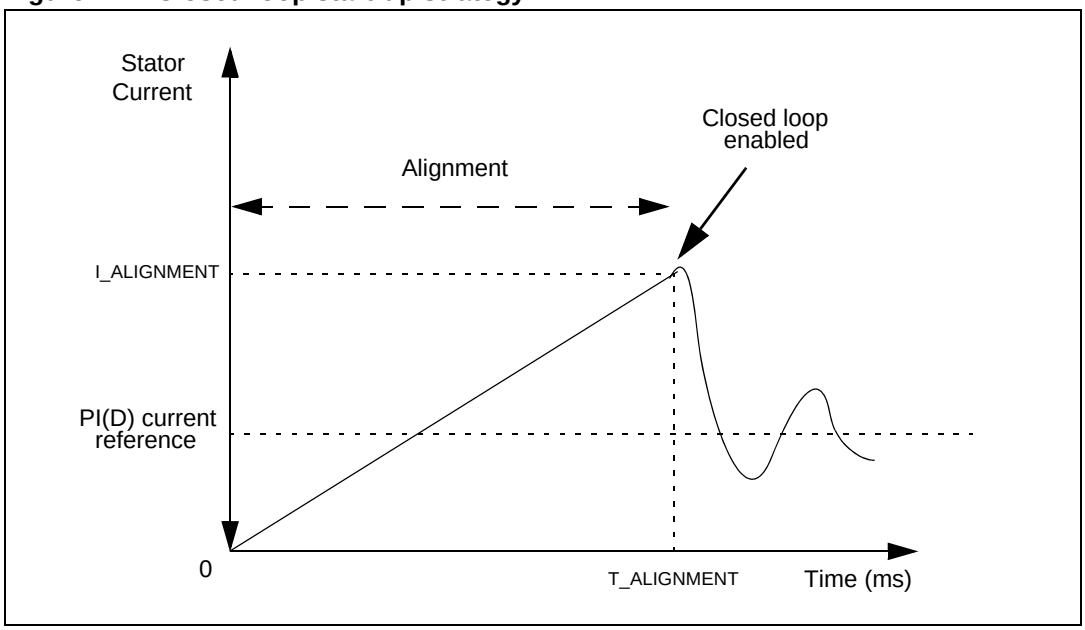


In closed loop, only the torque reference is the output of the speed PID regulator. Although you cannot act directly on the torque reference, you can set the flux reference and also observe both the measured flux and torque components; varying the flux reference can help to increase the motor speed while the system efficiency decreases (also known as field weakening).

As in open loop, pressing the joystick or the **KEY** button will start the motor.

The speed PID regulator is enabled and takes control of the torque reference right after the linear ramp-up alignment process as shown in *Figure 11*.

Figure 11. Closed loop start-up strategy



3.3 Setting up the system when using ICS sensors

The default configuration provides for the use of three-shunt resistors. *Section 3.3.1* describes how to change the firmware configuration from three-shunt resistors to two ICS stator current reading. This section gives you information about how to provide the STR750 with ICS feedback signals and to customize the firmware to use a different hardware.

Caution: When using two ICS for stator current reading, you must ensure that the sensors output voltage range is compatible with the STR750 supply voltage.

3.3.1 Connecting the two ICS sensors to the motor and to STR750

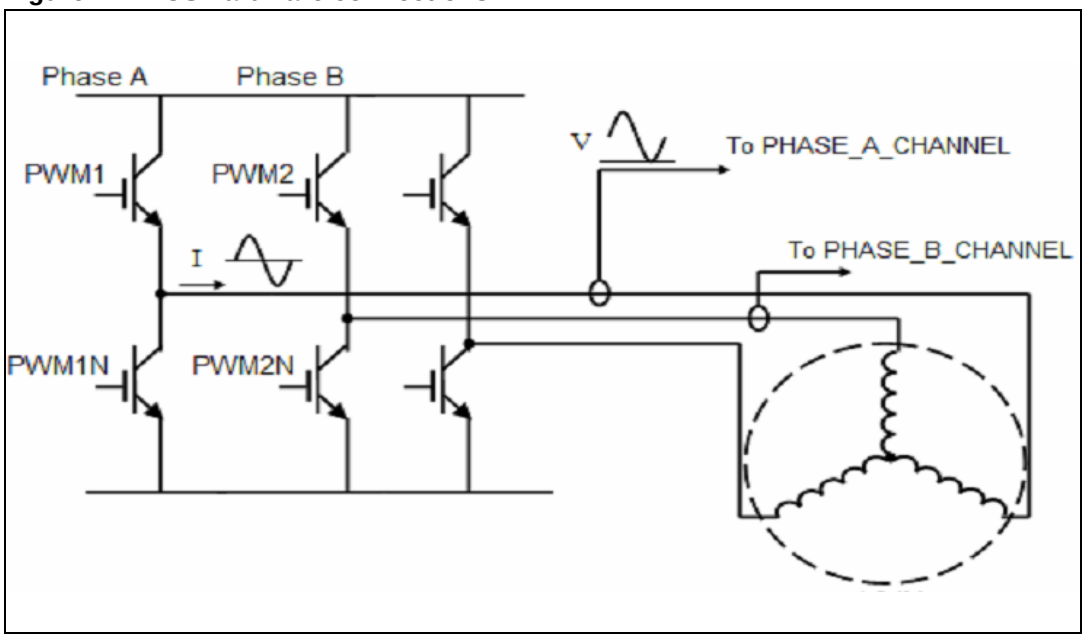
In order for the implemented PMSM FOC algorithm to work properly, it is necessary to ensure that the software implementation of the `75x_svpwm_ICS` module and the hardware connections of the two ICS are consistent.

As illustrated in *Figure 12*, the two ICS must act as transducers on motor phase currents coming out of the inverter legs driven by STR750 PWM signals PWM1 (Phase A) and PWM2 (Phase B). In particular, the current coming out of inverter Phase A must be read by an ICS whose output has to be sent to the analog channel specified by the PHASE_A_CHANNEL parameter in `MC_pwm_ics_prm.h`. Likewise, the current coming out

of inverter Phase B must be read by the other ICS and its output has to be sent to the analog channel specified by the PHASE_B_CHANNEL parameter in `MC_pwm_ics_prm.h`.

About the positive current direction convention, a positive half-wave on PHASE_X_CHANNEL is expected, corresponding to a positive half-wave on the current coming out of the related inverter leg (see direction of I in *Figure 12*).

Figure 12. ICS hardware connections



3.3.2 Selecting PHASE_A_CHANNEL and PHASE_B_CHANNEL

Default settings for PHASE_A_CHANNEL and PHASE_B_CHANNEL are respectively `ADC_CHANNEL11` and `ADC_CHANNEL10`. You can change the default settings if the hardware requires it by editing the `MC_pwm_ics_prm.h` file. However, there are a few rules to follow when selecting the new ADC channels:

- You must initialize the proper GPIOs as analog inputs; an example for channel 8 is given below:

```
/* ADC Channel 8 pin configuration */
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_29;
GPIO_Init(GPIO0, &GPIO_InitStructure);
```

- You must select two contiguous channels (for example, `ADC_CHANNEL8` and `ADC_CHANNEL9`) and the one with the highest number must be associated with PHASE_A_CHANNEL (for example, PHASE_A_CHANNEL -> `ADC_CHANNEL9`, PHASE_B_CHANNEL->`ADC_CHANNEL8`)

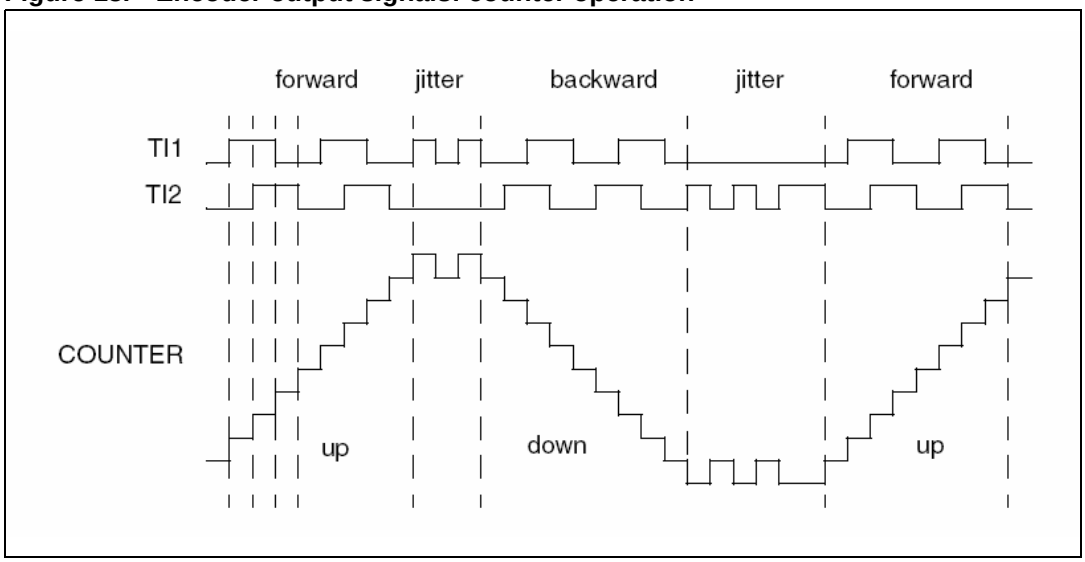
3.4 Managing the incremental encoder

Quadrature incremental encoders are widely used to read the rotor position of electric machines.

As the name implies, incremental encoders actually read angular displacements with respect to an initial position: if that position is known, then rotor absolute angle is known too.

Quadrature encoders have two output signals (represented in *Figure 13* as TI1 and TI2). With these, and with the STR750 standard timer in encoder interface mode, it is possible to get information about rolling direction.

Figure 13. Encoder output signals: counter operation



In addition, rotor angular velocity can be easily calculated as a time derivative of angular position.

To set up the PMSM software library for use with an incremental encoder, simply modify the MC_encoder_param.h header files according to the indications given in *Section 2.2.3 on page 18*.

However, some extra care should be taken, concerning what is considered to be the positive rolling direction. Because of this, and because of how the encoder output signals are wired to the microcontroller input pins, it is possible to have a sign discrepancy between the real rolling direction and the direction that is read. To avoid this kind of reading error, you can apply the following procedure:

1. You can correct it by simply swapping and rewiring the encoder output signals.
2. If this isn't practical, you can modify a software setting instead: in the `75x_encoder.c` file, replace the code line:

   ```
   TIM_InitStructure.TIM_IC1Polarity = TIM_IC1Polarity_Rising;
   ```

   with:

   ```
   TIM_InitStructure.TIM_IC1Polarity = TIM_IC1Polarity_Falling;
   ```

3.5 Fault messages

This section provides a list of possible fault message that can be displayed on the LCD when using the software library together with the STR750MC-KIT:

- **“Over Current”**

  An Emergency Stop was detected on the PWM peripheral dedicated pin. If using STR750-MCKIT it could mean that either the hardware over temperature protection or the hardware over current protection were triggered. Refer to the STR750-MCKIT User Manual for details,

- **“Over Heating”**

  An over temperature was detected on the dedicated analog channel; the digital threshold `NTC_THRESHOLD` and the relative hysteresis (`NTC_HYSTERESIS`) are specified in the `MC_Misc.c` source file. Refer to the STR750-MCKIT User Manual for details.

- **“Bus Over Voltage”**

  An over voltage was detected on the dedicated analog channel. The digital threshold (`OVERVOLTAGE_THRESHOLD`) is specified in the `MC_Misc.c` source file. Refer to the STR750-MCKIT User Manual for details.

- **"Bus Under Voltage"**

  The bus voltage is below 20V DC. The threshold is specified in the `MC_Misc.c` source file (`UNDERVOLTAGE_THRESHOLD` parameter). The corresponding FAULT flag is not cleared by firmware, therefore the STR750 must be reset after the bus voltage has been switched on.

3.6 Note on debugging tools

The third party JTAG interface should always be isolated from the application using the MB535 JTAG opto-isolation board; it provides protection for both the JTAG interface and the PC connected to it.

Caution: During a breakpoint, when using the JTAG interface for the firmware development, the motor control cell clock circuitry should always be enabled; if disabled, a permanent DC current may flow in the motor because the PWM outputs are enabled, which could cause permanent damage to the power stage and/or motor. A dedicated bit in the `PWM_CR` control register, the DBGC bit must be set to 1 (see *Figure 14*).

Figure 14. DBGC bit in PWM control register (from STR750 reference manual)

Control Register (PWM_CR)

Address Offset: 00h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DBASE					DBGC	MMS		res.	CMS		U/D	OPM	CNT_EN	CNT_RST	UFS
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bit 10	**DBGC**: *Debug Control* See *Section 9.5.3: Debug mode on page 186* for a detailed description. 0: The timer is stopped in debug mode 1: The timer continues working in debug mode

4 Motor control library routines

4.1 Library reference

Functions are described in the format given below:

Synopsis	Lists the referenced include files and prototype declarations.
Description	Describes the functions specifically with a brief explanation of how they are executed.
Input	Gives the format and units.
Returns	Gives the value returned by the function, including when an input value is out of range or an error code is returned.
Note	Indicates the limits of the function or specific requirements that must be taken into account before implementation.
Caution	Indicates important points that must be taken into account to prevent hardware failures.
Functions called	Lists called functions. Useful to prevent conflicts due to the simultaneous use of resources.
Code example	Indicates the proper way to use the function, and if there are certain prerequisites (interrupt enabled, etc.).

Some of these sections may not be included if not applicable for example, no parameters or obvious use.

4.2 Motor control software layer

The software related to motor control is part of several modules. These modules provide:

- Basic setup
- Control routines
- Related interrupt handling routines
- Current sampling for torque and flux regulation
- Speed acquisition for closed loop operation

4.2.1 `75x_svpwm_3shunt` module

Two important tasks are performed in the `75x_svpwm_3shunt` module:

- Space vector pulse width modulation (SVPWM)
- Three shunt resistor topology current reading

In order to reconstruct the currents flowing through a three-phase load with the required accuracy using three shunt resistors, it is necessary to properly synchronize A/D conversions with the generated PWM signals.

SVPWM_3ShuntInit

Synopsis	void SVPWM_3ShuntInit(void);
Description	The purpose of this function is to set-up microcontroller peripherals for performing 3 shunt resistors topology current reading and center aligned PWM generation. The function initializes the DMA, EIC, ADC, GPIO, PWM, and TIM0 peripherals. In particular, the DMA, ADC, PWM and TIM0 peripherals are configured to perform two synchronized A/D conversions per PWM switching period.
Input	None
Returns	None
Caution	It must be called at main level
Functions called	**Standard Library:** MRCC_PeripheralClockConfig, GPIO_Init, EIC_IRQInit, EIC_IRQCmd, DMA_Init, DMA_Cmd, TIM_DMAConfig, DMA_DeInit, ADC_DMACmd, PWM_DeInit, PWM_StructInit, PWM_Init, PWM_TRGOSelection, PWM_ClearFlag, PWM_ITConfig, PWM_ResetCounter, ADC_StructInit, ADC_Init, ADC_Cmd, ADC_StartCalibration, ADC_ConversionCmd, TIM_Init, TIM_SynchroConfig, TIM_ResetCounter, PWM_Cmd. **Motor Control Library:** SVPWM_3ShuntCurrentReadingCalibration

SVPWM_3ShuntCurrentReadingCalibration

Synopsis	void SVPWM_3ShuntCurrentReadingCalibration(void);
Description	The purpose of this function is to store the three analog voltages corresponding to zero current values for compensating the offset introduced by the amplification network.
Input	None
Returns	None
Caution	This function must be called before PWM outputs are enabled so that the current flowing through inverter legs is zero. When using STR750 MC Kit, the power board (MB459B) must be supplied before the control board (MB469B). This way, the current sensing conditioning network will reach steady state before performing calibration.
Functions called	**Standard Library:** ADC_GetFlagStatus, ADC_ConversionCmd, ADC_Init, ADC_ClearFlag, ADC_ITConfig **Motor Control Library:** SVPWM_3ShuntCalcDutyCycles

SVPWM_3ShuntGetPhaseCurrentValues

Synopsis	Curr_Components SVPWM_3ShuntGetPhaseCurrentValues(void);
Description	This function computes current values of Phase A and Phase B in q15 format starting from values acquired from the A/D Converter peripheral.
Input	None
Returns	Curr_Components type variable
Caution	In order to have a q15 format for the current values, the digital value corresponding to the offset must be subtracted. Thus, it must be called after SVPWM_3ShuntCurrentReadingCalibration
Functions called	None

SVPWM_3ShuntCalcDutyCycles

Synopsis	void SVPWM_3ShuntCalcDutyCycles (Volt_Components Stat_Volt_Input);
Description	After execution of the PMSM FOC algorithm, new stator voltages component V_α and V_β are computed. The purpose of this function is to calculate exactly the three duty cycles to be applied to motor phases starting from the value of those voltage components. Moreover, once the three duty cycles to be applied in next PWM period are known, this function sets the DMA, ADC and TIM0 peripherals for the next current reading. In particular, depending on the duty cycle values, the delay for the two current samplings are computed (see *Section 4.2.4 on page 39*). Refer to *Section 4.2.2* for information on the theoretical approach of SVPWM.
Input	V_α and V_β
Returns	None
Caution	None
Functions called	None

SVPWM_3ShuntGPADCConfig

Synopsis	void SVPWM_3ShuntGPADCConfig(void);
Description	The purpose of this function is to configure the A/D converter for general purpose conversions after conversions for current reading have been performed. In particular, this function starts a chain of regular conversions whose first channel is GP_CONVERSIONS_FIRST_CHANNEL (defined in 'MC_pwm_3shunt_prm.h'). In addition, the number of channels to be converted is set equal to GP_CONVERSIONS_NUMBER (defined in 'MC_pwm_3shunt_prm.h').
Input	None
Returns	None
Caution	As mentioned in *Section 4.2.2*, the overall duration of the regular chain conversion must be lower than the duration of the PMSM FOC routines (Clarke, Park, Reverse Park and SVPWM generation).
Functions called	None

4.2.2 Space vector PWM implementation

Figure 15 shows the stator voltage components V_α and V_β while *Figure 16* illustrates the corresponding PWM for each of the six space vector sectors.

Figure 15. V_α and V_β stator voltage components

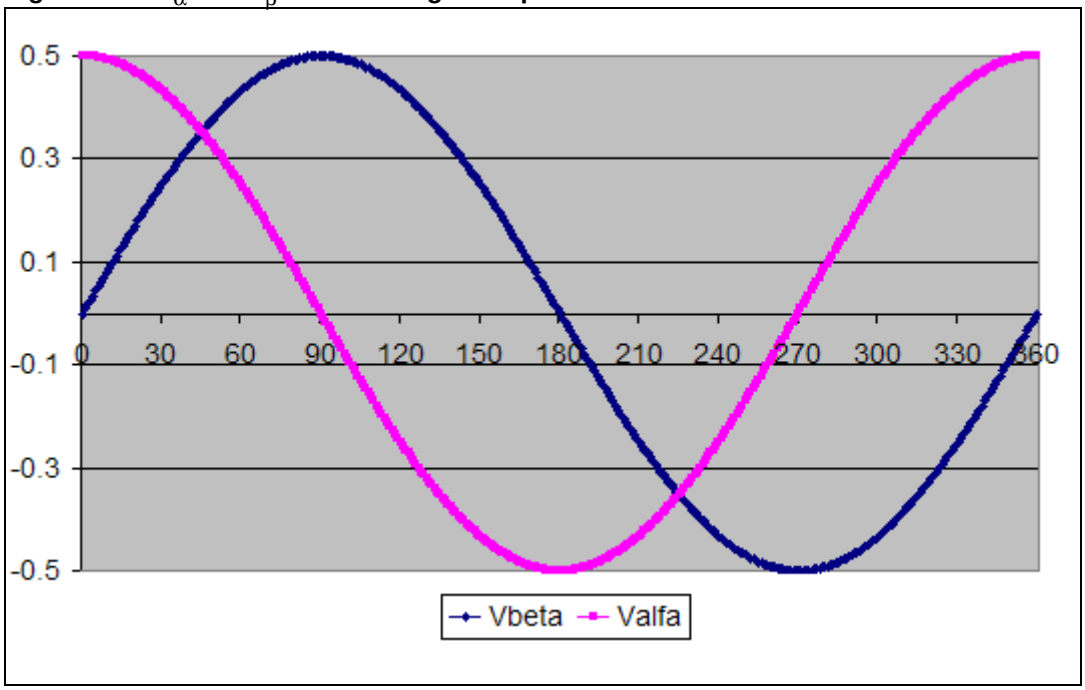
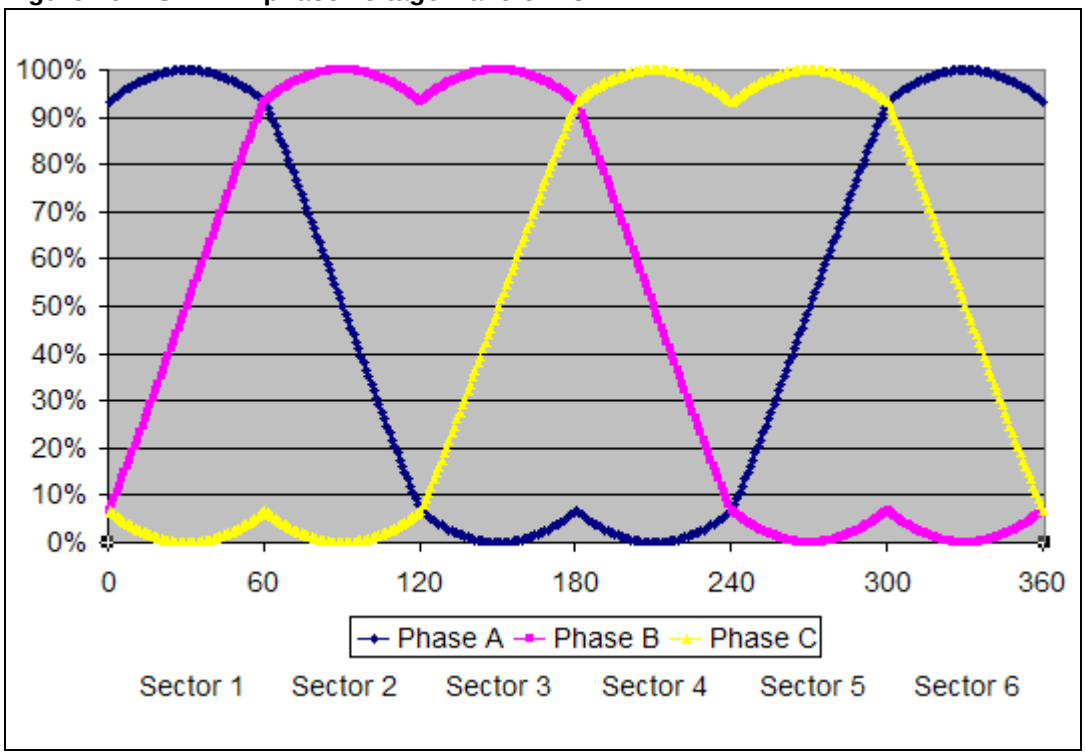


Figure 16. SVPWM phase voltage waveforms



With the following definitions for:

$$U_{\alpha} = \sqrt{3} * T * V_{alfa}$$

$$U_{\beta} = T * V_{beta}$$

and

$$X = U_{\beta}$$

$$Y = \frac{U_{\alpha} + U_{\beta}}{2}$$

$$Z = \frac{U_{\beta} - U_{\alpha}}{2}$$

literature demonstrates that the space vector sector is identified by the conditions shown in *Table 2*.

Table 2. Sector identification

	Y<0			Y>=0		
	Z<0	Z>=0		Z<0		Z>=0
		X<=0	X<0	X<=0	X>0	
Sector	**V**	**IV**	**III**	**VI**	**I**	**II**

The duration of the positive pulse widths for the PWM applied on Phase A, B and C are respectively computed by the following relationships:

Sector I, IV:

$$t_A = \frac{T + X - Z}{2}$$
$$t_B = t_A + Z$$
$$t_C = t_B - X$$

Sector II, V:

$$t_A = \frac{T + Y - Z}{2}$$
$$t_B = t_A + Z$$
$$t_C = t_A - Y$$

Sector III, VI:

$$t_A = \frac{T - X + Y}{2}$$
$$t_B = t_C + X$$
$$t_C = t_A - Y$$

Where T is the PWM period.

Now, considering that the PWM pattern is center aligned and that the phase voltages must be centered to 50% of duty cycle, it follows that the values to be loaded into the PWM output compare registers are given respectively by:

Sector I, IV:

$$TimePhA = \frac{T}{4} + \frac{T/2 + X - Z}{2}$$
$$TimePhB = TimePhA + Z$$
$$TimePhC = TimePhB - X$$

Sector II, V:

$$TimePhA = \frac{T}{4} + \frac{T/2 + Y - Z}{2}$$
$$TimePhB = TimePhA + Z$$
$$TimePhC = TimePhA - Y$$

Sector III, VI:

$$TimePhA = \frac{T}{4} + \frac{T/2 - X + Y}{2}$$
$$hTimePhB = TimePhC + X$$
$$TimePhC = TimePhA - Y$$

4.2.3 Current sampling in three shunt topology and general purpose A/D conversions

The three currents I_1, I_2, and I_3 flowing through a three-phase system follow the mathematical relation:

$I_1+I_2+I_3=0$

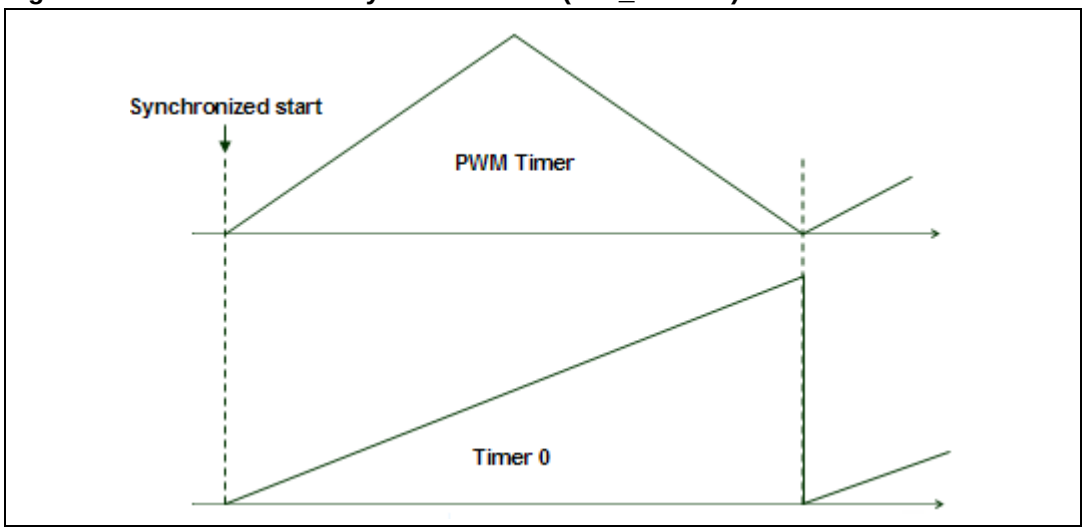
For this reason, to reconstruct the currents flowing through a generic three-phase load, it is sufficient to sample only two out of the three currents while the third one can be computed by using the above relation.

The flexibility of the STR750 A/D converter trigger, makes it possible to synchronize the two A/D conversions needed for reconstructing the current flowing through the three-phase AC induction motor at any given time along the PWM period. To do this, the control algorithm must have a full control of the A/D converter peripheral.

Furthermore, you have the possibility to add any A/D conversions required for your application (hereafter referred to as general purpose conversions). This section describes how this is achieved.

First of all, the `SVPWM_3ShuntInit` function performs the synchronization between PWM and TIM0 peripherals (*Figure 17* shows the two peripheral counters when `REP_RATE = 1`), then, the A/D converter peripheral is configured so that it is triggered by the TIM0 OC2 signal.

Figure 17. PWM and TIM0 synchronization (`REP_RATE=1`)



This way, when the value of the TIM0 counter matches the value contained in the OCR2 register, the first A/D conversion for current sampling is started.

Meanwhile, a DMA transaction reloads the TIM0 OCR2 register with the value corresponding to the delay required for the second current sampling conversion. Moreover, the end of this first A/D conversion triggers another DMA transaction which sets the next channel to be converted in the ADC register CLR2.

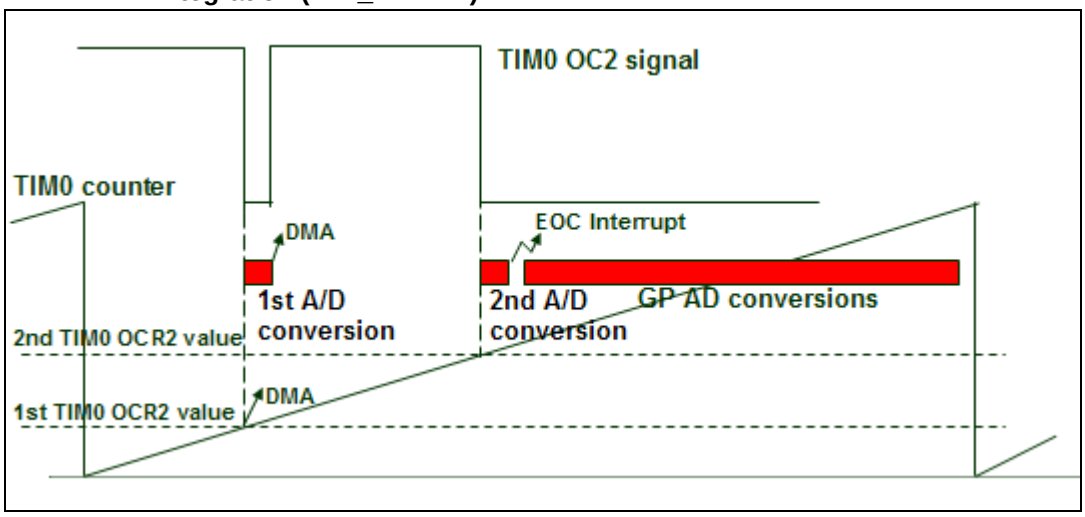
At the end of the second conversion, the three-phase load current has been updated and the PMSM FOC algorithm can then be executed in the A/D End of Conversion Interrupt Service Routine (EOC ISR). In this routine, the A/D converter is also reconfigured so that it can perform the general purpose chain of conversions while the CPU executes the PMSM FOC algorithm.

The entire process is illustrated in *Figure 18*.

After execution of the PMSM FOC algorithm, the A/D converter is configured to perform the next PWM period three-phase current sensing (delays and channels). This allows to reduce the CPU load (lower number of ADC ISR).

To specify the general purpose conversions to be performed, you can select the first channel and the number of channels to be converted by editing the `GP_CONVERSIONS_FIRST_CHANNEL` and `GP_CONVERSIONS_NUMBER` parameters respectively in the `MC_pwm_3shunt_prm.h` header file.

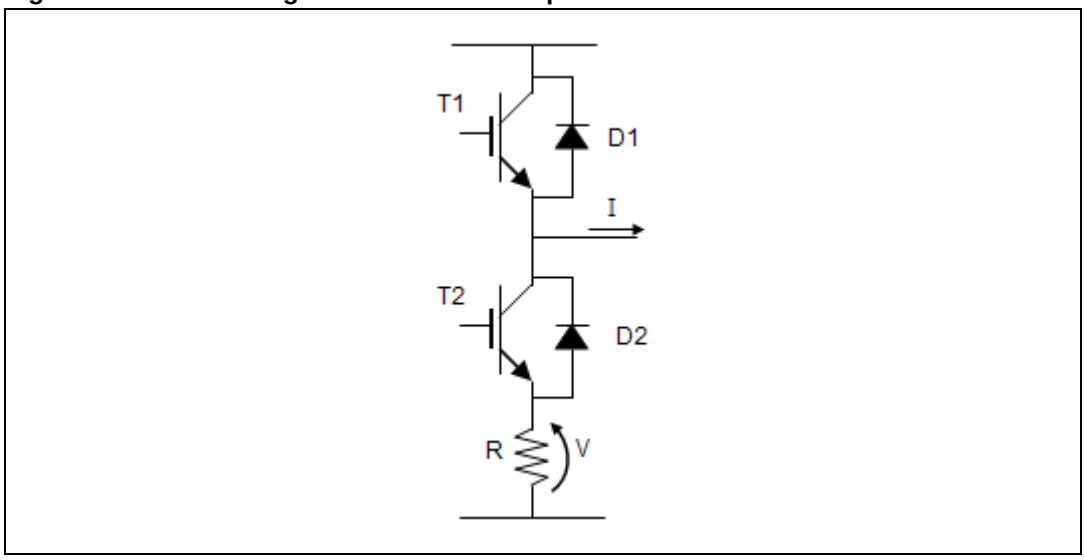
Figure 18. Three shunt topology current sampling and GP A/D conversions integration (`REP_RATE=1`)



4.2.4 Tuning delay parameters and sampling stator currents in three-shunt resistor topology

Figure 19 shows one of the three inverter legs with the related shunt resistor.

Figure 19. Inverter leg and shunt resistor position



To indirectly measure the phase current I, it is possible to read the voltage V providing that the current flows through the shunt resistor R.

It is possible to demonstrate that, whatever the direction of current I, it always flows through the resistor R if transistor T2 is switched on and T1 is switched off. This implies that in order to properly reconstruct the current flowing through one of the inverter legs, it is necessary to properly synchronize the conversion start with the generated PWM signals. This also means that current reading cannot be performed on a phase where the duty cycle applied to the low side transistor is either null or very short.

Fortunately, as discussed in *Section 4.2.3*,to reconstruct the currents flowing through a generic three-phase load, it is sufficient to simultaneously sample only two out of three currents, the third one being computed from the relation given in *Section 4.2.3*. Thus, depending on the space vector sector, the A/D conversion of voltage V will be performed only on the two phases where the duty cycles applied to the low side switches are the highest. In particular, by looking at *Figure 16*, you can deduct that in sectors 1 and 6, the voltage on the Phase A shunt resistor can be discarded; likewise, in sectors 2 and 3 for Phase B, and finally in sectors 4 and 5 for Phase C.

Moreover, in order to properly synchronize the two stator current reading A/D conversions, it is necessary to distinguish between the different situations that can occur depending on PWM frequency and applied duty cycles.

Note: The explanations below refer to space vector sector 1. They can be applied in the same manner to the other sectors.

Case 1: Duty cycle applied to Phase A low side switch is larger than $DT+T_N+ 2T_S + T_H + T_{DMA}$

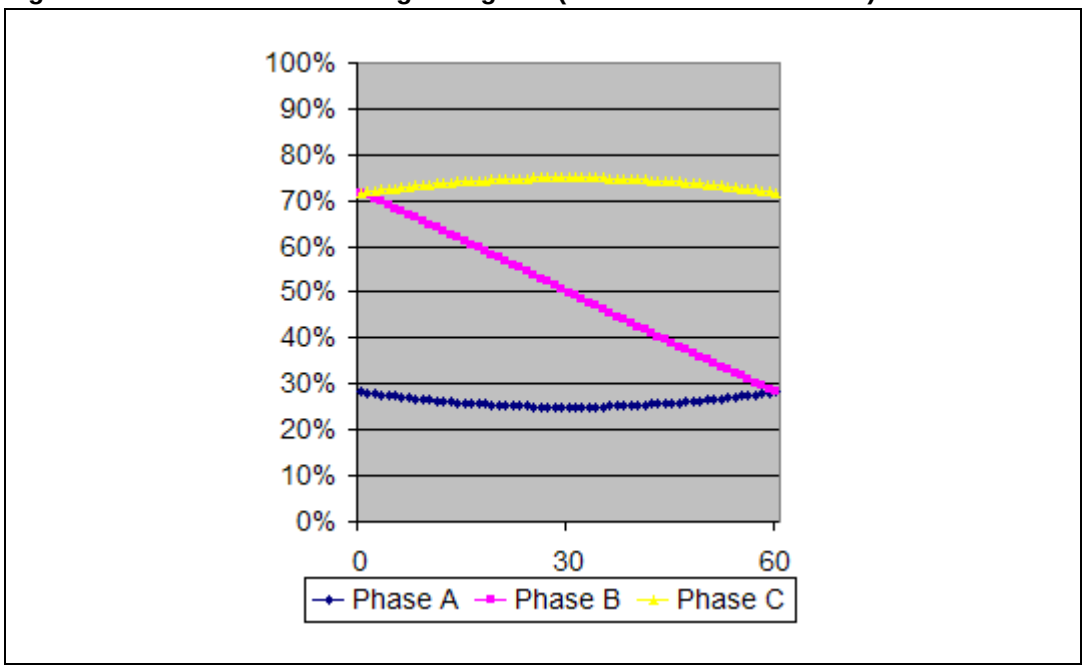
Where:

- DT is dead time.
- T_N is the duration of the noise induced on the shunt resistor voltage of a phase by the commutation of a switch belonging to another phase.
- T_S is the sampling time of the STR750 A/D converter. Refer to the STR750 reference manual for more detailed information.
- T_H is the holding time of the STR750 A/D converter. Refer to the STR750 reference manual for more detailed information.
- T_{DMA} is the time required for the DMA to load the value related to the next conversion delay in TIM0 OCR2 (refer to *Section 4.2.3: Current sampling in three shunt topology and general purpose A/D conversions on page 37* for further details).

This case typically occurs when SVPWM with low (<60%) modulation index is generated (see *Figure 20*). The modulation index is the applied phase voltage magnitude expressed as a percentage of the maximum applicable phase voltage (the duty cycle ranges from 0% to 100%).

Figure 21 offers a reconstruction of the PWM signals applied to low side switches of Phase A and B in these conditions plus a view of the analog voltages measured on the STR750 A/D converter pins for both Phase B and C (the time base is lower than the PWM period).

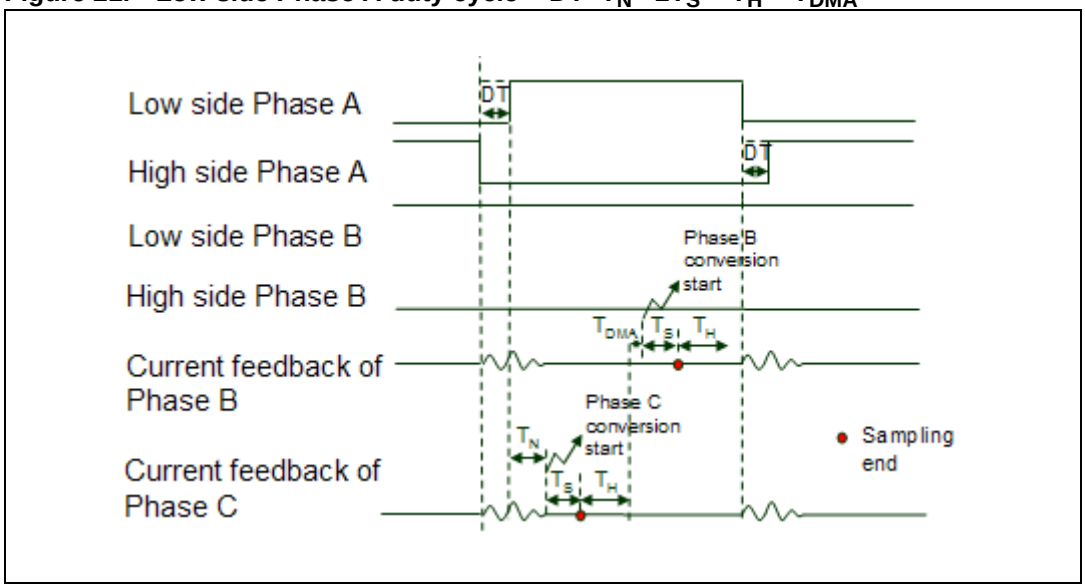
Figure 20. Low side switches gate signals (low modulation indexes)



Note that these current feedbacks are constant in the view in *Figure 21* because it is assumed that commutations on Phase B and C have occurred out of the visualized time window.

Moreover, it can be observed that in this case the two stator current sampling conversions can be performed between the two commutations of the Phase A low side switch, as shown in *Figure 21*.

Figure 21. Low side Phase A duty cycle > DT+T_N+ 2T_S + T_H + T_{DMA}



After the commutation of the Phase A low side switch, a blanking window equal to T_N is applied before starting conversion of phase C, then at the end of the first conversion, it is necessary to wait a T_{DMA} period before starting the phase B conversion.

Case 2: DT+TN+TS < Phase A duty cycle < DT+TN+ 2TS + TH + TDMA

In this case, only one of the two conversions can be performed between the two Phase A low side commutations. The other conversion is then synchronized depending on the difference of duty cycles between Phase B and A ($\Delta Duty_{A-B}$). In particular if $\Delta Duty_{A-B} < DT+T_N+T_S$ (as shown in the red circle in *Figure 22*), the sampling of Phase C cannot be performed between Phase B low side switching on and Phase A high side switching off (see *Figure 23*). Therefore, Phase C current sampling is performed before Phase B high side commutation.

Figure 22. $DT+T_N+T_S$< Low side Phase A duty cycle < $DT+T_N+2T_S+T_H+T_{DMA}$ and $\Delta Duty_{A-B}<DT+T_N+T_S$

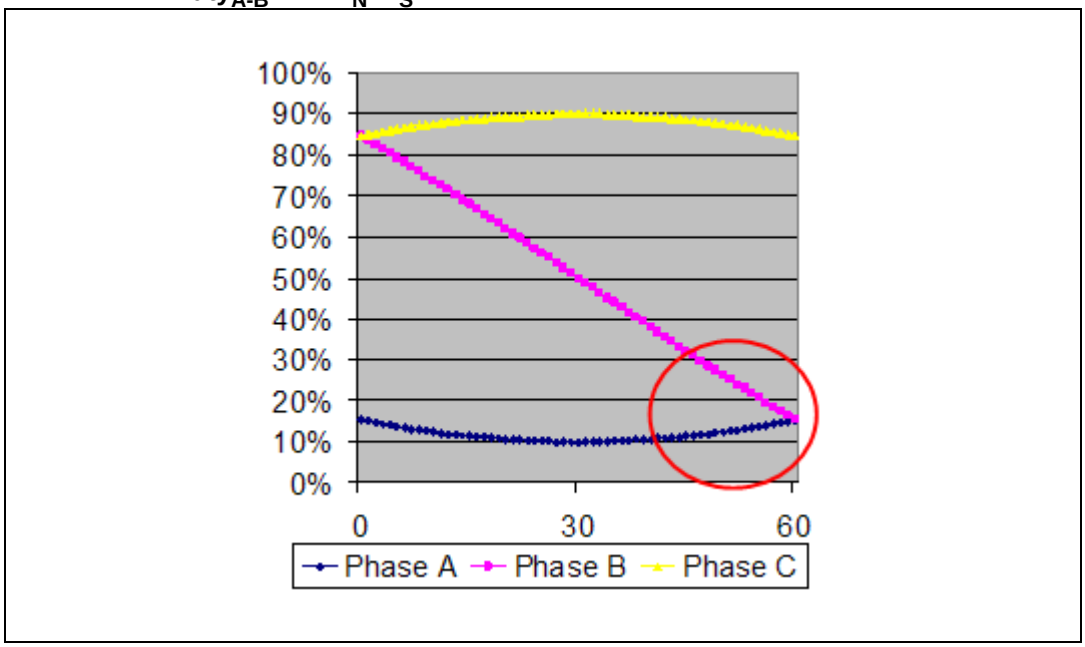
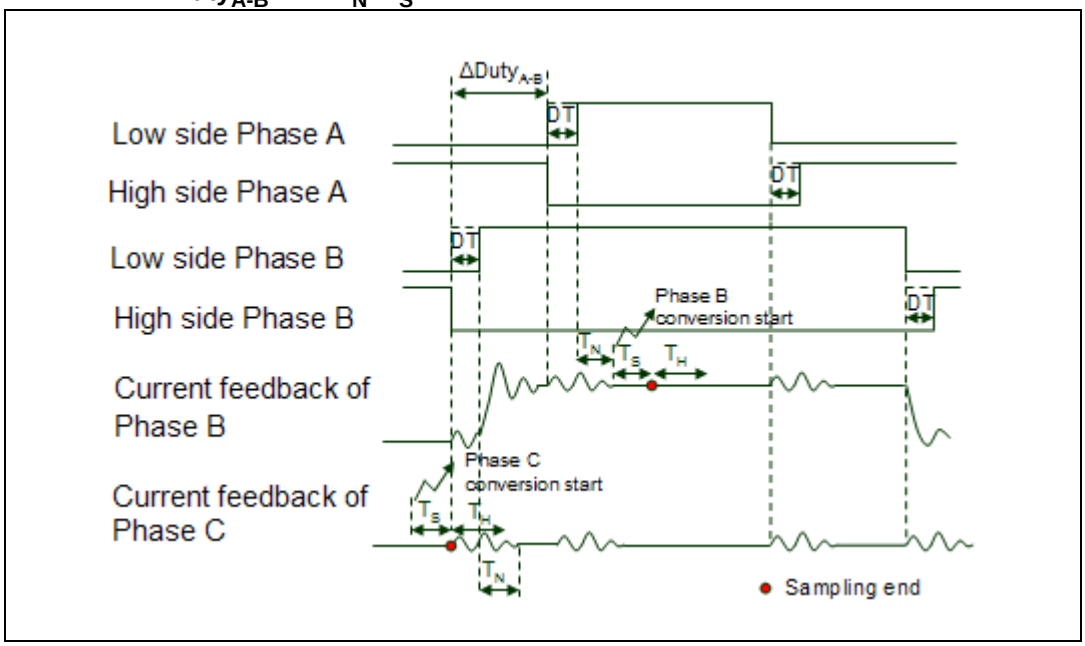


Figure 23. $DT+T_N+T_S$ < Low side Phase A duty cycle < $DT+T_N+2T_S+T_H+T_{DMA}$ and $\Delta Duty_{A-B}<DT+T_N+T_S$



On the contrary, if $\Delta Duty_{A-B} > DT+T_N+T_S$ (as shown in the red circle in *Figure 24*), Phase C conversion is performed between Phase B low side switch on and Phase A high side switch off (see *Figure 25*).

Figure 24. $DT+T_N+T_S$ < Low side Phase A duty cycle < $DT+T_N+2T_S+T_H+T_{DMA}$ and $\Delta Duty_{A-B}>DT+T_N+T_S$

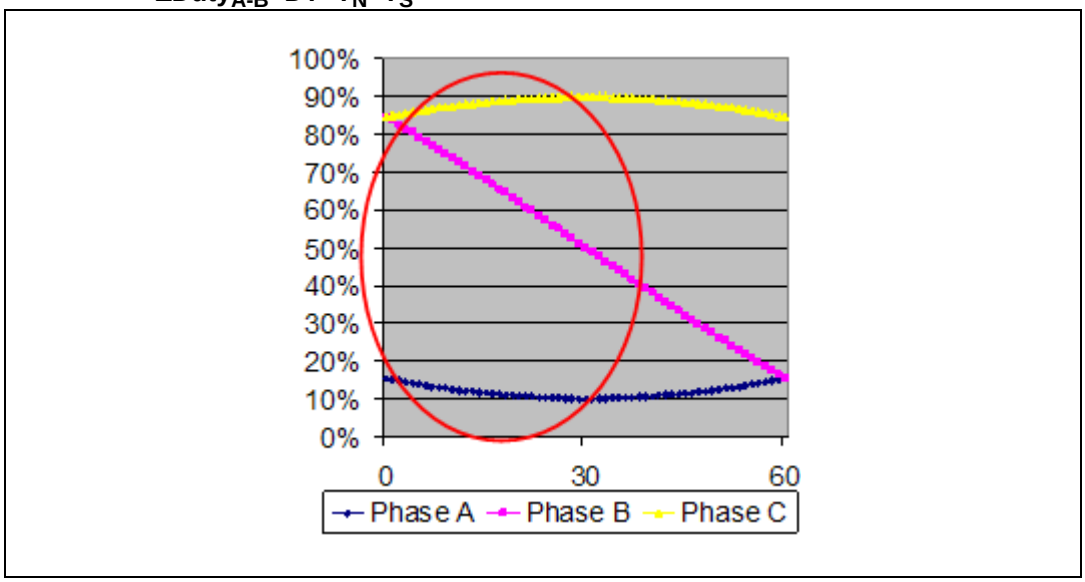
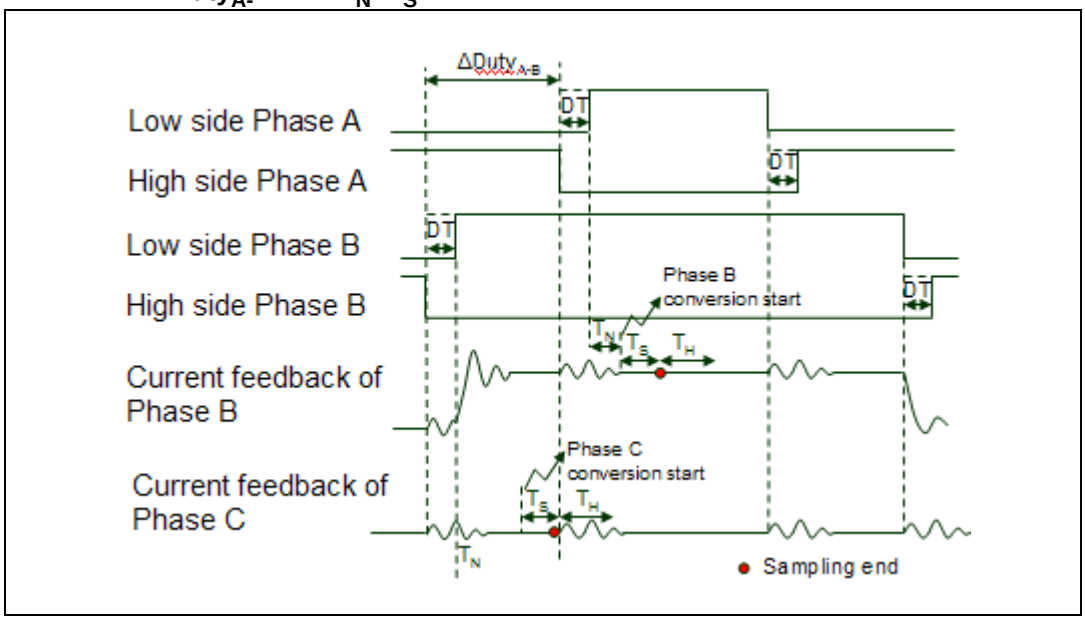


Figure 25. $DT+T_N+T_S$ < Low side Phase A duty cycle < $DT+T_N+2T_S+T_H+T_{DMA}$ and $\Delta Duty_{A-B} > DT+T_N+T_S$

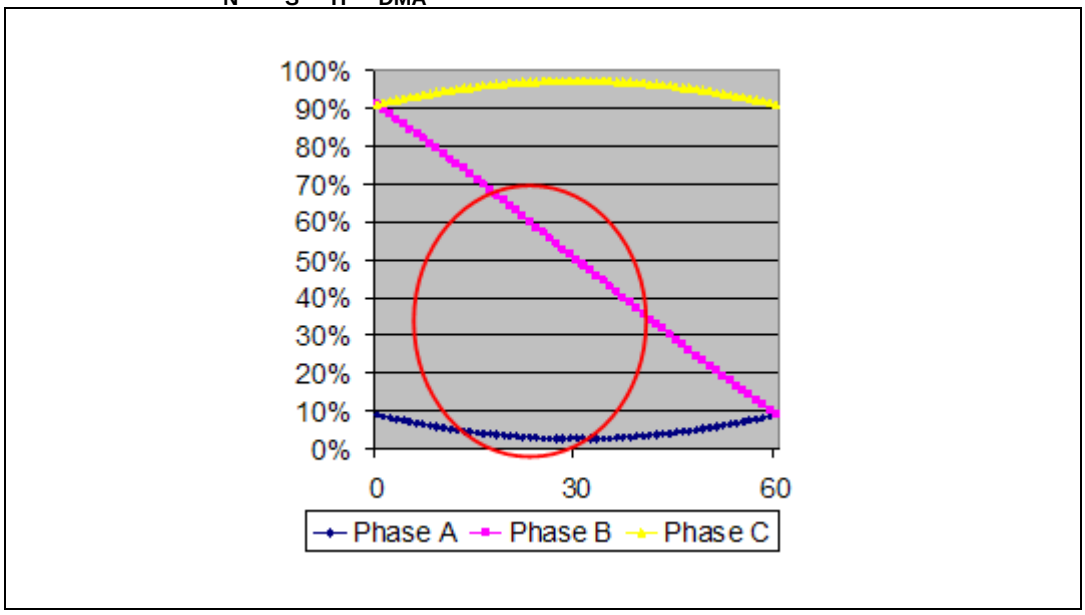


Case 3: Phase A pulse width < DT+TN+TS

In this case, the duty cycle applied to Phase A is so short that no current sampling can be performed in between the two low side commutations.

Then if the difference of duty cycles between Phase B and A is long enough to allow two A/D conversions to be performed between Phase B low side switch on and Phase A high side switch off, the strategy shown in *Figure 27* is used.

Figure 26. Low side duty cycle Phase A < $DT+T_N+T_S$ and $\Delta Duty_{A-B}$ > $DT+T_N+2T_S+T_H+T_{DMA}$



Otherwise, if the difference of duty cycles between Phase B and A is long enough to allow only one A/D conversion to be performed between Phase B low side switch on and Phase A high side switch off, the strategy shown in *Figure 29* is used.

In *Figure 29*, T_{Rise} represents the time required by the analog voltage on the shunt resistor of a phase (signal 'Current feedback of Phase B') to settle after a commutation of the low side switch belonging to the same phase.

Figure 27. Low side duty cycle Phase A < $DT+T_N+T_S$ and $\Delta Duty_{A-B} > DT+T_N+2T_S+T_H+T_{DMA}$

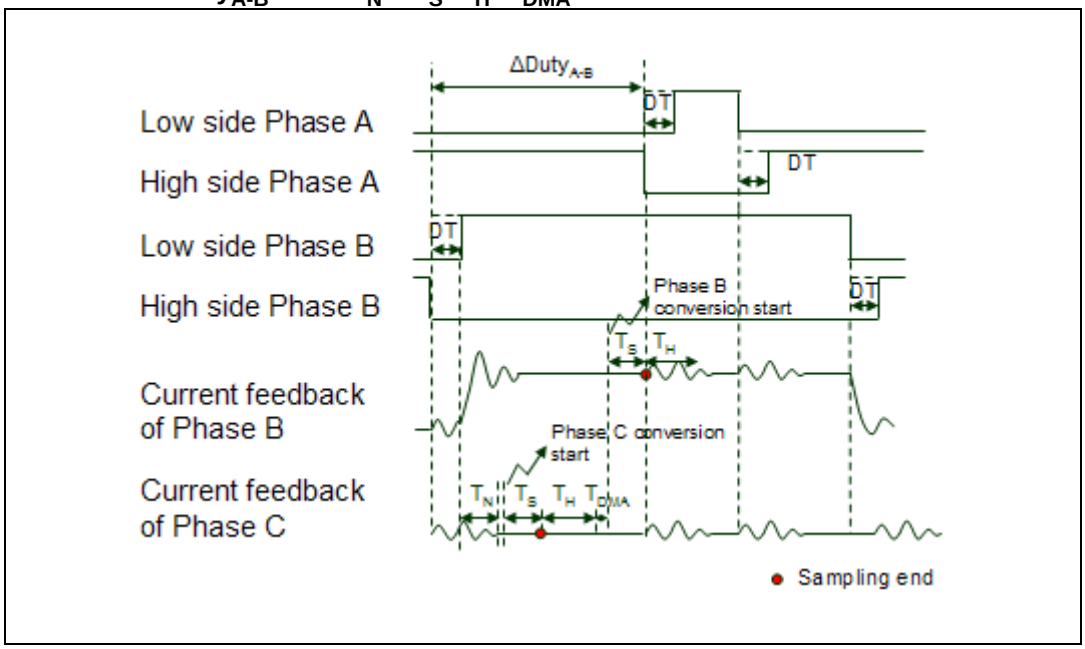


Figure 28. Figure 31: Low side duty cycle Phase A < $DT+T_N+T_S$ and $DT+T_{Rise}+T_S < \Delta Duty_{A-B} < DT+T_N+2T_S+T_H+T_{DMA}$

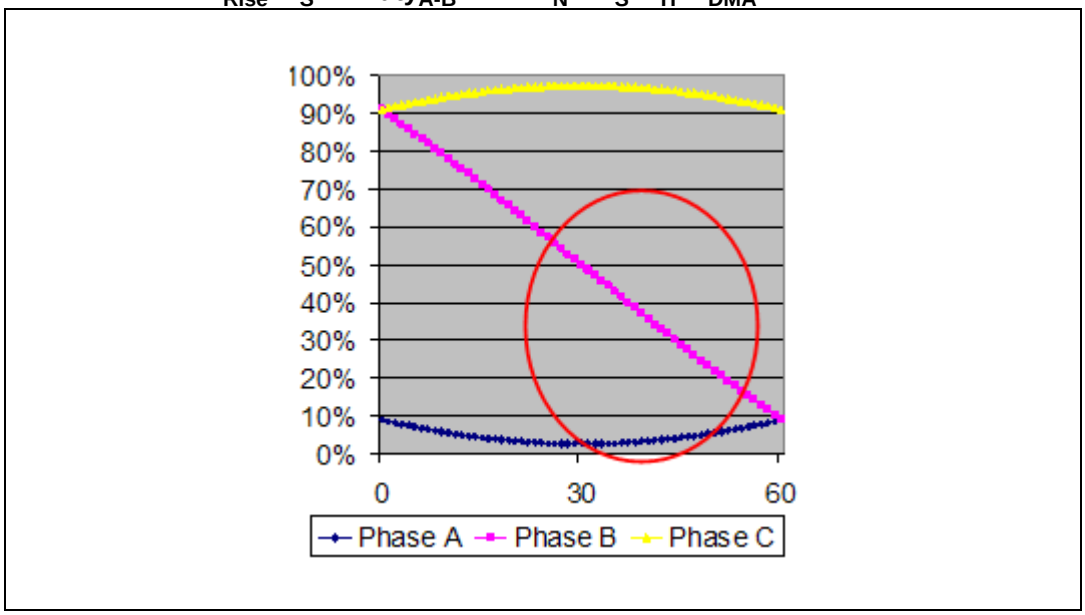
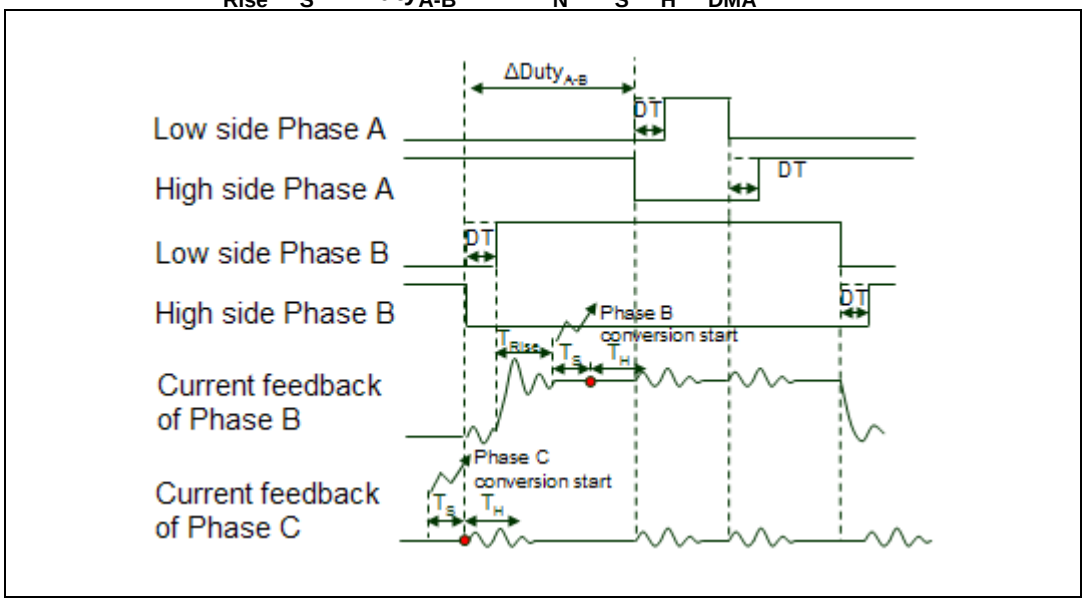


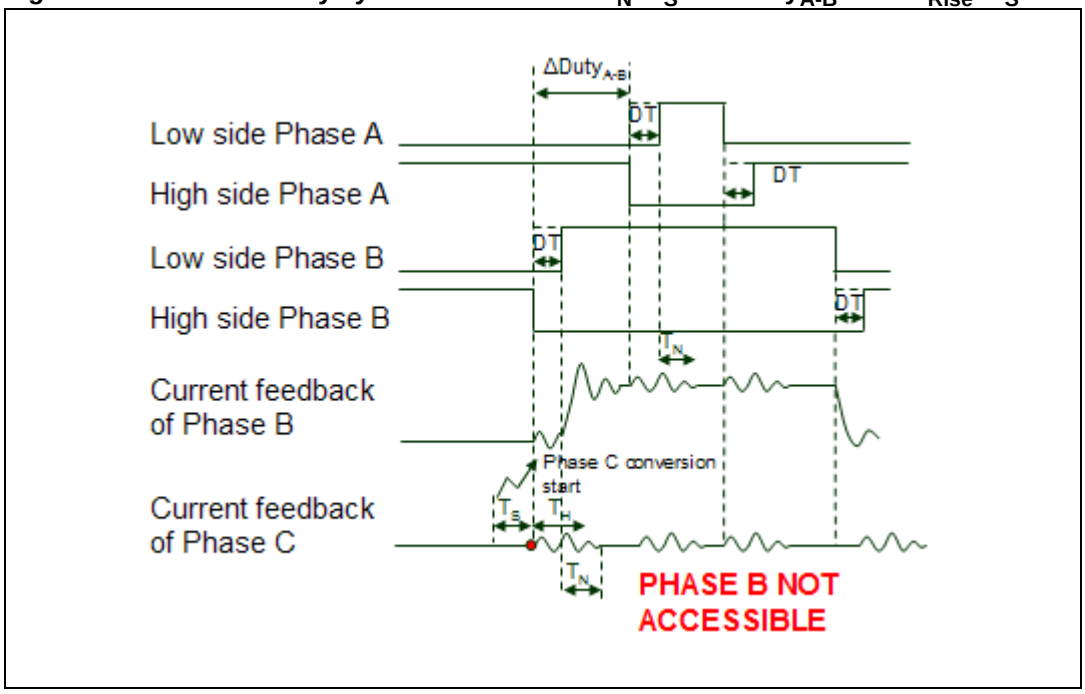
Figure 29. Low side duty cycle Phase A < $DT+T_N+T_S$ and $DT+T_{Rise}+T_S < \Delta Duty_{A-B} < DT+T_N+2T_S+T_H+T_{DMA}$



Finally, when a high modulation index (> 92%) and high frequency (>11kHz) PWM signal is generated, it could happen that both Phase A pulse width is lower than $DT+T_N+T_S$ and that $\Delta Duty_{A-B} < DT+T_{Rise}+T_S$. In this case, it is not possible to perform the current reading on Phase B, (see *Figure 30*), so the PWM patterns are slightly modified to relapse in the case shown in *Figure 29*. Because this PWM pattern modification produces a distortion on the phase currents, it is better to limit the scope of the modification by limiting the modulation index depending on the selected PWM frequency.

- DT = 0.7µs
- T_N = 2.55µs
- T_S = 1.6µs
- T_H = 2.67µs
- T_{DMA} = 0.7µs
- T_{Rise} =2.6µs

Figure 30. Low side duty cycle Phase A < DT+T_N+T_S and $Duty_{A-B}$< DT+T_{Rise}+T_S



The maximum applicable duty cycles are listed in *Table 3* as a function of the PWM frequency.

Table 3. PWM frequency vs maximum duty cycle

PWM frequency	Max duty cycle	Max modulation index (MMI)
Up to 11.4kHz	100%	100%
12.2kHz	99.5%	99%
12.9kHz	99%	98%
13.7kHz	98.5%	97%
14.4kHz	98%	96%
15.2kHz	97.5%	95%
16kHz	97%	94%
16.7kHz	96.5%	93%
17.5kHz	96%	92%

Note: The figures above were measured using the MB459 motor control board. This evaluation platform is designed to support several motor driving topologies (PMSM and AC induction) and current reading strategies (single and three-shunt resistors). Therefore, the figures provided in Table 3 should be understood as a starting point and not as a best case.

You can further increase the maximum applicable duty when using your own hardware system by editing the following definitions in the MC_pwm_3shunt_prm.h header file:

```
#define HOLD_TIME 0xA0 //2.67usec 1/60MHz units
#define DMA_TIME  0x2A  //0.7usec
#define SAMPLING_TIME 0x60//1.6usec
#define TNOISE 0x96//2.55usec
#define TRISE 0x96 //2.6usec
```

4.2.5 75x_svpwm_ICS module

Two important tasks are performed in the 75x_svpwm_ICS module:

- Space vector pulse width modulation (SVPWM),
- Three-phase current reading when two isolated current sensors (ICS) are used.

In order to reconstruct the currents flowing through a three phase load with the required accuracy using two ICS', it is necessary to properly synchronize A/D conversions with the generated PWM signals.

Two tasks are included in a single software module.

SVPWM_IcsInit

Synopsis	void SVPWM_IcsInit(void);
Description	The purpose of this function is to set-up microcontroller peripherals for performing ICS reading and center aligned PWM generation. The function initializes EIC, ADC, GPIO, and PWM peripherals. In particular ADC and PWM peripherals are configured to perform one injected chain of two A/D conversions every time PWM registers are updated (event called U event). Refer to *Section 4.2.6* for further information on A/D conversion triggering in ICS configuration.
Input	None
Returns	None
Note	It must be called at main level
Functions called	**Standard Library:** MRCC_PeripheralClockConfig, GPIO_Init, EIC_IRQInit, EIC_IRQCmd, PWM_DeInit, PWM_StructInit, PWM_Init, PWM_TRGOSelection, PWM_ClearFlag, PWM_ITConfig, PWM_ResetCounter, ADC_StructInit, ADC_Init, ADC_Cmd, ADC_StartCalibration, ADC_ConversionCmd, PWM_Cmd. **Motor Control Library:** SVPWM_IcsCurrentReadingCalibration

SVPWM_IcsCurrentReadingCalibration

Synopsis	void SVPWM_IcsCurrentReadingCalibration(void);
Description	The purpose of this function is to store the two analog voltages corresponding to zero current values for compensating the offset introduced by both ICS and amplification network.
Input	None
Returns	None
Caution	The function must be called before PWM outputs are enabled so that current flowing through inverter legs is zero. When using the STR750 MC Kit, ICS sensors must be supplied before the control board (MB469B). This way, the current sensing conditioning network can reach steady state before performing calibration.
Functions called	**Standard Library:** ADC_GetFlagStatus, ADC_ConversionCmd, ADC_GetConversionValue

SVPWM_IcsGetPhaseCurrentValues

Synopsis	Curr_Components SVPWM_IcsGetPhaseCurrentValues(void);
Description	This function computes current values of Phase A and Phase B in q15 format from the values acquired from the A/D converter.
Input	None
Returns	Curr_Components type variable
Caution	In order to have a q1.15 format for the current values, the digital value corresponding to the offset must be subtracted when reading phase current A/D converted values. Thus, the function must be called after SVPWM_IcsCurrentReadingCalibration.
Functions called	None

SVPWM_IcsCalcDutyCycles

Synopsis	void SVPWM_IcsCalcDutyCycles (Volt_Components Stat_Volt_Input);
Description	After execution of the PMSM FOC algorithm, new stator voltages component V_α and V_β are computed. The purpose of this function is to calculate exactly the three duty cycles to be applied to motor phases from the values of these voltage components. Refer to *Section 4.2.2* for details about the theoretical approach of SVPWM and its implementation.
Input	V_α and V_β
Returns	None
Caution	None
Functions called	None

4.2.6 Isolated current sensor topology current sampling and general purpose (GP) A/D conversions integration

The three currents I_1, I_2, and I_3 flowing through a three-phase system follow the mathematical relationship:

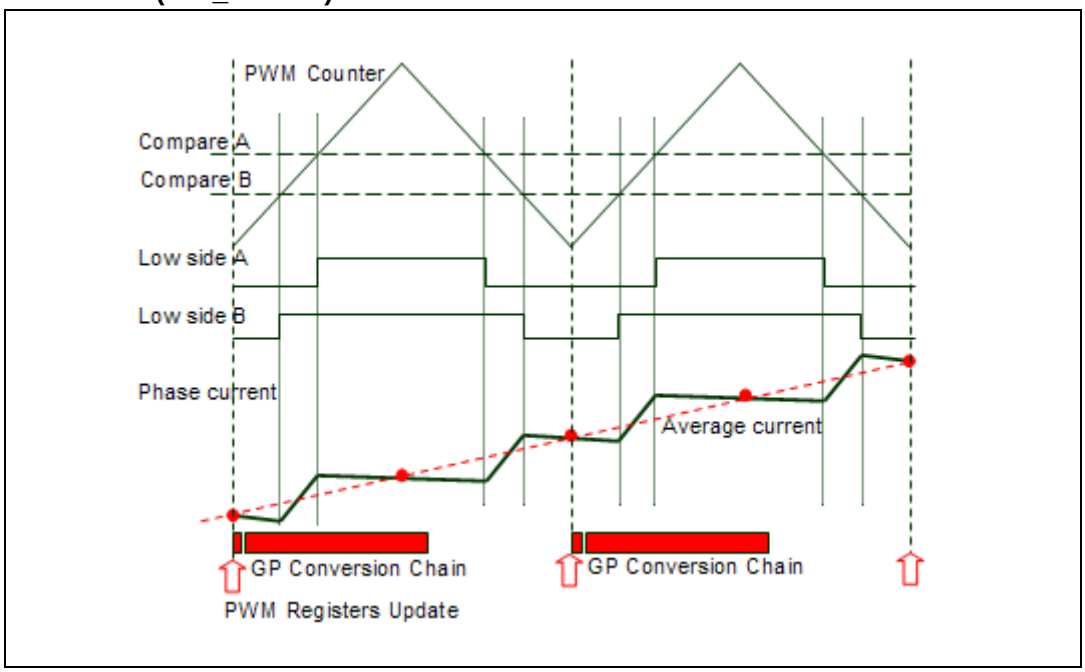
$$I_1+I_2+I_3=0$$

Therefore, to reconstruct the currents flowing through a generic three-phase load, it is sufficient to sample only two out of the three currents while the third one can be computed by using the above relationship.

The flexibility of the STR750 A/D converter trigger makes it possible to synchronize the two A/D conversions necessary for reconstructing the stator currents flowing through the three-phase AC induction motor with the PWM registers update whose rate is also adjusted by the repetition counter. This is important because, as shown in *Figure 31*, it is precisely during counter overflow and underflow that the average level of current is equal to the sampled current. Refer to the STR750 Reference Manual to learn more about A/D conversion triggering and the repetition counter.

Finally, at the end of the injected chain conversion for current reading, the general purpose A/D conversions are performed while the CPU executes the PMSM FOC algorithm.

Figure 31. Stator currents sampling and GP conversions in ICS configuration (`REP_RATE=1`)



4.2.7 `MC_Clarke_Park.h` module

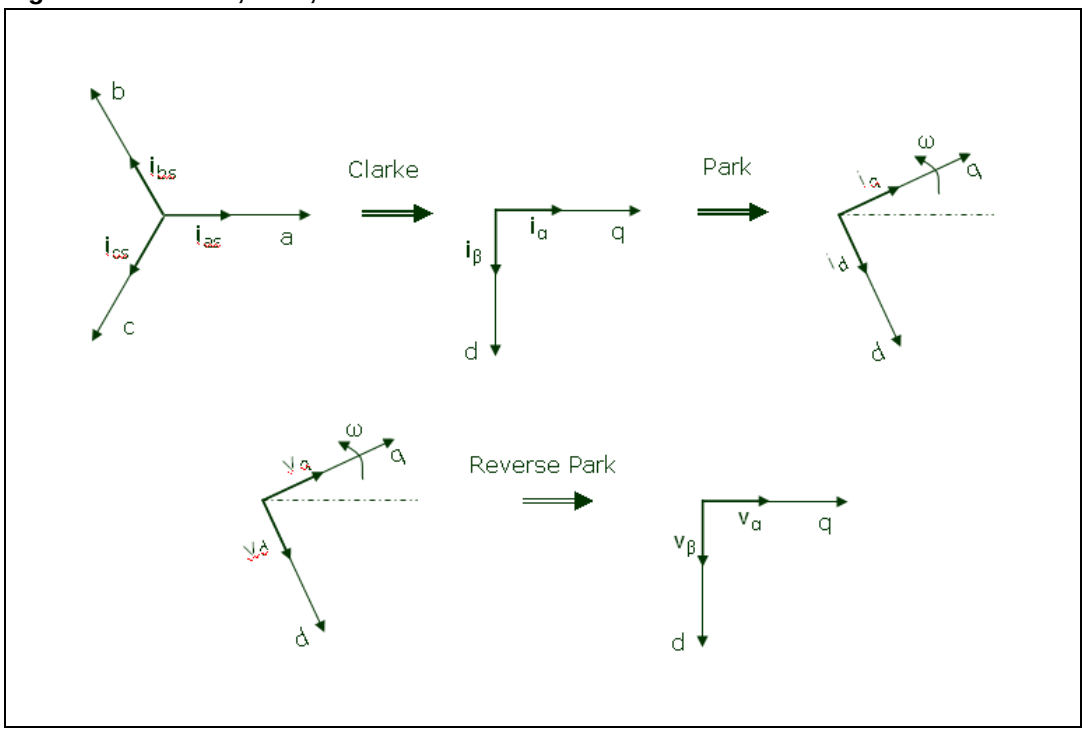
This module is designed to perform transformations of electric quantities between frames of reference that rotate at different speeds.

Based on the arbitrary reference frame theory, the module provides three functions, named after two pioneers of electric machine analysis, E. Clarke and R.H. Park.

These functions implement three variable changes that are required to carry out field-oriented control (FOC):

- those required to carry out field oriented control (FOC): Clarke transforms stator currents to a stationary orthogonal reference frame (named *qd* frame, see *Figure 32*);
- then, from that arrangement, Park transforms currents to a frame that rotates at an arbitrary speed (which, in PMSM FOC drive, is synchronous with the rotor flux);
- Reverse Park transformation brings back stator voltages from a rotating *qd* frame to a stationary one.

Figure 32. Clarke, Park, and Reverse Park transformations



Clarke

Synopsis	Curr_Components Clarke (Curr_Components Curr_Input)
Description	This function transforms stator currents i_{as} and i_{bs} (which are directed along axes each displaced by 120 degrees) into currents i_α and i_β in a stationary *qd* reference frame; *q,d* axes are directed along paths orthogonal to each other. See *Section 4.2.8* for the details.
Input	Stator currents i_{as} and i_{bs} (in q1.15 format) as members of the variable Curr_Input, which is a structure of type Curr_Components.
Returns	Stator currents i_α and i_β (in q1.15 format) as members of a structure of type Curr_Components.
Functions called	mul_q15_q15_q31

Park

Synopsis	Curr_Components Park (Curr_Components Curr_Input, s16 Theta)
Description	The purpose of this function is to transform stator currents i_α and i_β, which belong to a stationary qd reference frame, to a rotor flux synchronous reference frame (properly oriented), so as to obtain i_{qs} and i_{ds}. See *Section 4.2.8* for the details.
Input	Stator currents i_α and i_β (in q1.15 format) as members of the variable Curr_Input, which is a structure of type Curr_Components; rotor flux angle $\theta_{\lambda r}$ (65536 pulses per revolution).
Returns	Stator currents i_{qs} and i_{ds} (in q1.15 format) as members of a structure of type Curr_Components.
Functions called	mul_q15_q15_q31

Rev_Park

Synopsis	Volt_Components Rev_Park (Volt_Components Volt_Input)
Description	This function transforms stator voltage v_q and v_d, belonging to a rotor flux synchronous rotating frame, to a stationary reference frame, so as to obtain v_α and v_β. See *Section 4.2.8* for the details.
Input	Stator voltages v_{qs} and v_{ds} (in q1.15 format) as members of the variable Volt_Input, which is a structure of type Volt_Components.
Returns	Stator voltages v_α and v_β (in q1.15 format) as members of a structure of type Volt_Components.
Caution	None.
Functions called	mul_q15_q15_q31

`Rev_Park_Circle_Limitation`

Synopsis	void RevPark_Circle_Limitation(void)
Description	After the two new values (V_d and V_q) of the stator voltage producing flux and torque components of the stator current, have been independently computed by flux and torque PIDs, it is necessary to saturate the magnitude of the resulting vector, equal to $\sqrt{V_d^2 + V_q^2}$ passing before them to the SVPWM block. The purpose of this routine is to perform the saturation. Refer to *Section 4.2.9: Circle limitation on page 56* for more detailed information
Input	None
Returns	None
Caution	The limitation of the stator voltage vector must be done in accordance with the PWM frequency as shown in *Table 3: PWM frequency vs maximum duty cycle on page 47*.
Functions called	None

4.2.8 Detailed explanation about reference frame transformations

By making a change of variables, that refers stator and rotor quantities to a frame of reference rotating at any angular velocity, it is possible to reduce the complexity of the system electrical equations.

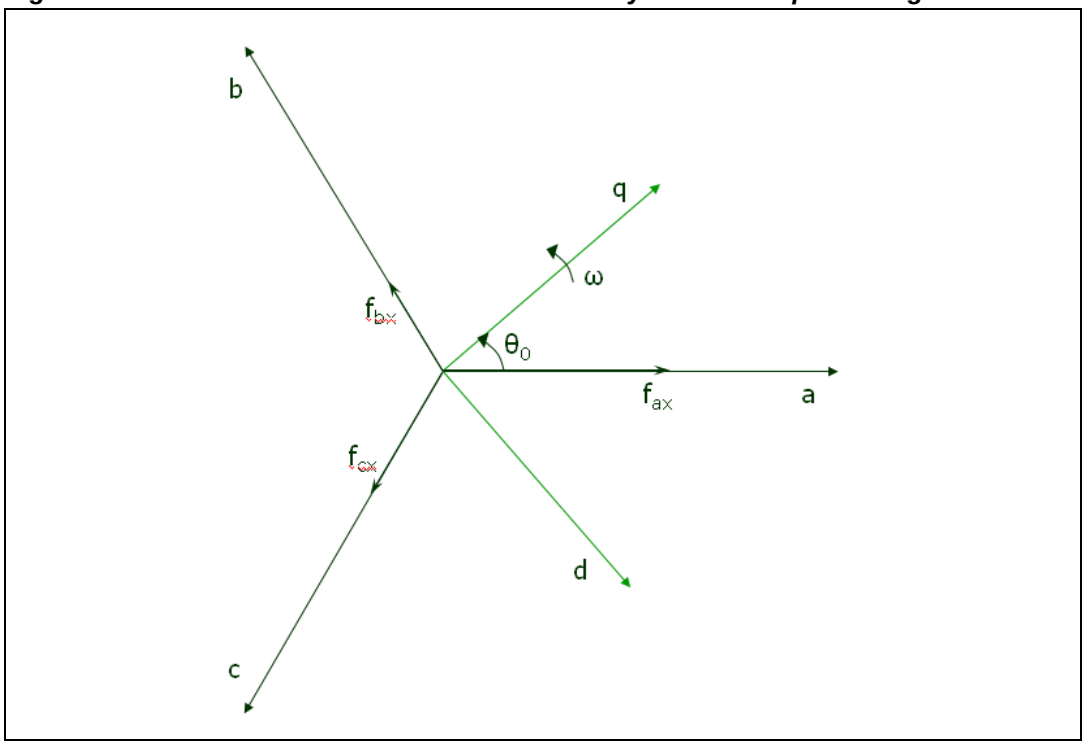
This strategy is often referred to as the Reference-Frame theory [1].

Supposing f_{ax}, f_{bx}, f_{cx} are three-phase instantaneous quantities directed along axis each displaced by 120 degrees, where x can be replaced with s or r to treat stator or rotor quantities (see *Figure 33*); supposing f_{qx}, f_{dx}, f_{0x} are their transformations, directed along paths orthogonal to each other; the equations of transformation to a reference frame (rotating at an arbitrary angular velocity ω) can be expressed as:

$$f_{qdox} = \begin{pmatrix} f_{qx} \\ f_{dx} \\ f_{0x} \end{pmatrix} = \frac{2}{3} \begin{bmatrix} \cos\theta & \cos\left(\theta - \frac{2\pi}{3}\right) & \cos\left(\theta + \frac{2\pi}{3}\right) \\ \sin\theta & \sin\left(\theta - \frac{2\pi}{3}\right) & \sin\left(\theta + \frac{2\pi}{3}\right) \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{pmatrix} f_{ax} \\ f_{bx} \\ f_{cx} \end{pmatrix}$$

where θ is the angular displacement of the q-d reference frame at the time of observation, and θ_0 that displacement at t=0 (see *Figure 33*).

Figure 33. Transformation from an *abc* stationary frame to a *qd* rotating frame



With Clark's transformation, stator currents i_{as} and i_{bs} (which are directed along axes each displaced by 120 degrees) are resolved into currents i and i on a stationary *qd* reference frame.

Appropriate substitution into the general equations (given above) yields:

$$i_\alpha = i_{as}$$
$$i_\beta = \frac{i_{as} + 2i_{bs}}{\sqrt{3}}$$

In Park's change of variables, stator currents i_α and i_β, which belong to a stationary qd reference frame, are resolved to a rotor flux synchronous reference frame (properly oriented), so as to obtain i_{qs} and i_{ds}.

Consequently, with this choice of reference, $\omega=\omega_{\lambda r}$; thus:

$$i_{qs} = -i_\alpha \sin\theta + i_\beta \cos\theta$$
$$i_{ds} = i_a \cos\theta + i_\beta \sin\theta$$

On the other hand, reverse Park transformation takes back stator voltage v_q and v_d, belonging to a rotor flux synchronous rotating frame, to a stationary reference frame, so as to obtain v_α and v_β:

$$v_\alpha = -v_{qs} \sin\theta + v_{ds} \cos\theta$$
$$v_\beta = v_{qs} \cos\theta + v_{ds} \sin\theta$$

4.2.9 Circle limitation

As discussed above, FOC allows to separately control the torque and the flux of a 3-phase permanent magnet motor. After the two new values(V_d^* and V_q^*) of the stator voltage producing flux and torque components of the stator current, have been independently computed by flux and torque PIDs, it is necessary to saturate the magnitude of the resulting vector ($|\vec{V}^*|$) before passing them to the SVPWM block.

The saturation boundary is normally given by the value (S16_MAX=32767) which produces the maximum output voltage magnitude (corresponding to a duty cycle going from 0% to 100%).

Nevertheless, when using three shunt resistor configuration and depending on PWM frequency, it might be necessary to limit the maximum PWM duty cycle to guarantee the proper functioning of the stator currents reading block.

For this reason, the saturation boundary could be a value slightly lower than S16_MAX depending on PWM switching frequency when using three shunt resistor configuration.

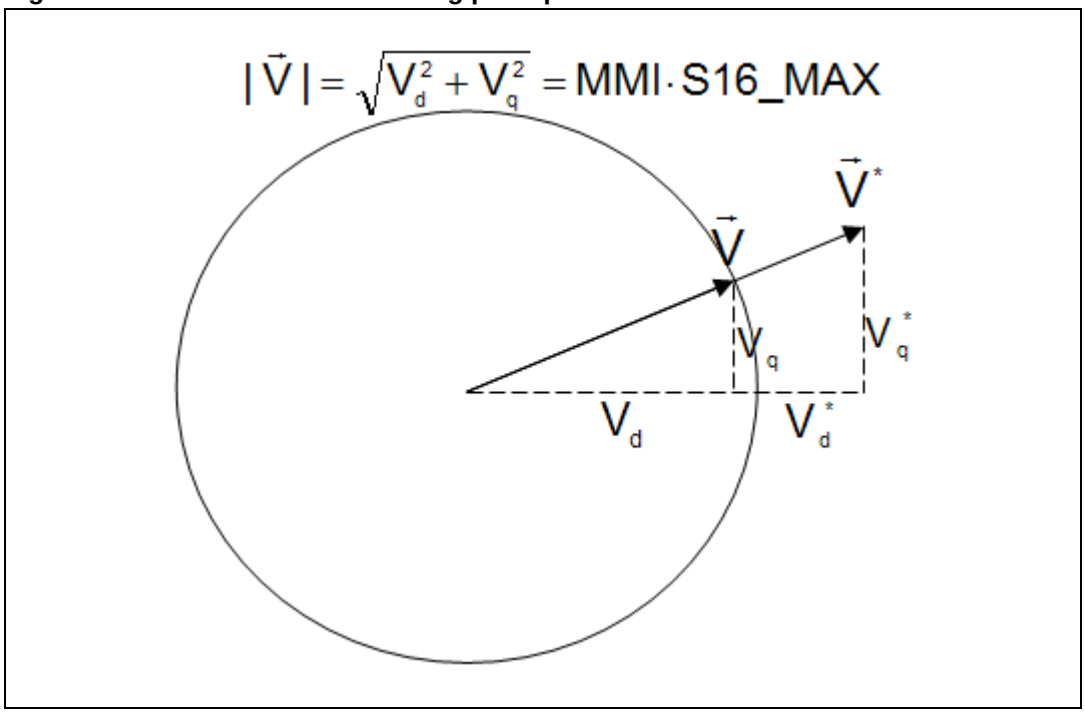
Table 3 on page 47, repeated below for convenience, shows the maximum applicable modulation index as a function of PWM switching frequency when using the STR750-MCKIT.

PWM frequency	Max duty cycle	Max modulation index (MMI)
Up to 11.4kHz	100%	100%
12.2kHz	99.5%	99%
12.9kHz	99%	98%
13.7kHz	98.5%	97%
14.4kHz	98%	96%
15.2kHz	97.5%	95%
16kHz	97%	94%
16.7kHz	96.5%	93%
17.5kHz	96%	92%

Note: The figures above were measured using the MB459 board. This evaluation platform is designed to support several motor driving topologies (PMSM and AC induction) and current reading strategies (single and three-shunt resistors). Therefore, the figures provided in should be understood as a starting point and not as a best case.

The `RevPark_Circle_Limitation` function performs the discussed stator voltage components saturation, as illustrated in *Figure 34*.

Figure 34. Circle limitation working principle



V_d and V_q represent the saturated stator voltage component to be passed to the SVPWM block. From geometrical considerations, it is possible to draw the following relationship:

$$V_d = \frac{V_d^* \cdot MMI \cdot S16\_MAX}{|\vec{V}^*|}$$

$$V_q = \frac{V_q^* \cdot MMI \cdot S16\_MAX}{|\vec{V}^*|}$$

In order to speed up the computation of the above equations while keeping an adequate resolution, the value

$$\frac{MMI \cdot S16\_MAX^2}{|\vec{V}^*|}$$

is computed and stored in a look-up table for different values of $|\vec{V}^*|$. Furthermore, considering that MMI depends on the selected PWM frequency, a look-up table is stored in `MC_Clarke_Park.h` (with MMI ranging from 92 to 100%).

Once you have selected the required PWM switching frequency, you should uncomment the Max Modulation Index definition corresponding to the selected PWM frequency, as shown in *Chapter 2.2.4: Drive control parameters: MC_Control_Param.h file on page 18*.

For information on selecting the PWM switching frequency, you will find advice in *Section A.2 on page 80*. To determine the max modulation index corresponding to the PWM switching frequency, refer to *Table 3 on page 47*.

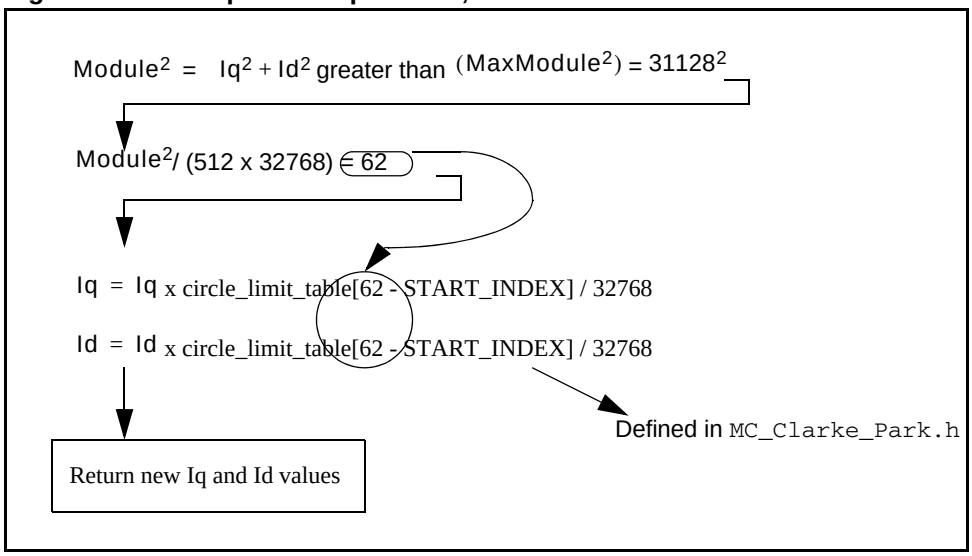
The following section provides an example of how the limitation is performed by the firmware.

Example: max modulation index of 95%.

For a MMI of 95%, the Max module value is 95%*S16_MAX = 31128 (as per the MAX_MODULE value stored in `MC_Clarke_Park.h`).

The *Figure 35* shows the firmware implementation working principle, minimizing the CPU load (look-up table access and shift operation for division). The table `circle_limit_table[...]` stores a maximum of 81 values (generated from the `circle limitation.xls` file located in the `Design tools` folder). Access is as follows:

Figure 35. Example with Iq = 32000, Id = -5000



Note that the actual maximum PWM duty cycle is equal to:

Max duty cycle = 100% - (100% - maximum modulation index chosen) / 2

A 95% modulation index corresponds to a 100 - (100-95) / 2 = 97.5% maximum duty cycle output signals to the power stage.

4.2.10 `75x_encoder.c` module

`ENC_Init`

Synopsis	void ENC_Init(void)
Description	The purpose of this function is to initialize the encoder timer. The peripheral clock, input pins and update interrupt are enabled. The peripheral is configured in 4X mode, which means that the counter is incremented/decremented on the rising/falling edges of both timer input 1 and 2 (TIMx_TI0 and TIMx_TI1 pins).
Functions called	MRCC_PeripheralClockConfig GPIO_Init EIC_IRQInit TIM_StructInit, TIM_Init, TIM_ClearFlag, TIM_ITConfig, TIM_ResetCounter, Tim_Cmd
See also	STR750 datasheet: synchronizable standard timer.

`ENC_GetPosition`

Synopsis	u32 ENC_GetPosition(void)
Description	This function returns the encoder timer value, giving a direct reading of the rotor position from 0 to 4*(number of encoder pulses per revolution). For the SHINANO motor included with the starter kit, the encoder delivers 400 pulses per revolution. This routine returns: 0 for 0 degrees, 4*400/2=800 for 180 degrees.
Input	None
Output	Unsigned 32 bits
Functions called	None
See also	STR750 datasheet: synchronizable standard timer.

`ENC_Get_Electrical_Angle`

Synopsis	s16 ENC_Get_Electrical_Angle(void)
Description	This function returns the electrical angle in signed 16-bit format. This routine returns: 0 for 0 degrees, -32768 (S16_MIN) for -180 degrees, +32767 (S16_MAX) for +180 degrees.
Input	None
Output	Signed 16 bits
Functions called	None

ENC_Get_Mechanical_Angle

Synopsis	s16 ENC_Get_Electrical_Angle(void)
Description	This function returns the mechanical angle in signed 16-bit format. This routine returns: 0 for 0 degrees, -32768 (S16_MIN) for -180 degrees, +32767 (S16_MAX) for +180 degrees.
Input	None
Output	Signed 16 bits
Functions called	None
Caution	**Link between Electrical/Mechanical frequency/RPM** Electrical frequency = number of pair poles x mechanical frequency RPM speed = 60 x Mechanical frequency (RPM: revolutions per minute) example: electrical frequency = 100 Hz, motor with 8 pair poles: *100Hz electrical <-> 100/8 =12.5Hz mechanical <-> 12.5 x 60=750 RPM*

ENC_ResetEncoder

Synopsis	void ENC_resetEncoder(void)
Description	This function resets the encoder timer (hardware register) value to zero.
Functions called	TIM_ResetCounter
See also	STR750 datasheet: synchronizable standard timer.

ENC_Clear_Speed_Buffer

Synopsis	void ENC_Clear_Speed_Buffer(void)
Description	This function resets the buffer used for speed averaging.
Functions called	None

ENC_Get_Speed

Synopsis	s16 ENC_Get_Speed(void)
Description	This function returns the rotor speed in Hz. The value returned is given with 0.1Hz resolution, which means that 1234 is equal to 123.4 Hz.
Input	None
Output	Signed 16 bits
Functions called	None
Caution	This routine returns the mechanical frequency of the rotor. To find the electrical speed, use the following conversion: *electrical frequency = number of pole pairs * mechanical frequency*

ENC_Get_Average_Speed

Synopsis	s16 ENC_Get_Average_Speed(void)
Description	This function returns the average rotor speed in Hz. The value returned is given with 0.1Hz resolution, which means that 1234 is equal to 123.4 Hz.
Input	None
Output	Signed 16 bits
Functions called	ENC_Get_Speed()
Note	The averaging is done with the values stored in ‘Speed_Buffer[]’. The size of this buffer is set through the ‘SPEED_BUFFER_SIZE’ statement, which must be equal to a power of 2 to allow the use of the shift operation for divisions.
Caution	This routine returns the mechanical frequency of the rotor. To find the electrical speed, use the following conversion: *electrical frequency = mechanical frequency * number of pole pairs*

TIMx_UP_IRQHandler - interrupt routine

Synopsis	void TIMx_UP_IRQHandler(void)
Description	This is the encoder timer (TIMER 0, 1 or 2) update routine. An interruption is generated whenever an overflow/underflow of the counter value occurs (TIM_CNT). The ‘Encoder_Timer_Overflow’ variable is then incremented.
Functions called	None
Caution	This is an interrupt routine.
See also	STR750 Datasheet: Synchronizable Standard Timer.

4.2.11 `75x_TBTimer.c` module

`TB_Timebase_Timer_Init`

Synopsis	void TB_Timebase_Timer_Init(void)
Description	The purpose of this function is to initialize the Timebase Timer. The peripheral clock, interrupt, auto-reload value and counter mode are set up. The peripheral is configured to generate an interruption every 500µs, thus providing a general purpose timebase.
Functions called	EIC_IRQInit TB_StructInit, TB_Init, TB_ITConfig, TB_Cmd, TB_ResetCounter TB_ResetCounter
See also	STR750 datasheet: timebase timer.

`TB_Wait`

Synopsis	void TB_Wait(u16 time)
Description	This function produces a programmable delay equal to the time variable multiplied by 500µs.
Input	Unsigned 16 bits
Output	None
Functions called	None
Caution	This routine exits only after the programmed delay has elapsed. Meanwhile, the code execution remains frozen in a waiting loop. Care should be taken when this routine is called at main/interrupt level: a call from an interrupt routine with a higher priority than the timebase interrupt will freeze code execution.
See also	STR750 datasheet: timebase timer.

`TB_Set_Delay_500us`

Synopsis	void TB_Set_Delay_500us(u16 hDelay)
Description	This function is used to update the wTimebase_500us static variable.
Input	Unsigned 16 bits
Output	None
Functions called	None

TB_Delay_IsElapsed

Synopsis	bool TB_Delay_IsElapsed(void)
Description	This function returns TRUE if ‘wTimebase_500us’ variable has reached 0, else FALSE.
Input	None
Output	Boolean

TB_Set_DisplayDelay_500us

Synopsis	void TB_Set_DisplayDelay_500us(u16 hDelay)
Description	This function is used to update the ‘wTimebase_display_500us’ static variable.
Input	Unsigned 16 bits
Output	None

TB_Set_DebounceDelay_500us

Synopsis	void TB_Set_DebounceDelay_500us(u16 hDelay)
Description	This function is used to update the ‘wKey_debounce_500us’ static variable.
Input	Unsigned 16 bits
Output	None

TB_DebounceDelay_IsElapsed

Synopsis	bool TB_DebounceDelay_IsElapsed(void)
Description	This function returns TRUE if ‘wKey_debounce_500us’ variable has reached 0, else FALSE.
Input	None
Output	Boolean

TB_IRQHandler

Synopsis	void TB_IRQHandler(void)
Description	This is the Timebase timer interrupt routine. This peripheral is configured to produce an interruption every 500µs, thus providing a general purpose timebase allowing the refresh of various variables used mainly as counters (for example PID sampling time).
Input	None
Output	None
Functions called	ENC_Get_Average_Speed PID_Speed_Regulator TB_ClearFlag
Note	This is an interrupt routine.
See also	STR750 datasheet: timebase timer.

4.2.12 `75x_it.c` module

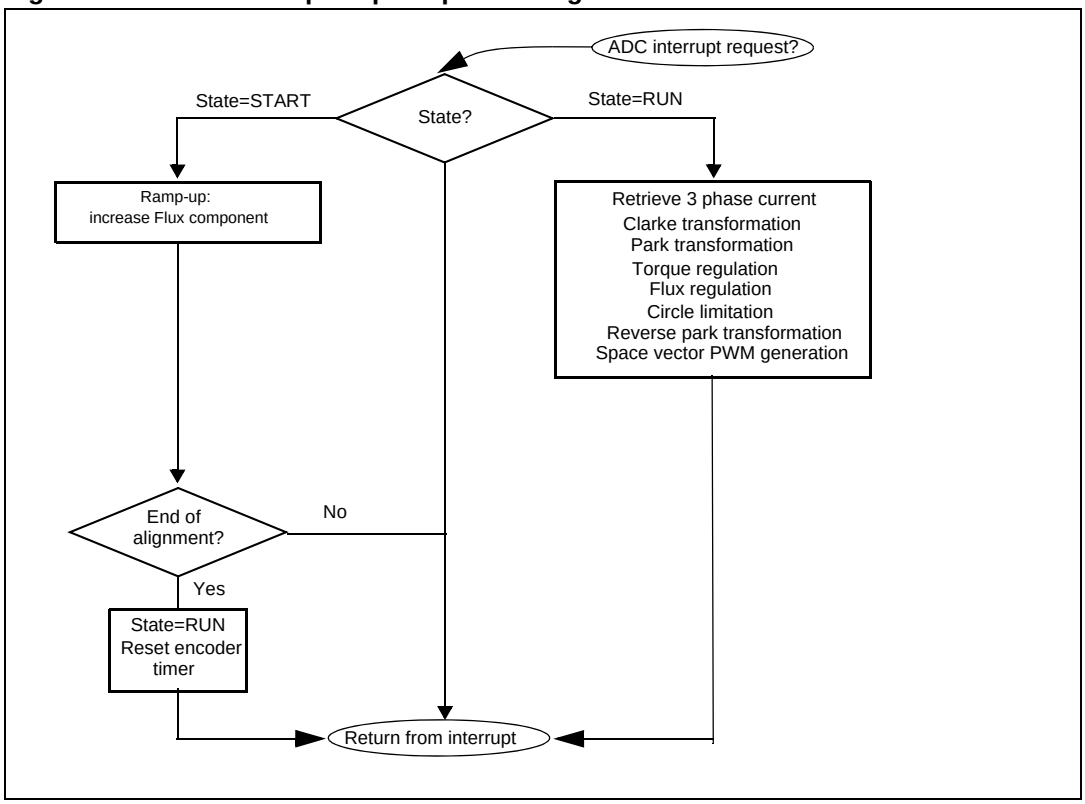
`PWM_EM_IRQHandler`

Synopsis	void PWM_EM_IRQHandler(void)
Description	The purpose of this function is to refresh the 'wGlobal_Flags' and 'State' variables upon detection of a signal on the dedicated emergency pin.
Functions called	PWM_ClearFlag, PWM_ITConfig
Note	This is an interrupt routine.
See also	STR750 datasheet: synchronizable PWM timer.

`ADC_IRQHandler`

Synopsis	void ADC_IRQHandler(void)
Description	The purpose of this function is to handle the ADC interrupt request. All the PMSM FOC algorithm is processed in this interrupt routine. Triggered by ADC ECH / JECH ISR, the function loads stator currents (read by ICS or shunt resistors) and carries out Clark and Park transformations, converting them to $i_{qs}^{\lambda r}$ and $i_{ds}^{\lambda r}$ (see *Figure 5*). Then, these currents are fed to PID regulators together with reference values $i_{qs}^{\lambda r}$ * and $i_{ds}^{\lambda r}$ *. The regulator output voltages $v_{qs}\lambda^{r}$ * and $v_{ds}\lambda^{r}$ * then must be transformed back to a stator frame (through Reverse Park conversion), and finally drive the power stage. In order to correctly perform Park and Reverse Park transformation, it is essential to accurately estimate the rotor flux position ($\theta^{\lambda r}$): this is done by calling the ENC_Get_Electrical_Angle routine.
Functions called	***3 shunts configuration:*** Clarke, Park, PID_Torque_Regulator, PID_Flux_Regulator, RevPark_Circle_Limitation, Rev_Park, SVPWM_3ShuntCalcDutyCycles, ENC_ResetEncoder, SVPWM_3ShuntGPADCConfig, SVPWM_3ShuntGetPhaseCurrentValues ***Isolated current sensors (ICS) configuration:*** Clarke, Park, PID_Torque_Regulator, PID_Flux_Regulator, RevPark_Circle_Limitation, Rev_Park, SVPWM_IcsCalcDutyCycles, ENC_ResetEncoder, SVPWM_IcsGetPhaseCurrentValues
Note	This is an interrupt routine.
See also	STR750 datasheet: synchronizable PWM timer.

Figure 36. ADC interrupt request processing



4.2.13 `MC_PID_regulators.c` module

`PID_Init`

Synopsis	void PID_Init(void)
Description	The purpose of this function is to initialize the PID for torque, flux and speed regulation. For each one, a set of default values are loaded: target (speed, torque or flux), proportional, integral and derivative gains, lower and upper limiting values for the output.
Functions called	None
Note	Default values for PID regulators are declared and can be modified in **MC_Control_Param.h** file (see *Section I on page 19*).

PID_Flux_Regulator

Synopsis	s16 PID_Flux_regulator(PID_FluxTYPEDEF *PID_Flux, s16 qId_input)
Description	The purpose of this function is to compute the proportional, integral and derivative terms (if enabled, see Id_Iq_DIFFERENTIAL_TERM_ENABLED in *Section 2.2.1 on page 16*) for the flux regulation.
Input	PID_FluxTYPDEF (see 'MC_type.h' for structure declaration) signed 16 bits
Output	Signed 16 bits
Functions called	None
Note	Default values for the PID flux regulation are declared and can be modified in the **MC_Control_Param.h** file (see *Section 2.2.4 on page 18*).
See also	*Figure 42 on page 74* shows the PID block diagram. *Chapter 5: PID regulator implementation and tuning on page 70*.

PID_Torque_Regulator

Synopsis	s16 PID_Torque_regulator(PID_TorqueTYPEDEF *PID_Torque, s16 qIq_input)
Description	The purpose of this function is to compute the proportional, integral and derivative terms (if enabled, see Id_Iq_DIFFERENTIAL_TERM_ENABLED in *Section 2.2.1 on page 16*) for the torque regulation.
Input	PID_TorqueTYPDEF (see **MC_type.h** for structure declaration) signed 16 bits
Output	signed 16 bits
Functions called	None
Note	Default values for the PID torque regulation are declared and can be modified in the **MC_Control_Param.h** file (see *Section 2.2.4 on page 18*).
See also	*Figure 42 on page 74* shows the PID block diagram. *Chapter 5: PID regulator implementation and tuning on page 70*.

PID_Speed_Regulator

Synopsis	s16 PID_Speed_regulator(PID_SpeedTYPEDEF *PID_Speed, s16 speed)
Description	The purpose of this function is to compute the proportional, integral and derivative terms (if enabled, see SPEED_DIFFERENTIAL_TERM_ENABLED in section 2.2.1 on page 16) for the speed regulation.
Input	PID_SpeedTYPDEF (see 'MC_type.h' for structure declaration) signed 16 bits
Output	signed 16 bits
Functions called	None
Note	Default values for the PID speed regulation are declared and can be modified in the **MC_Control_Param.h** file (see *Section 2.2.4 on page 18*).
See also	*Figure 43 on page 75* shows the PID block diagram. *Chapter 5: PID regulator implementation and tuning on page 70*.

PID_Reset_Integral_terms

Synopsis	void PID_Reset_Integral_terms(void)
Description	The purpose of this function is to reset all the integral terms of the torque, flux and speed PID regulators.

PID_Speed_Coefficients_update

Synopsis	void PID_Speed_coefficients_update(s16 motor_speed)
Description	This function automatically computes the proportional, integral and derivative gain for the speed PID regulator according to the actual motor speed. The computation is done following a linear curve based on 4 set points. See *Section 5.2.2 on page 72* for more information.
Functions called	None
Note	Default values for the four set points are declared and can be modified in the **MC_Control_Param.h** file (see *Section 2.2.4 on page 18*).

PID_Integral_Speed_update

Synopsis	void PID_Integral_Speed_update(s32 value)
Description	The purpose of this function is to load the speed integral term with a default value.

4.3 Application layer

The application layer is split into several modules, mainly for the control of the keys, LCD display, temperature and bus voltage monitoring, and main loop. The following is a brief description of these modules.

4.3.1 `main.c` module

This module contains the initialization and the main control loop of the overall firmware.

4.3.2 `MC_Keys.c` module

The purpose of the MC_Keys.c module is to centralize all information regarding the keyboard reading. Any action on the keyboard is processed in the Keys_process routine.

4.3.3 `MC_Display.c` module

The purpose of the MC_Display.c module is to centralize all information regarding the LCD display management.

4.3.4 `75x_LCD.c` module

This module contains some dedicated routines for the control of the LCD embedded with the starter kit.

4.3.5 `MC_dac.c` module

This module contains some dedicated routines for the control of an external digital-to-analog (DAC) device (AD7303).

4.3.6 `MC_misc.c` module

This module contains some dedicated routines for monitoring the temperature of the power stage and the bus voltage.

5 PID regulator implementation and tuning

The regulators implemented for Torque, Flux and Speed are actually Proportional Integral Derivative (PID) regulators (see note below regarding the derivative term). PID regulator theory and tuning methods are subjects which have been extensively discussed in technical literature. *Section 5.1* provides a basic reminder of the theory.

5.1 Theoretical background

The purpose of such regulators is to maintain a level of torque, flux or speed according to a desired target.

Figure 37. PID general equation

torque = f(rotor position)
flux = f(rotor position)
} torque and flux regulation for maximum system efficiency

torque = f(rotor speed)
} torque regulation for speed regulation of the system

Where: $Error_{sys_T}$ Error of the system observed at time t = T

$Error_{sys_{T-1}}$ Error of the system observed at time t = T - Tsampling

$$f(X_T) = K_p \times Error_{sys_T} + K_i \times \sum_{0}^{T} Error_{sys_t} + K_d \times (Error_{sys_T} - Error_{sys_{T-1}}) \quad (1)$$

Derivative term can be disabled

Equation 1 corresponds to a classical PID implementation, where:

- K_p is the proportional coefficient,
- K_i is the integral coefficient.
- K_d is the differential coefficient.

Note: As mentioned in Figure 37, the derivative term of the PID can be disabled independently (through a compiler option, see `75x_MCconf.h` file) for the torque/flux or the speed regulation; a PI can then be quickly implemented whenever the system doesn't require a PID control algorithm.

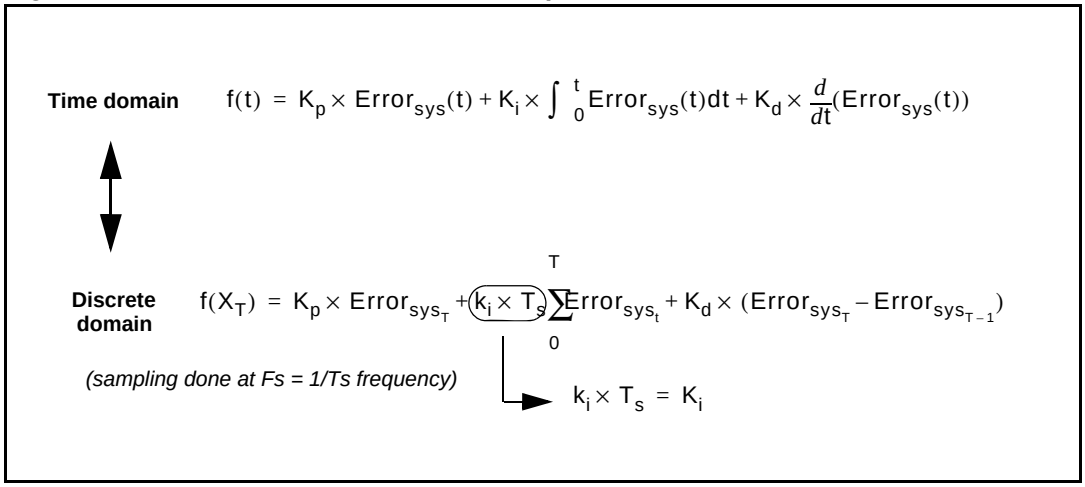
5.2 Regulation sampling time

The sampling time needs to be modified to adjust the regulation bandwidth. As an accumulative term (the integral term) is used in the algorithm, increasing the loop time decreases its effects (accumulation is slower and the integral action on the output is delayed). Inversely, decreasing the loop time increases its effects (accumulation is faster and the integral action on the output is increased). This is why this parameter has to be adjusted prior to setting up any coefficient of the PID regulator.

In order to keep the CPU load as low as possible and as shown in equation (1) in *Figure 37*, the sampling time is directly part of the integral coefficient, thus avoiding an extra

multiplication. *Figure 38* describes the link between the time domain and the discrete system.

Figure 38. Time domain to discrete PID equations



5.2.1 Adjusting the regulation sampling time

In theory, the higher the sampling rate, the better the regulation. In practice, you must keep in mind that:

- The related CPU load will grow accordingly.
- For speed regulation, there is absolutely no need to have a sampling time lower than the refresh rate of the speed information fed back by the external sensors; this becomes especially true when all sensors are used while driving the motor at low to medium speed.
- At high speed, in most cases, system inertia is such that the system response is slow: in these conditions, there is no need to have a high sampling rate.

The speed regulation loop sampling time must be set in the `75x_TBtimer.c` file (Time Base timer interrupt routine). Note that the sampling time is actually a multiple of the period of the Timebase timer interrupt update routine (500 µS by default). This is an 8-bit value (255 max).

Figure 39. Speed regulation sampling time adjustment in `75x_TBtimer.c`

```
void TB_IRQHandler(void)
{
.....
if (bPID_Speed_Sampling_Time_500us != 0 )
  {
    bPID_Speed_Sampling_Time_500us --;
  }
  else
  {
  bPID_Speed_Sampling_Time_500us = PID_SPEED_SAMPLING;

.........
}
```

For the torque and flux regulation loop sampling time, the PID_SPEED_SAMPLING parameter must be set in the MC_Control_Param.h file. Note that the sampling time is:

- A multiple of the PWM switching period in three-shunt configuration. The REP_RATE value can only be an odd number (8-bit value).
- A multiple of half of the PWM switching period in isolated current sensor configuration. The REP_RATE value can be any number (8-bit value).

The torque/flux regulation sampling time adjustment is defined as follows in the MC_Control_Param.h file:

```
#define REP_RATE (1)// (N.b): Internal current loop is performed every
                    //             (REP_RATE + 1)/(2*PWM_FREQ) seconds.
                    // REP_RATE has to be an odd number in case of three-shunt
                    // current reading; this limitation doesn't apply to ICS
```

5.2.2 Adjusting the speed regulation loop Ki, Kp and Kd vs the motor frequency

Depending on the motor frequency, it might be necessary, to use different values of K_p, K_i and K_d.

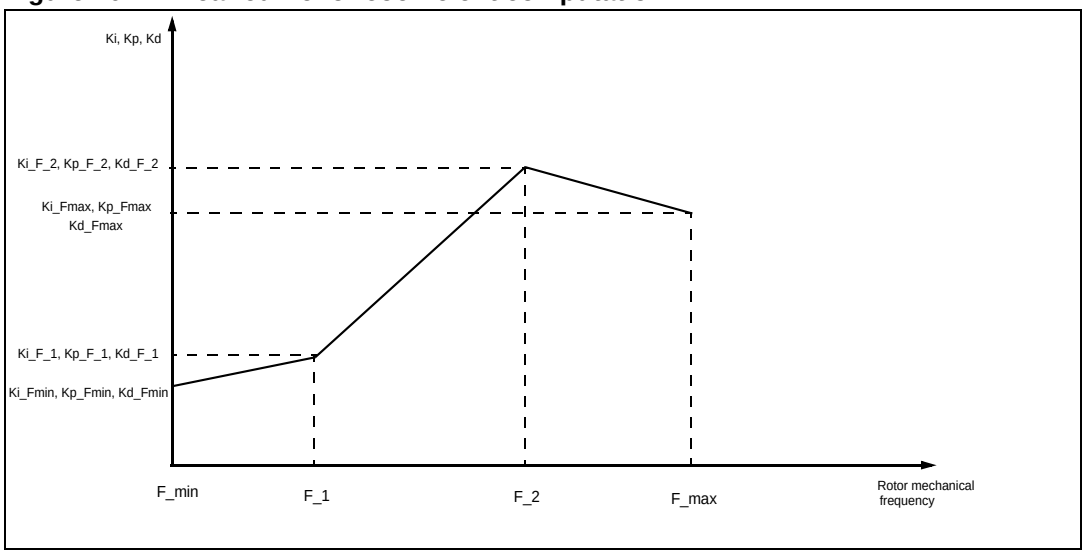
These values have to be input in the code to feed the regulation loop algorithm. A function performing linear interpolation between four set-points (PID_Speed_Coefficient_update) is provided as an example in the software library (see MC_PID_regulators.c) and can be used in most cases, as long as the coefficient values can be linearized. If that is not possible, a function with a larger number of set-points or a look-up table may be necessary.

To enter the four set-points, once the data are collected, edit the MC_Control_param.h file and fill in the field dedicated to the Ki, Kp and Kd coefficient calculation as shown below.

```
//Settings for min frequency
#define Freq_Min   10      // 1 Hz mechanical
#define Ki_Fmin    1000    // Frequency min coefficient settings
#define Kp_Fmin    2000
#define Kd_Fmin    3000

//Settings for intermediate frequency 1
#define F_1                 50 // 5 Hz mechanical
#define Ki_F_1     2000    // Intermediate frequency 1 coefficient settings
#define Kp_F_1     1000
#define Kd_F_1     2500

//Settings for intermediate frequency 2
#define F_2                200 // 20 Hz mechanical
#define Ki_F_2     1000     // Intermediate frequency 2 coefficient settings
#define Kp_F_2     750
#define Kd_F_2     1200

//Settings for max frequency
#define Freq_Max   500     // 50 Hz mechanical
#define Ki_Fmax    500     // Frequency max coefficient settings
#define Kp_Fmax    500
#define Kd_Fmax    500
```

Once the motor is running, integer, proportional and derivative coefficients are computed following a linear curve between F_min and F_1, F_1 and F_2, F_2 and F_max (see *Figure 40*). Note that F_min, F_1, F_2, F_max are mechanical frequencies, with 0.1 Hz resolution (for example F_1 = 1234 means F_1 = 123.4Hz).

Figure 40. Linear curve for coefficient computation



Disabling the linear curve computation routine, `75x_it.c` module

If you want to disable the linear curve computation, you must comment out the PID_Speed_Coefficients_update(..) routine. In this case, the default values for Ki, Kp, Kd for torque, flux and speed regulation are used. See PID_TORQUE_Kx_DEFAULT, PID_FLUX_Kx_DEFAULT, PID_SPEED_Kx_DEFAULT, in the MC_control_Param.h file.

To disable the linear curve computation routine in the `75x_it.c` module:

```
void TB_IRQHandler(void)
{
........
if ((wGlobal_Flags & CLOSED_LOOP) == CLOSED_LOOP)
      {
       if (State == RUN)
       {
        //PID_Speed_Coefficients_update(hRot_Freq_Hz); // to be commented out
....... .
}
```

5.3 Tricks and traps

When tuning the PID parameters you should consider the worst case conditions, which may be when the load varies quickly and unpredictably, when the inertia is at a minimum, or when the mains voltage is maximum for an off-line application.

If regulation tuning is performed in no-load condition (at the highest), it will most probably be unresponsive in the final application, and vice versa: regulation tuning performed in the application may become unstable in no-load conditions.

5.4 Implementing closed loop regulation

Below is an example of the use of the speed regulation process.

Figure 41. Speed regulation loop call in `75x_TBTimer.c`

```
if (bPID_Speed_Sampling_Time_500us != 0 )
{
 bPID_Speed_Sampling_Time_500us --;
}
else
{
bPID_Speed_Sampling_Time_500us = PID_SPEED_SAMPLING;

....
PID_Speed_Coefficients_update(hRot_Freq_Hz);
PID_Torque_InitStructure.Reference = PID_Speed_Regulator(&PID_Speed_InitStructure,
hRot_Freq_Hz); // 6 hz
}
```

PI(D) computed every 2ms

Torque value increased by PID (resp. decreased) when motor speed too low (resp. high)

Note: The `PID_Speed_Regulator` routine needs to be fed with a mechanical frequency input.

The following flow diagrams (*Figure 42* and *Figure 43*) show the decision tree for the computation of the torque/flux and speed regulation routines.

Figure 42. Torque/flux control loop block diagram

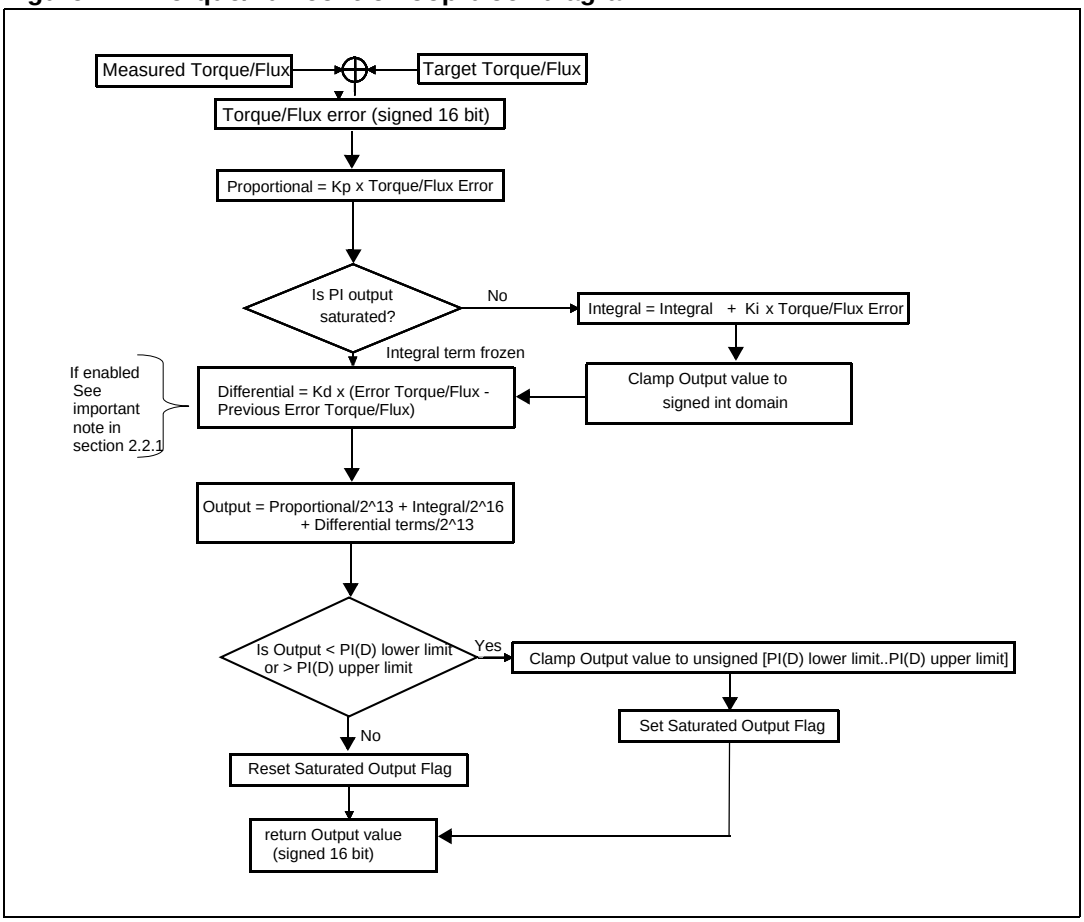
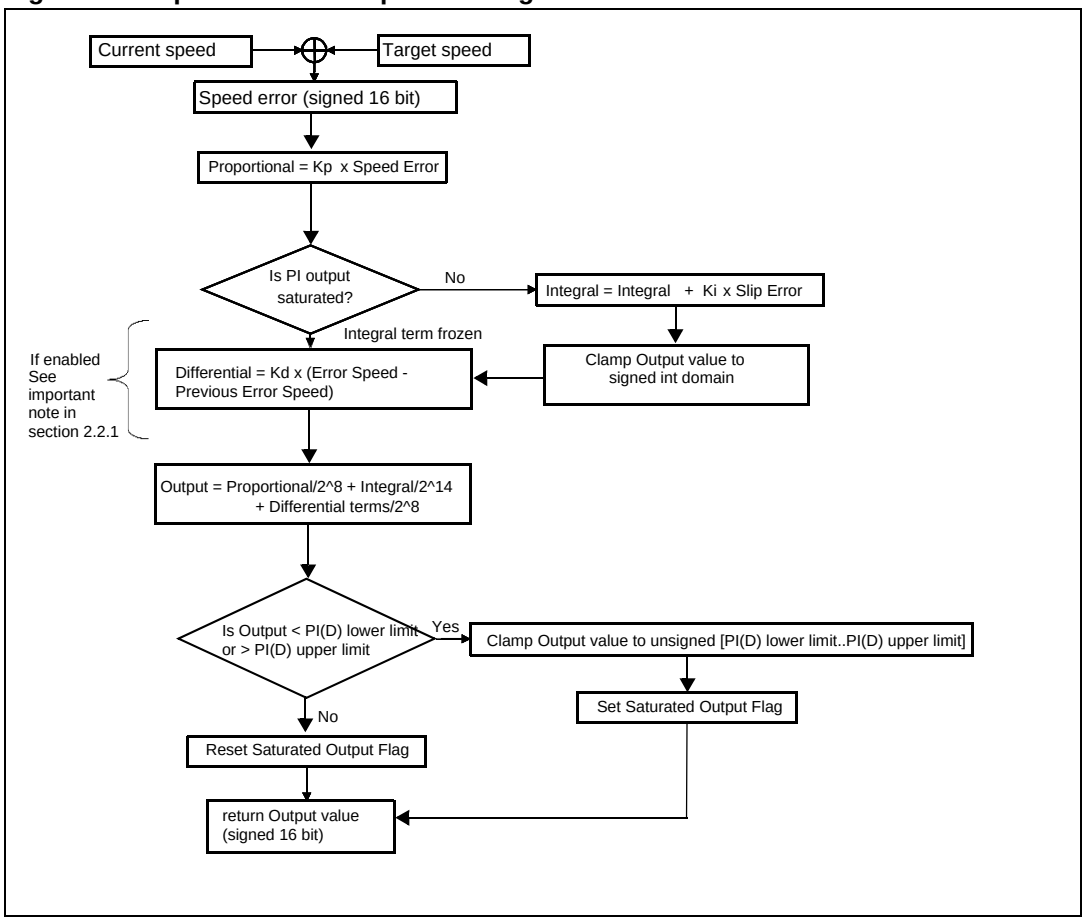


Figure 43. Speed control loop block diagram



6 MISRA compliance

Based on the 'The Motor Industry Software Reliability Association's *Guidelines for the Use of the C Language in Vehicle Based Software*', the purpose of this section is to provide a report of any MISRA deviation in the version 1.0 of the library modules.

6.1 Analysis method

The software library was checked for MISRA compliance using the IAR Embedded Workbench® toolchain. The IAR Systems' implementation is based on version 1 of the MISRA C rules, dated April 1998.

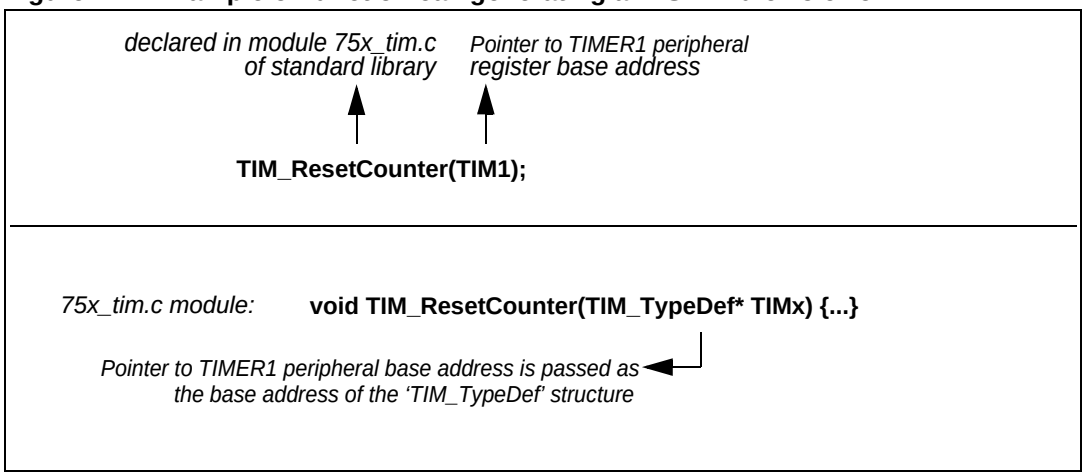
6.2 Limitations

Compliance tests were performed on **required** MISRA rules only, and not on advisory rules.

Due to the extensive use of the STR750 standard library which itself is not fully MISRA compliant (as of September 2006), the interaction (through function calls for example) between the standard library and PMSM library modules necessarily induces non-compliances.

The main reason is due to the fact the STR750 standard library routines rely on base-address pointer parameters (e.g. pointer to a hardware register memory address) that are then re-casted as the first address of a structure inside the function call, as shown in the example in *Figure 44*.

Figure 44. Example of function call generating a MISRA rule 45 error



6.3 MISRA compliance for PMSM library files

Table 4. MISRA compliance of PMSM library files

Module name	MISRA compliant	Deviation
MC_Clarke_Park.h	Yes	
MC_RevPark.h	Yes	
MC_qmath.h	Yes	
MC_const.c	Yes	
MC_const.h	Yes	
MC_type.h	Yes	
75x_TBTimer.c	Yes	
75x_TBTimer.h	Yes	
MC_Globals.c	Yes	
MC_Globals.h	Yes	
MC_Display.c	Yes	
MC_Display.h	Yes	
MC_PMSM_motor_param.h	Yes	
75x_MClib.h	Yes	
MC_Control_Param.h	Yes	
75x_conf.h	Yes	
75x_MCconf.h	Yes	
MC_encoder_param.h	Yes	
75x_svpwm_3shunt.c		MISRA rule 45 non-compliance due to STR750 standard library function call. (See *Section 6.2: Limitations on page 76*)
75x_svpwm_3shunt.h		MISRA rule 45 non-compliance due to STR750 standard library function call. (See *Section 6.2: Limitations on page 76*)
75x_svpwm_ics.c		MISRA rule 45 non-compliance due to STR750 standard library function call. (See *Section 6.2: Limitations on page 76*)
75x_svpwm_ics.h		MISRA rule 45 non-compliance due to STR750 standard library function call. (See *Section 6.2: Limitations on page 76*)
Main.c		MISRA rule 45 non-compliance due to STR750 standard library function call. (See *Section 6.2: Limitations on page 76*)
75x_encoder.c		MISRA rule 45 non-compliance due to STR750 standard library function call. (See *Section 6.2: Limitations on page 76*)
75x_encoder.h	Yes	

Table 4. MISRA compliance of PMSM library files

Module name	MISRA compliant	Deviation
75x_it.c		MISRA rule 45 non-compliance due to STR750 standard library function call. (See *Section 6.2: Limitations on page 76*)
75x_lcd.c		MISRA rule 45 non-compliance due to STR750 standard library function call. (See *Section 6.2: Limitations on page 76*)
75x_lcd.h	Yes	
MC_Keys.c		MISRA rule 45 non-compliance due to STR750 standard library function call. (See *Section 6.2: Limitations on page 76*)
MC_Keys.h	Yes	
MC_Misc.c		MISRA rule 45 non-compliance due to STR750 standard library function call. (See *Section 6.2: Limitations on page 76*)
MC_Misc.h	Yes	
MC_DAC.c		MISRA rule 45 non-compliance due to STR750 standard library function call. (See *Section 6.2: Limitations on page 76*)
MC_DAC.h	Yes	

Appendix A Additional information

A.1 Adjusting CPU load related to PMSM FOC algorithm execution

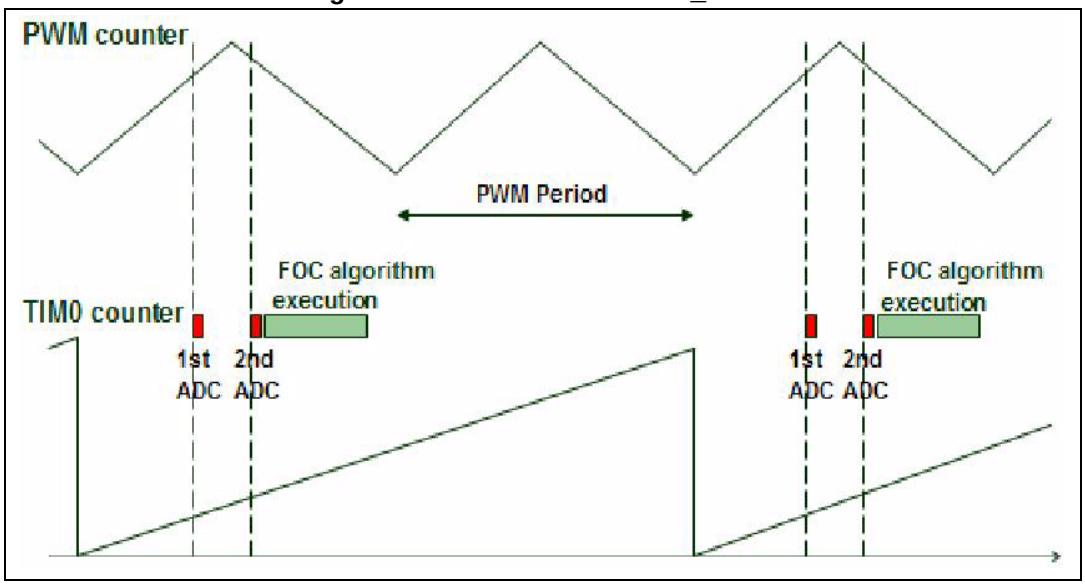
The Synchronizable-PWM Timer peripheral has the built-in capability of updating PWM registers only after a given number of PWM semi-periods. This feature is handled by a programmable repetition counter. It is particularly useful to adjust the CPU load related to PMSM FOC algorithm execution for a given PWM frequency (refer to STR750 Reference Manual for more information on programmable repetition counter).

When using ICS, the injected chain of conversions for current reading is directly triggered by a PWM register update event. Moreover, since the PMSM FOC algorithm is executed at the end of the injected chain of conversions in the related ISR, changing the repetition counter has a direct impact on PMSM FOC refresh rate and thus on CPU load.

However, in the case of three shunt topology current reading, to ensure that the PMSM FOC algorithm is executed once for each PWM register update, it is necessary to keep the synchronization between current conversions triggering and PWM signal. In the proposed software library, this is automatically performed, so that you can reduce the frequency of execution of the PMSM FOC algorithm by simply changing the default value of the repetition counter (the REP_RATE parameter in the `MC_Control_Param.h` header file).

Figure 45 shows current sampling triggering, and PMSM FOC algorithm execution with respect to PWM period when `REP_RATE` is set to 3.

Figure 45. AD conversions for three shunt topology stator currents reading and PMSM FOC algorithm execution when `REP_RATE=3`



*Note: Because three shunt resistor topology requires low side switches to be on when performing current reading A/D conversions, the `REP_RATE` parameter must be an **odd** number in this case.*

Considering that the raw PMSM FOC algorithm execution time is about 25.7µs when in three shunt resistor stator current reading configuration, the related contribution to CPU load can be computed as follows:

$$CPU_{Load} = \frac{F_{PWM}}{RefreshRate} \times 25.7 \times 10^{-6} \times 100 = \frac{F_{PWM}}{(REPRATE + 1)/\ 2} \times 25.7 \times 10^{-6} \times 100$$

A.2 Selecting PWM frequency for 3 shunt resistor configuration

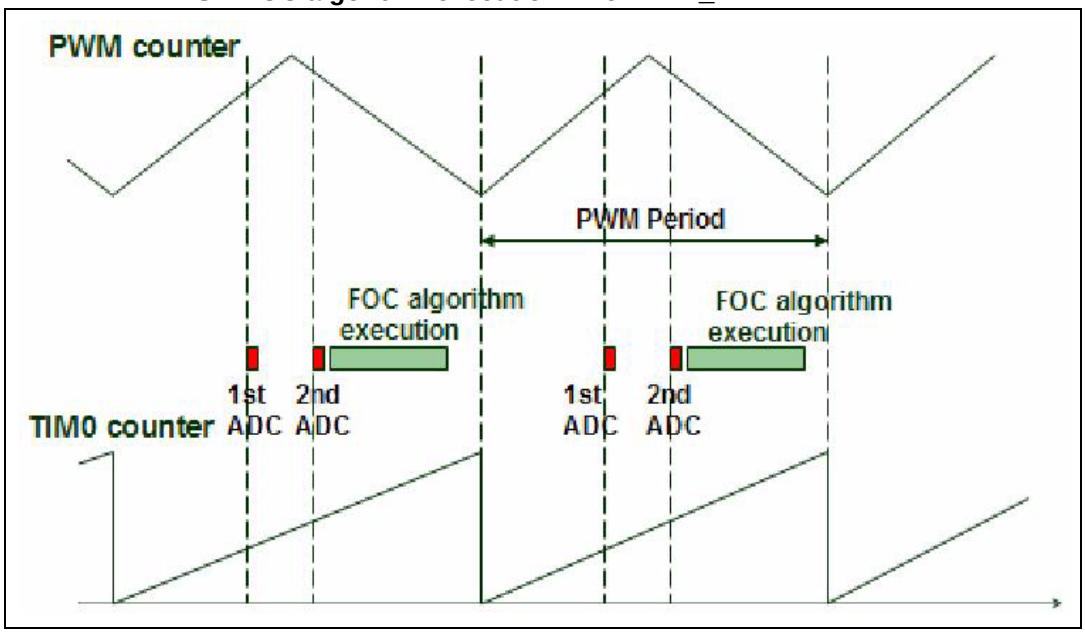
Beyond the well known trade-off between acoustical noise and power dissipation, consideration should be given to selecting the PWM switching frequency using the PMSM software library.

As discussed in *Section 4.2.4 on page 39*, depending on the PWM switching frequency, a limitation on the maximum applicable duty cycle could occur if using three shunt resistor configuration for current reading. *Table 3: PWM frequency vs maximum duty cycle on page 47*, summarizes the performance of the system when the software library is used in conjunction with the STR750-MCKIT hardware.

Note: The MB459 board is an evaluation platform; it is designed to support different motor driving topologies (PMSM and AC induction) and current reading strategies (single and three shunt resistors). Therefore, the figures given in Table 3 on page 47 should be understood as a starting point and not as a best case.

Moreover, to keep the synchronization between TIM0 and PWM peripherals, it is always necessary to finish executing the PMSM FOC algorithm before the next PWM period begins as shown in *Figure 46*.

Figure 46. FAD conversions for three shunt topology stator current readings and PMSM FOC algorithm execution when `REP_RATE=1`



Given that the raw execution time of the PMSM FOC algorithm is around 25.7µs and that other delays (such as the time necessary to enter ADC ISR) have to be considered, this limits to about 14.4 kHz the maximum PMSM FOC algorithm execution rate when using `REP_RATE =1`. However, no limitations occur in the typical range of PWM frequencies when using `REP_RATE=3`.

The following table summarizes the performance of the system for different PWM frequencies.

Table 5. System performance when using STR750-MCKIT

PWM frequency	Max applicable duty cycle	Max FOC algorithm execution rate
Up to 11.4kHz	100%	Equal to PWM frequency
12.2kHz	99.5%	
12.9kHz	99%	
13.7kHz	98.5%	
14.4kHz	98%	
15.2kHz	97.5%	Equal to PWM frequency/2 (REP_RATE=3)
16kHz	97%	
16.7kHz	96.5%	
17.5kHz	96%	

A.3 Fixed-point numerical representation

The PMSM software library uses fixed-point representation of fractional signed values. Thus, a number *n* is expressed as

$$n = m.f$$

where *m* is the integer part (magnitude) and *f* the fractional part, and both *m* and *f* have fixed numbers of digits.

In terms of two's complement binary representation, if a variable *n* requires QI bits to express - as powers of two - its magnitude (of which 1 bit is needed for the sign), QF bits – as inverse powers of two - for its fractional part, then we have to allocate QI + QF bits for that variable.

Therefore, given a choice of QI and QF, the variable representation has the following features:

- Range: $-2^{(QI-1)}$ $n < 2^{(QI-1)} - 2^{(-QF)}$;
- Resolution: $= 1 / 2^{QF}$.

The equation below converts a fractional quantity *q* to fixed-point representation *n*:

$$n = floor\left(q \cdot 2^{QF}\right)$$

A common way to express the choice that has been made is the "q QI.QF" notation.

So, if a variable is stored in q3.5 format, it means that 3 bits are reserved for the magnitude, 5 bits for the resolution; the expressible range is from -4 to 3.96875, the resolution is 0.03125, the bit weighting is:

bit n.	7	6	5	4	3	2	1	0
value	-4	2	1	1/2	1/4	1/8	1/16	1/32

This software library uses the PU (“Per Unit”) system to express current values. They are always referred to a base quantity that is the maximum measurable current I_{max} (which, for the proposed hardware, can be estimated approximately at $I_{max} = 0.6 / R_{shunt}$); so, the “per unit” current value is obtained by dividing the physical value by that base:

$$i_{PU} = \frac{i_{S.I.}}{I_{\max}}$$

In this way, i_{pu} is always in the range from -1 to +1. Therefore, the q1.15 format, which ranges from -1 to 0.999969482421875, with a resolution of 0.000030517578125, is perfectly suitable (taking care of the overflow value (-1)·(-1)=1) and thus extensively used.

Thus, the complete transformation equation from SI units is:

$$i_{q1.15} = floor\left(\frac{i_{S.I.}}{I_{MAX}} \cdot 2^{QF}\right)$$

A.4 Additional or up-to-date technical literature

More information can be found on the ST website (www.stmcu.com).

More specifically, the latest documents and software can be found directly at: http://www.stmcu.com/inchtml-pages-str750.html.

In addition, FAQ and Forums can be found directly at :
http://www.stmcu.com/forumsid-17.html for STR7 general enquiries.

http://www.stmcu.com/forumsid-13.html for motor control related enquiries.

A.5 References

[1] P. C. Krause, O. Wasynczuk, S. D. Sudhoff, Analysis of Electric Machinery and Drive Systems, Wiley-IEEE Press, 2002.

[2] T. A. Lipo and D. W. Novotny, Vector Control and Dynamics of AC Drives, Oxford University Press, 1996.

[3] P. Vas, Sensorless Vector and direct Torque Control, Oxford University Press, 1998.

7 Revision history

Table 6. Document revision history

Date	Revision	Changes
9-Feb-2006	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.

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