AN5166
Application note

Guidelines for control and customization of power boards with STM32 MC SDK v5.0

Introduction

This document is intended for users wishing to design their own application board to drive a Permanent-Magnet Synchronous Motor (PMSM) starting from an ST Motor Control (MC) board. To this end, it reviews the hardware, the firmware and the related motor control software tools.

For new projects users can start directly with the Motor Control PC software tools provided with the STM32 MC SDK v5.0 (X-CUBE-MCSDK and X-CUBE-MCSDK-FUL).

The following documents, available on *www.st.com*, are considered as references:

- AN2834: How to get the best ADC accuracy in STM32 microcontrollers
- UM2392: STM32 motor control SDK v5.0.0 firmware
- UM2380: STM32 motor control SDK v5.0 tools

Contents

List of tables

List of figures

1 General information

The STM32 MC SDK v5.0 is used for the development of motor-control applications running on STM32 32-bit microcontrollers based on Arm®(a) Cortex®-M processors.

Table 1 lists the acronyms relevant for a better understanding of this document.

Table 1. List of acronyms

Acronym	Description
API	Application programming interface
FOC	Field-oriented control
FW	Firmware
GUI	Graphical user interface
HAL	Hardware abstraction layer
HFI	High frequency injection
HW	Hardware
ICS	Isolated current sensor
IDE	Integrated development environment
LL	Low-level
MC	Motor control
MC WB	Motor control Workbench (STMicroelectronics software tool)
MP	Motor profiler (STMicroelectronics software tool)
MTPA	Maximum torque per Ampere
PFC	Power factor correction
PMSM	Permanent-magnet synchronous motor
SDK	Software development kit
SPL	Standard peripheral libraries
SW	Software

arm

a. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

2 Derivative design

The easiest way to design a new hardware board starting from an ST MC one, goes through the following steps:

1. Selection of the same 3-phase inverter topology
   a) Same STM32 MCU line (e.g. STM32F303xx)
   b) Same current sensing topology (e.g. 1-shunt topology)
   c) Similar motor driver (e.g. STSPIN230 as triple half-bridge motor driver)
   d) Similar power (e.g. X-NUCLEO-IHM11M1 as low voltage/low current)
2. Adaptation to the new hardware
   a) Resistor shunt(s) computation
   b) Resistor network(s) gain calculation
   c) External or internal operational amplifier(s) component(s) usage
   d) External or internal comparator(s) usage
   e) Computation of protection threshold source(s) for input comparator(s)
3. Modification of the STM32 Workbench project (PC software tools)
   a) Find the selected ST example
   b) Update with all the hardware adaptation
4. Modification from the STM32CubeMX project (PC software tools)
   c) When needed, change the MCU line with a compatible one
   d) Complete the application configuration

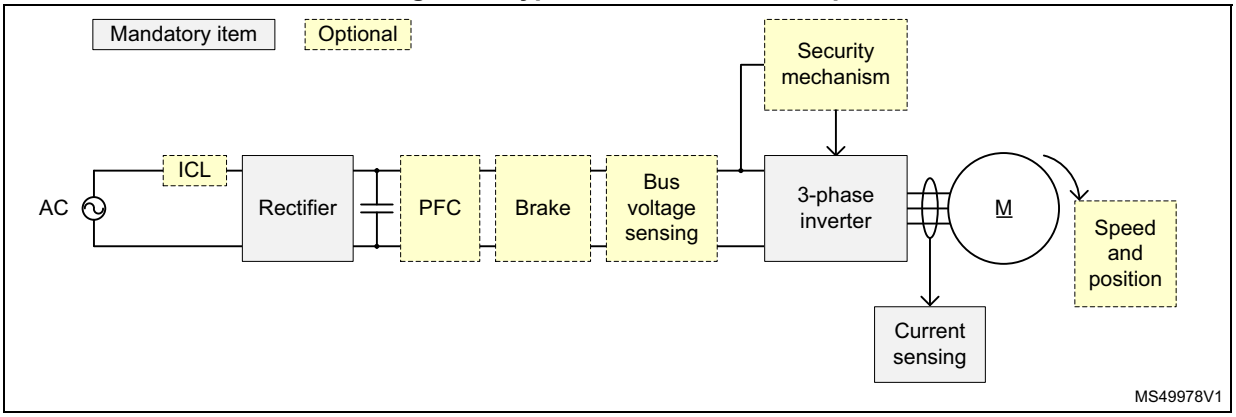
2.1 HW setup

To be effective, the control of a 3-phase motor relies on:

- Mandatory features:
  – A microcontroller to execute the FW
  – A 3-phase inverter to control motor voltage and current
  – A current sensing acquisition to feedback the regulation
- Optional features (only some of them discussed in this document):
  – An In-rush current limiter to reduce peak current during motor start-up
  – A Power Factor Correction (PFC) mechanism to reduce energy consumption
  – A brake mechanism to release motor energy
  – A bus voltage sensing to monitor the input voltage
  – A security mechanism to protect people and hardware
  – Acquisition of data from speed and position sensors for regulation feedback

Figure 1 shows a typical hardware setup (implemented on ST evaluation boards) to control a 3-phase PMSM.

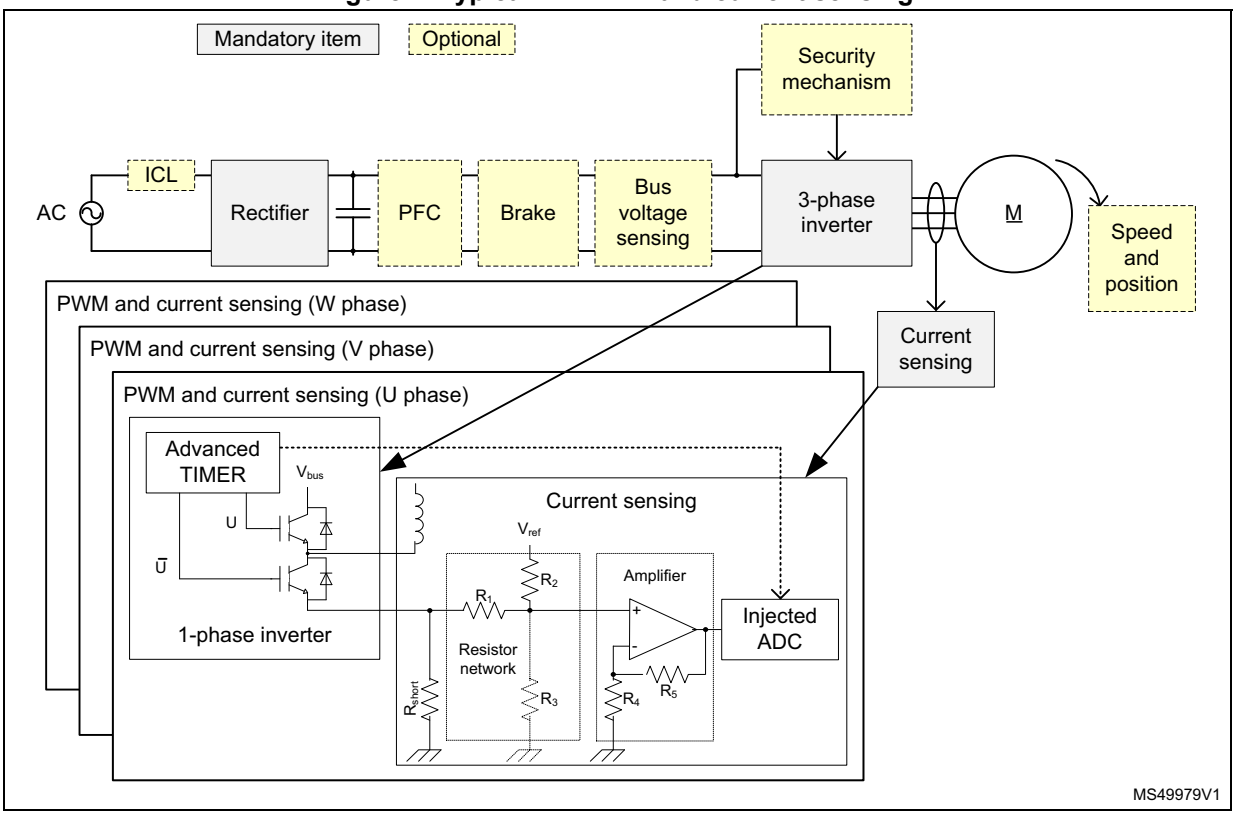
Figure 1. Typical MC hardware setup



2.1.1 3-phase inverter

The motor control subsystem uses one advanced timer with PWM digital outputs to drive the 3-phase inverter (see *Figure 2*).

Figure 2. Typical MC PWM and current sensing



On ST boards the power stage implements high-side and low-side switches to drive the three phases. Depending on the power components, and for each motor phase, the supported PWM digital outputs (or channels) are:

- High-side and low-side drive
  - Hardware activates the needed PWM digital outputs as well as its complementary signals.
  - User has to configure the PWM switching dead times to avoid short-circuits in the inverters, using the STM32 MC Workbench PC software tool.
- High-side drive only
  - Hardware enables the drivers and activates the needed PWM digital outputs.
  - User has to verify that the used hardware components manage the PWM switching dead time to avoid short circuits in the inverters.

From the firmware perspective, the configuration of the driver bridges is easily performed using the STM32 MC Workbench PC software. User configures the driver bridges dead time values and their active polarities, according to its HW implementation, through the dedicated Phase Driver blocks within the Power stage area.

Depending on the implemented current sensing topology, in addition to the used PWM channels there can be need for other timer channels to trigger the sample time for the ADC (see *Table 2*).

Table 2. Advanced timer usage

Supported MCUs	Topology		
	1-shunt	3-shunt	Dual ICS
STM32F030RC/R8 STM32F031C6 STM32F051R8/C8 STM32F072VB/RB STM32F1xx STM32F302VB/VC STM32F303VB/VC STM32F303ZE/VE/RE STM32F302R8 STM32F415ZG STM32F417IG STM32F407IG STM32F446ZE/RE	– PWM for each phase drive – 3 channels – Trigger for SVPWM efficiency – 1 channel – Trigger for ADC usage – 2 channels	– PWM for each phase drive – 3 channels – Trigger for ADC usage – 1 channel	– PWM for each phase drive – 3 channels – Trigger for ADC usage – 1 channel

2.1.2 Current sensing

For accuracy reason, current measurements are triggered from the advanced timer (thanks to its internal capability).

The motor control subsystem requires the usage of ADCs. Depending upon the used ADCs and current measurement topology, three implementations are supported.

- The 3-shunt current reading mode needs:
  - A resistor network (R_1, R_2 and R_3) to polarize the measured voltage (acting as the current consumption image of the motor) before its amplification via an operational

amplifier together with its resistor network (R_4 and R_5). This setup needs to be implemented for each phase (three times in total).

  - Two ADCs with two channels each, to measure the voltage at the same time, from the two enabled phases at this time (highest accuracy), or, alternatively one ADC with three channels to measure the voltage at two different times, from the two enabled phases (lower accuracy).

- The 1-shunt current reading mode needs:
  - A resistor network (R_1, R_2 and R_3) to polarize the measured voltage (acting as the current consumption image of the motor) before its amplification via an operational amplifier together with its resistors network (R_4 and R_5). This setup is needed only once.
  - One ADC with one channel to measure the voltage at the sampling time (twice per period), when one (or two) phase(s) is (are) active at that time.
- The dual ICS reading mode needs:
  - When needed, a resistor network (R_1, R_2 and R_3) to polarize the measured voltage (acting as the current consumption image of the phase) before its amplification via an operational amplifier, together with its resistor network (R_4 and R_5). This setup has to be implemented for at least two of the three phases.
  - Two ADCs with one channel each to measure the voltage (at the same sampling time) for the two hard-connected phases.

From the firmware perspective, going from one topology reading mode to another one is easily performed using the STM32 MC Workbench PC software. The user must adapt the resistors network, according to its HW implementation, through the current sensing block within the power stage area.

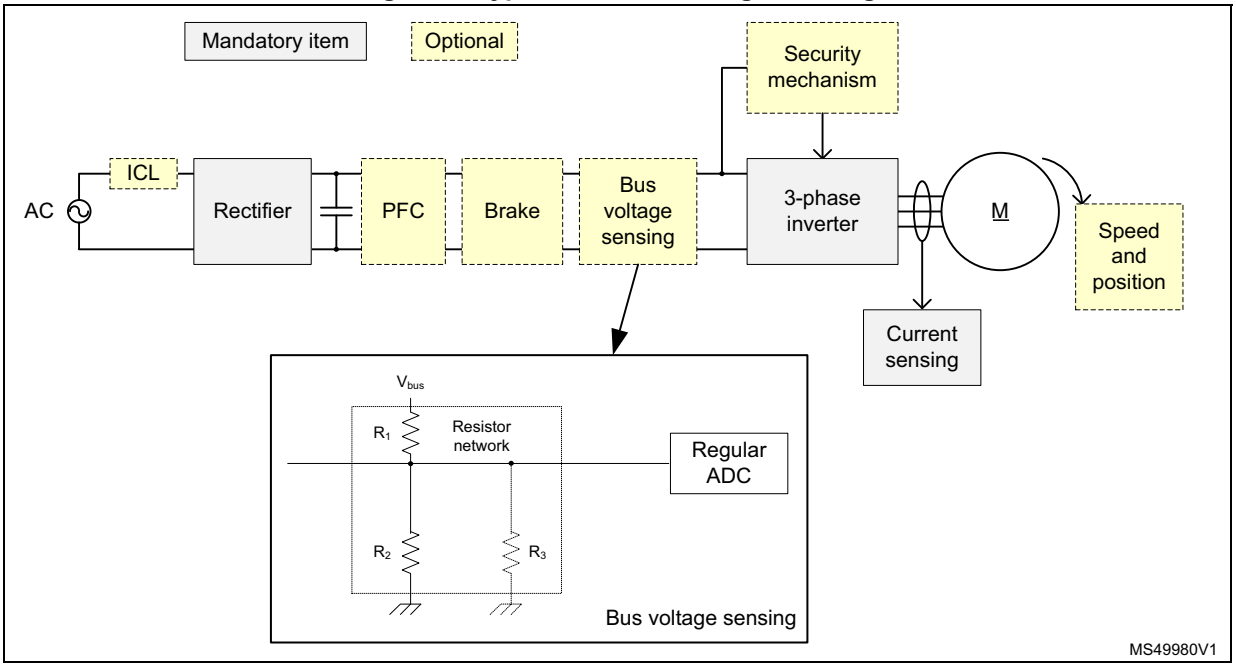
Table 3. Current sensing

Supported MCUs	Topology		
	1-shunt	3-shunt	Dual ICS
STM32F030RC/R8 STM32F031C6 STM32F051R8/C8 STM32F072VB/RB	ADC (x1) - 1 channel	ADC (x1) - 3 channels	N/A
STM32F1xx		ADC (x2) - 2 channels each	ADC (x2) - 1 channel each
STM32F302VB/VC STM32F303VB/VC STM32F303ZE/VE/RE			
STM32F302R8		ADC (x1) - 3 channels	N/A
STM32F415ZG STM32F417IG STM32F407IG STM32F446ZE/RE		ADC (x2) - 2 channels each	ADC (x2) - 1 channel each

2.1.3 Bus voltage sensing

A resistor network (R_1, R_2 and R_3) is implemented to adapt the ADC input range from the bus voltage DC range (see *Figure 3*).

Figure 3. Typical MC bus voltage sensing



From the firmware perspective, adapting the resistor values is easily performed using the STM32 MC Workbench PC software. User configures the resistor network according to its HW implementation, through the bus voltage sensing block within the power stage area.

The bus voltage is measured periodically using one ADC with one channel (regular conversion only). However, due to the MC firmware real-time constraints (current sensing acquisition priority), the regular conversion can be delayed. When this option is disabled, the MC firmware always uses the nominal rated bus voltage as a fixed value, defined within the STM32 MC Workbench PC software.

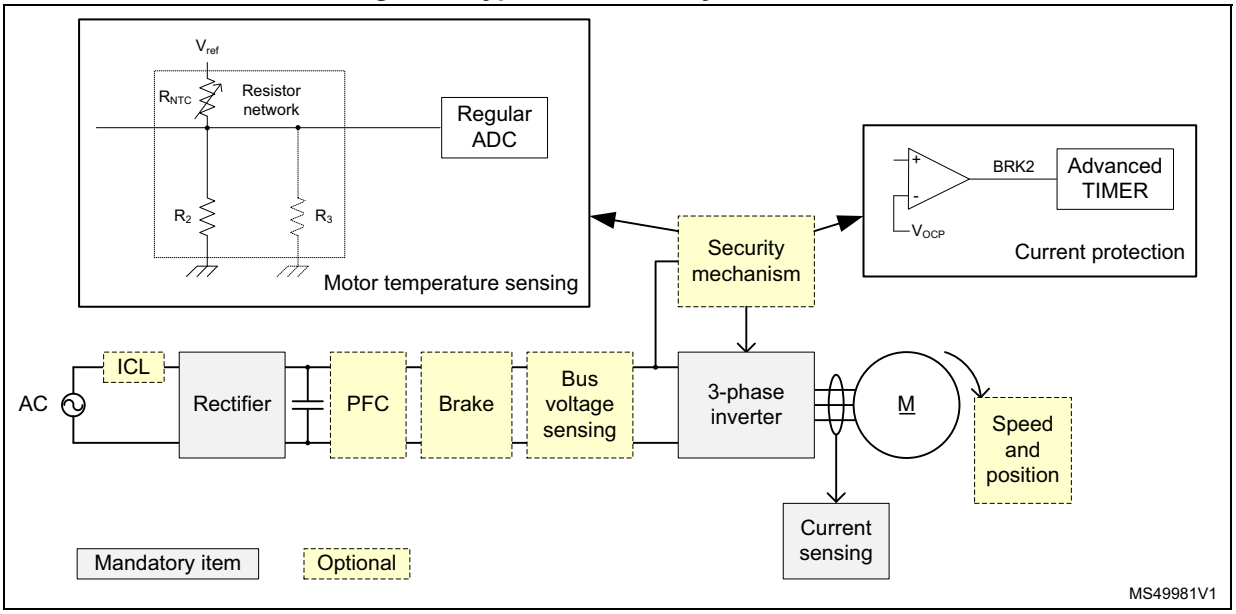
2.1.4 Security mechanisms

Some security mechanisms (see *Figure 4*) are implemented through hardware (thanks to the internal capabilities). The supported hardware security mechanisms are:

- Over-voltage protection
  Can be done by hardware, but is managed by the firmware (thanks to the bus voltage sensing acquisition).
- Over-current protection
  A reference voltage (MCU internal or external) is transformed in a reference current as the maximum upper limit, and then compared with the measured current. Then, the comparator output may trigger the Break or the Break2 input of the advanced timer to stop the PWM generation immediately.
- Motor temperature sensing
  A resistor network (R_{NTC}, R_2 and R_3) is used to polarize the measured voltage (acting

as temperature image of the electronic power drive). One ADC with one channel keeps measuring the voltage regularly.

Figure 4. Typical MC security mechanisms



From the firmware perspective, setting the over-current (or the motor temperature) protection threshold values is easily performed using the STM32 MC Workbench PC software. User configures these threshold values, according to its HW implementation, through the over-current protection block (or the temperature sensing block) within the power stage area.

2.1.5 MCU

The microcontroller is used to execute the MC firmware, to drive and to acquire signals from the motor.

When the design is derived from ST example boards, user may change only within the same MCU line (e.g. from STM32F030RC to STM32F030K6).

When changing the MCU the user has to carefully check availability of all the same peripherals (as described in previous paragraphs), namely:

- ADC(s) with its (theirs) available channels
- advanced timer
- operational amplifiers
- comparators
- clock frequencies.

Refer to *Section 2.3: STM32 MC Workbench* and *Section 2.4: STM32CubeMX* before changing the microcontroller.

2.2 Interactions between STM32 MC Workbench and STM32CubeMX

The purpose of the STM32 MC Workbench is to produce an *.ioc file from the information provided by users about their motor control application. This file is then processed by STM32CubeMX to generate a software project that can be directly open in the favorite IDE chosen by the user.

The STM32 MC Workbench stores two sets of data in the *.ioc file:

1. Configuration information related to the MCU chosen for the motor control application and the used peripherals (hardware IPs)
2. Pure motor control parameters and their values.

The first set allows the STM32CubeMX to generate the initialization code for the MCU and its peripherals. Thanks to this, all peripherals required by the motor control application are ready to be used when the motor control software subsystem starts initialization.

The second set provides the information needed by STM32CubeMX to generate the motor control cockpit and to choose the proper motor control source or library files to include in the generated project. Some of these data are also made accessible to the firmware in the generated source file. The firmware uses them for its own initialization and also during operation.

The two sets of data are not independent from one another. Values used by the firmware, as well as parameters needed to configure peripherals, may be linked. As a consequence, modifying the motor control parameters in the generated source files is hazardous and should be avoided as such modifications impact the MC firmware and not the related peripherals.

The *.ioc files generated by the STM32 MC Workbench can be further modified by users from STM32CubeMX. When doing so, the software project needs to be generated again, directly from the STM32CubeMX. However, note that the STM32 MC Workbench cannot fully read *.ioc files back, so any changes made to an *.ioc file from the STM32CubeMX may be overwritten.

This issue is mitigated by the “Update” feature of the STM32 MC Workbench. This feature lets the users choose whether they want to fully regenerate the *.ioc file or only update the pure motor control parameters (the second set of data). In the first case, all modifications made by the STM32CubeMX in the *.ioc file are lost. In the second case, modifications made with the STM32CubeMX are left intact and only the parts of the *.ioc file that deal with the second set of data (the pure motor control parameters) are replaced with the new parameter values. Refer to the referenced user manuals for detailed information on the “Update” feature.

The interactions between the STM32 MC Workbench and the STM32CubeMX have important consequences on the development of a motor control application. The following sections provide recommendations to exploit these interactions at best and to smooth the development flow.

2.3 STM32 MC Workbench

2.3.1 Modifying the motor control parameters

The STM32 MC Workbench is used to configure the hardware peripherals setup, as well as its firmware parameters, and to generate the motor control project through STM32CubeMX.

Note that the motor control parameters are mainly configured by the “Drive Management” stage within the STM32 MC Workbench.

The right method to modify motor control parameters consists in using the STM32 MC Workbench to make the modifications, and then to re-generate the project, using the “Update” feature.

2.3.2 Using another MCU

The STM32 MC Workbench supports a subset of the STM32 MCUs, mainly those that are present on Nucleo and Eval boards supported by the STM32 MC SDK v5.0.

Users can change the MCU used in its motor control application thanks to the import feature of the STM32CubeMX. This feature allows the user to import an existing *.ioc file in a new STM32CubeMX project in which only the MCU has been selected. This feature is subject to some restrictions (refer to UM1718 “STM32CubeMX for STM32 configuration and initialization C code generation”, available on *www.st.com*, for more information).

During the *.ioc file import, some warning or error messages may appear when STM32CubeMX doesn't know how to match some features with those of the new MCU (e.g. naming mismatch, or extended feature). These messages are very useful to adapt and to configure the new MCU to the HW implementation.

Also note that the use of this feature implies the creation of a new *.ioc file with no link with the one generated by the STM32 MC Workbench. The STM32 MC Workbench cannot be used to update directly this new *.ioc file.

2.3.3 Clock tree configuration

Clock configuration is a sensitive factor for the performance of a motor control software subsystem. The STM32 MC Workbench sets it to the optimal value from the motor control perspective, and it must not be changed by STM32CubeMX.

In particular, this applies to the core clock, but also to the clocks of the ADC(s) and to the timers used by the motor control software subsystem.

2.3.4 Handling interrupts

For performance reasons, the STM32 MC FoC firmware implements the handlers of the interrupts it uses. As a consequence, the STM32 MC Workbench prevents the STM32CubeMX from generating these handlers itself.

In addition, the STM32 MC Workbench selects these interrupts for initialization sequence ordering. The order in which interrupts are initialized is not important in itself, this feature ensures that the initialization of these interrupts will be performed last in the initialization sequence generated by the STM32CubeMX, after the motor control software subsystem.

In the last stage the STM32 MC Workbench configures the priority group feature of the NVIC peripheral as well as the priorities and sub priorities of the interrupts used by the

motor control subsystem. These three items must be left untouched in the *.ioc file. They must not be changed with the STM32CubeMX. Refer to UM2392 for more details on the handling of interrupts priorities in a motor control subsystem.

2.3.5 Changing a dedicated pin assignment

The STM32 MC Workbench generates an initial configuration for the various input and output pins used by the motor control application. Whenever possible, user selects pins assignment using the STM32 MC Workbench. However, if really needed, the pin assignment can be changed by users from the STM32CubeMX.

Some pins are given explicit names that start with M1_ or M2_ for, respectively, Motor 1 or Motor 2 related pins, and M1M2_ or M2M1_ for pins shared by both motors. These names are used internally by the motor control cockpit code for the initialization of the motor control subsystem. An example of motor control pin name is M1_PWM_VH, which refers to the PWM output for phase V of Motor 1.

To change the pin assignment users need to select a suitable alternate pin for the function they want to move, configure this new pin accordingly to the moved feature, and then transfer exactly the name from the previous pin to the new one.

Once a pin change has been made, any further modification done with the STM32 MC Workbench must be saved using the “Update” feature to preserve the new assignment within the *.ioc file.

2.4 STM32CubeMX

STM32CubeMX can be used to edit and modify the hardware peripherals configuration in the selected MCU, to adapt to the end user application.

However, users need to check that the configuration of a new peripheral does not change the clock tree parameters previously set by the STM32 MC Workbench for the motor control peripherals.

2.4.1 User project configuration

Aside from the hardware peripherals configuration, the STM32CubeMX allows the user to configure how its software project is generated. However, it is not recommended to change the project settings chosen from the STM32 MC Workbench in the generated *.ioc file.

2.4.2 Modifying the motor control parameters

Changing the configuration of peripherals already used by the motor control application is tricky and, if absolutely needed, must be carried out with great care. Two cases are distinguished here:

- Changing the values of parameters set by the STM32 MC Workbench.This may result in a malfunctioning application, for the same reasons highlighted in *Section 2.2: Interactions between STM32 MC Workbench and STM32CubeMX*.
- Using free channels of a motor control peripheral. This is not an issue, provided that there is no impact on the rest of the peripheral and especially on its general parameters. For instance, using additional channels of the timer used for motor control is possible as long as the Trigger Source, the Slave Mode and the Auto-reload register are left untouched. However, care must be taken when using free channels of the ADC

peripheral. The application can book them for regular conversions (not for injected conversions) for its own needs. However, it shall then use the API functions (refer to UM2392).

In both cases, once changes in peripherals configuration have been made, any further modification done to the project using the STM32 MC Workbench, has to be saved using its “Update” feature to preserve the pin configurations in the *.ioc file.

Some of the features offered by the STM32CubeMX cannot be used with the STM32 MC SDK v5.0. Refer to the release note delivered together with the software package for a complete list of these limitations.

2.4.3 Adding new pins or changing pin assignment

Configuring peripherals not used by the motor control application usually is not an issue, and can be done freely.

However, note that changes of motor control dedicated pins carried out outside the STM32 MC Workbench will be overwritten when using it back and updating the ioc file. The preferred way to change pin assignment is obviously through the STM32 MC Workbench.

3 Conclusion

To summarize, any derived end user application has to take care of the following items:

- Motor Control pin assignment
  - Assignment is done from the STM32 MC Workbench
  - Peripheral parameters are configured from the STM32 MC Workbench
- Bus voltage sensing
  - Compatibility with ADC input range
  - Maximization of the full ADC range usage
- Security mechanisms
  - Temperature sensing compatibility with ADC input range
  - Temperature sensing, thus maximizing the full ADC range usage
  - Over-voltage protection triggering from upper limit tuning
  - Over-current protection triggering from upper limit tuning
- 3-phase inverter:
  - Fill-in the power drivers dead-time from the STM32 MC Workbench
  - Fill-in the power drivers polarities from the STM32 MC Workbench
  - PWM digital outputs with either complementary signals or enabled signals
- Current sensing:
  - Fill-in the T-rise and T-noise parameters from the STM32 MC Workbench
  - Signal polarization not clamping the negative and positive waveform
  - Amplifier tuned to maximize the full ADC range usage
- End user application (STM32CubeMX usage):
  - Additional pin assignment
  - ADC regular channel usage only, cannot be Injected channel usage
  - Clock tree configuration left unchanged with the provided MC firmware one
  - Additional interrupt handling and priority usage not pre-empting the MC ones
  - Import STMCX file (generated from STM32 MC Workbench) before changing the MCU selection, if needed.

4 Revision history

Table 4. Document revision history

Date	Revision	Changes
04-Jun-2018	1	Initial release.