Getting started with the STSW-SPIN32F060x firmware package

Introduction

The STSW-SPIN32F060x firmware package in association with the EVSPIN32F0601S1/602S1 evaluation boards allows control of a 3-phase permanent magnet (PMSM) or a brushless DC motor (BLDC) with a six-step (trapezoidal) control algorithm.

The EVSPIN32F0601S1/602S1 boards are designed around the STSPIN32F0601/602 systems-in-package with embedded three-phase motor controller and STM32F031 microcontroller, implementing a single-shunt topology and either sensor-less or Hall effect sensor control.

1 Acronyms and abbreviations

Table 1. Acronyms and abbreviation

Acronym	Description
IPC	Industrial & Power Conversion division
MDG	Microcontrollers & Digital ICs Group
HW	Hardware
IDE	Integrated Development Environment
MCU	Micro-Controller Unit
GPIO	General Purpose Input Output
ADC	Analog to Digital Converter
VM	Voltage Mode
CM	Current Mode
SL	Sensors-Less
HS	Hall Sensors
SC	Sense Comparators
BEMF	Back Electro Motive Force
FW	Firmware
LF	Low Frequency
MF	Medium Frequency
HF	High Frequency
ZC	Zero-Crossing

2 STSW-SPIN32F060x firmware package overview

2.1 Code architecture

The firmware package is designed to support the EVSPIN32F0601S1/602S1 boards, but can be adapted for other boards mounted with the STSPIN32F0601 SIP or variant.

The Six-step driving library described herein is designated “6Step Lib” for the sake of simplicity.

The document describes how to configure this brushless scalar firmware library for the STSPIN32F0601/602 3-phase controller and drive a BLDC motor in both open and closed loop.

The 6Step Lib supports four different modes of control:

1. Sensor-less voltage mode
2. Sensor-less current mode
3. Voltage mode with Hall effect sensors feedback,
4. Current mode with Hall effect sensors feedback.

In sensor-less mode, the rotor position is determined from the detection of the BEMF zero-crossing point.

The firmware library and the example are written in C programming language and uses either the STM32Cube HAL embedded abstraction-layer software or optimized access to the STM32F031 resources. A prerequisite for using this library is basic knowledge of C programming, 3-phase motor drives and power inverter hardware. In-depth knowledge of STM32 functions is only required for customizing existing modules and for adding new ones in comprehensive application development.

The IDE tool supported is IAR Workbench.

Drivers abstract low-level hardware information so the middleware components and applications can fully manage the STSPIN32F0601/602 through a complete set of APIs which send commands to the motor driver in a hardware-independent manner. The package includes an application example for the EVSPIN32F0601S1/602S1 board to drive a high voltage three-phase BLDC/PMSM motor and several Motor Control parameters files to be used directly with their corresponding motor or as templates for similar ones.

The code architecture is based on the STM32Cube Ecosystem. The 6-step firmware package provides several motor control project examples with a design similar to an STM32Cube Expansion Package. It uses the STM32Cube MCU packages corresponding to the MCU of the control or inverter board. Each project was originally developed using the STM32CubeMX configuration tool.

The 6step firmware package consists of a main STSW-SPIN32F0601S1 folder and three subfolders: Drivers, Middlewares and Projects, which contains a motor control example using the EVSPIN32F0601S1 board.

2.2 Drivers

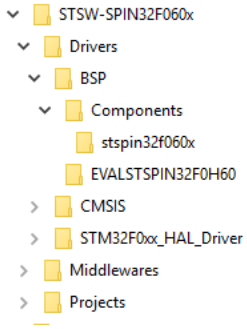
This folder contains the source code for the STSPIN32F0601/602 series components, the EVALSTSPIN32F0H60 board (generic name of the series of boards which includes EVSPIN32F0601S1/602S1), STM32Cube HAL and CMSIS drivers.

The STM32Cube HAL and CMSIS drivers generated by the STM32CubeMX tool have been copied into the Drivers folder.

The interface file for the component drivers contains structures that are specific to each component. The hardware resource mapping is handled by the customization of the 6step middleware configuration templates and the mapping from the STM32CubeMx project.

The interface file for the board drivers contains definitions specific to each board, such as resistor values, and ADC channel redefinition associated with board functions, such as reading a voltage on a potentiometer to set a speed command.

Figure 1. **Drivers folder content**



2.3 Middleware

This folder contains the core code of the motor control algorithms and the code for the different user interfaces. The code is split into files corresponding to combinations of the sensing method, control mode, speed loop feature and set point ramping feature (`6step_sl_vm_spdlp_spr.c`). There is also a set of dedicated files for the serial interface: `6step_com.c`, `6step_com_sl.c`, `6step_com_hs.c`, and `6step_com_sc.c`. The code for the other available interfaces is embedded in `6step_sl_vm_spdlp_spr.c`.

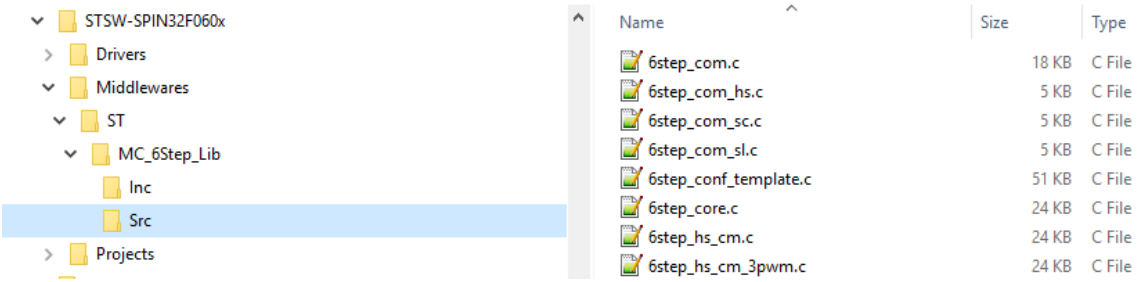
The middleware has template source and header files that need to be copied and renamed without the "_template" string in the user project. These files are then modified according to custom application requirements.

The `6step_conf_template.c` file contains:

- HAL/LL call-back functions which are implemented with middleware functions.
- HAL/LL driver interface functions which are functions defined in the middleware and implemented with HAL/LL functions.

The `6step_conf_<sensing method>_<drive mode>_template.h` and `6step_conf_<sensing mode>_<drive mode>_spdlp_template.h` middleware template files contain the definitions for the control parameters of the motor application example.

Figure 2. **Middlewares folder content**

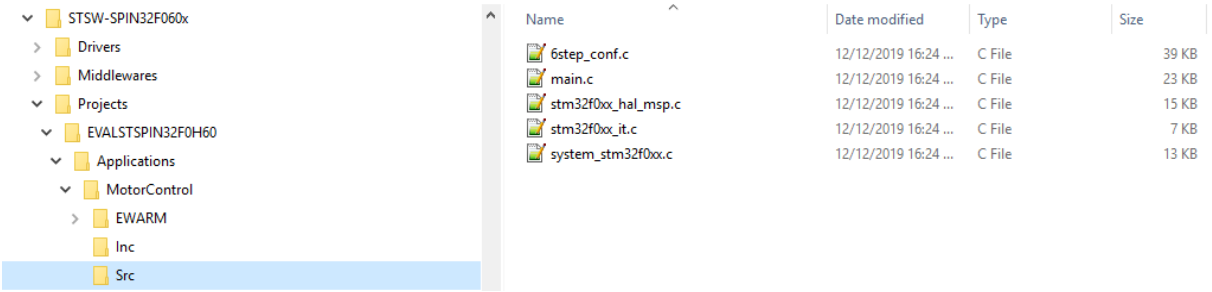


2.4 Projects

The Projects folder relates to the inverter board hardware configuration. The current project was implemented on IAR Embedded Workbench version 8.30.1 from IAR Systems, and should also be compatible with any subsequent versions of the IDE.

The STM32CubeMX tool was used to configure the relevant peripherals for the motor control application. All hardware resources were mapped with this tool. The ioc file and the src and inc folders generated by this tool were copied into the Projects\EVALSTSPIN32F0H60\Applications\MotorControl folder.

Figure 3. **Projects folder content**



The Src folder contains the following files:

- `6step_conf.c`: 6step middleware configuration file in which functions can be customized to adapt the 6step middleware to specific hardware.
- `main.c`: main file which calls the functions to initialize the MCU peripherals such as timers, ADCs, and GPIOs, the functions to configure the middleware according to board characteristics, the function to initialize the middleware, and the function to configure the communication interface according to user preferences. The `6step_core.h` header file include connects the user level with the motor middleware and all available APIs.
- `stm32f0xx_hal_msp.c`: the standard ST Cube HAL file for MCU peripheral initialization and de-initialization.
- `stm32f0xx_it.c`: the ST Cube HAL file for MCU interrupt request and handling function. The interrupt handler implementation is configured in the `6step_conf.c` file.
- `system_stm32f0xx.c`: the standard CMSIS system source file.

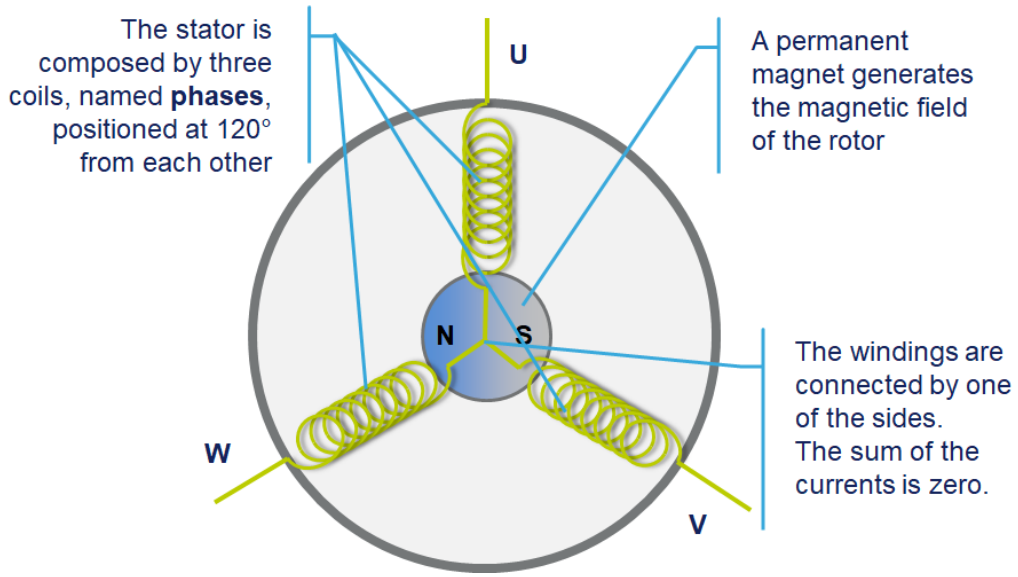
The Inc folder contains the following files:

- `6step_conf.h`: selects the sensing method, control mode and other definitions to build the 6-step middleware. This is the user implementation of the `6step_conf_template.h` middleware template.
- `6step_conf_<sensing method>_<drive mode>.h, 6step_conf_<sensing method>_<drive mode>_spdlp.h`: these files are user implementations of the corresponding _template.h middleware templates, with the definitions for the control parameters of the user motor application.
- `EVALSTSPIN32F0H60_conf.h`: this is the user implementation of the corresponding Drivers BSP _template.h file. It must be modified to include header files pertaining to the user choice of control board or inverter board MCU. It can be also modified to include user modifications to the power or inverter board, such as changed component values.
- `main.h`: this file contains definitions created by STM32CubeMx for MCU peripheral initialization.
- `stm32f0xx_hal_conf.h, stm32 f0xx_it.h`: these header files are generated by STM32CubeMx.

3 Brushless DC motor basics

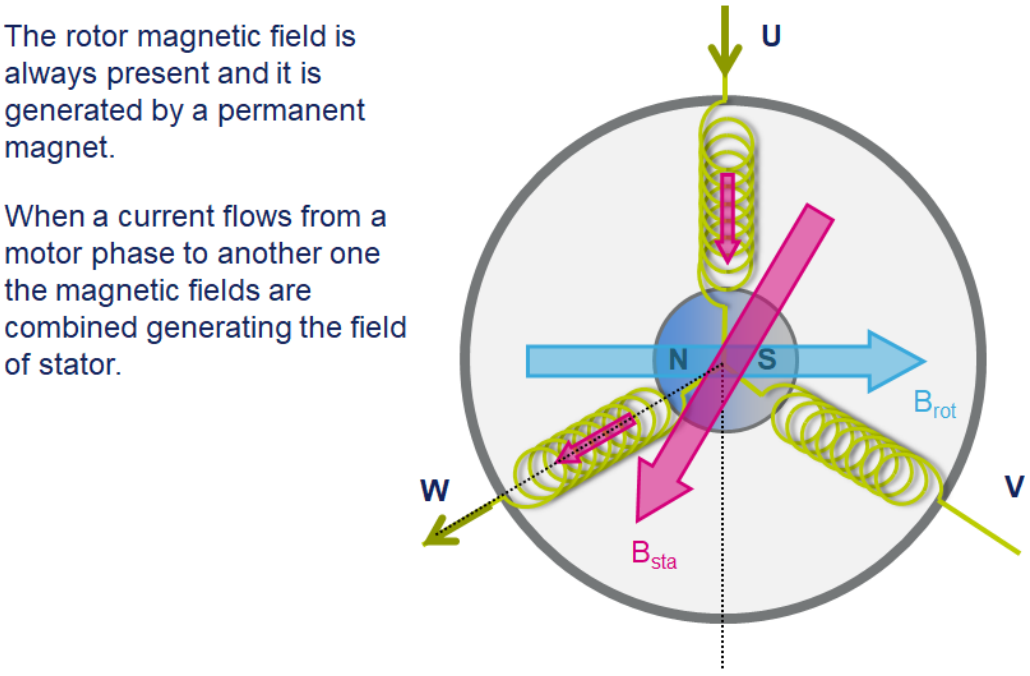
A brushless three phase motor consists of a fixed stator made of a set of three windings, and a mobile rotor containing an internal permanent magnet. The rotor may have several pole pairs distributed around the stator.

Figure 4. Motor stator and rotor arrangement



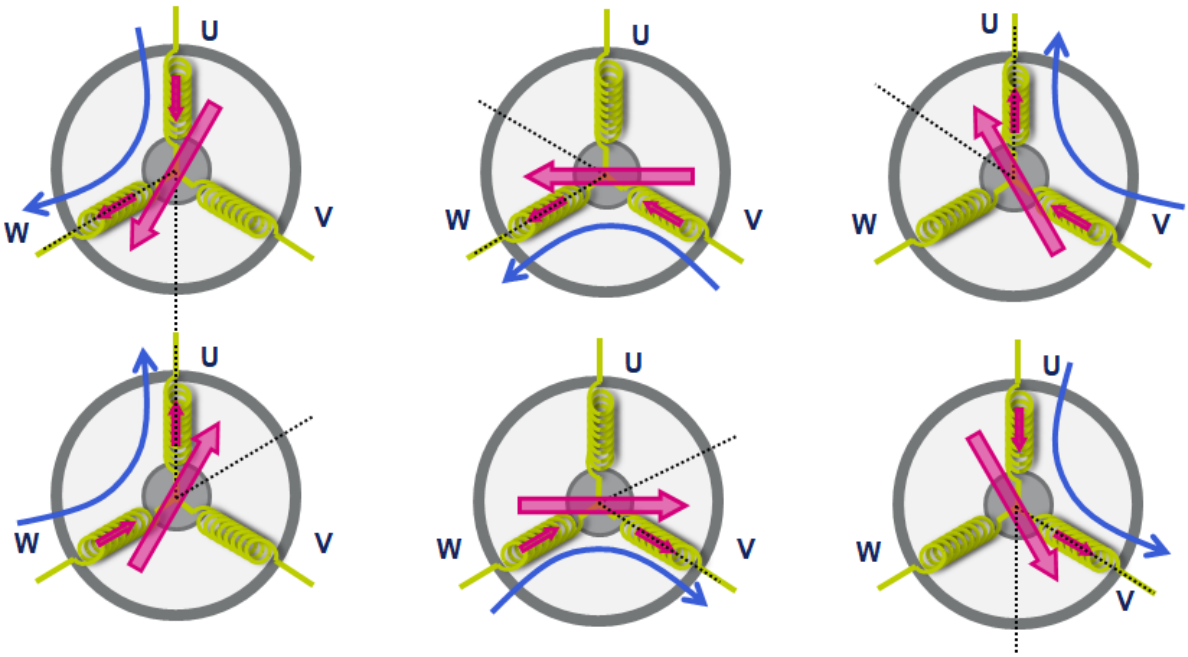
In six-step driving, the electrical cycle is divided into six commutation steps. For each step, the bus voltage is applied to one of the three phase windings of the motor, while the ground is applied to a second winding. This creates a current flow in these two windings and generates a stator magnetic field. The third winding remains open.

Figure 5. Motor stator and rotor magnetic fields



The successive steps are executed in the same way, except that the bus voltage and the ground are applied to different motor phase windings to generate a rotating stator magnetic field with six different discrete positions.

Figure 6. Motor stator magnetic fields discrete positions



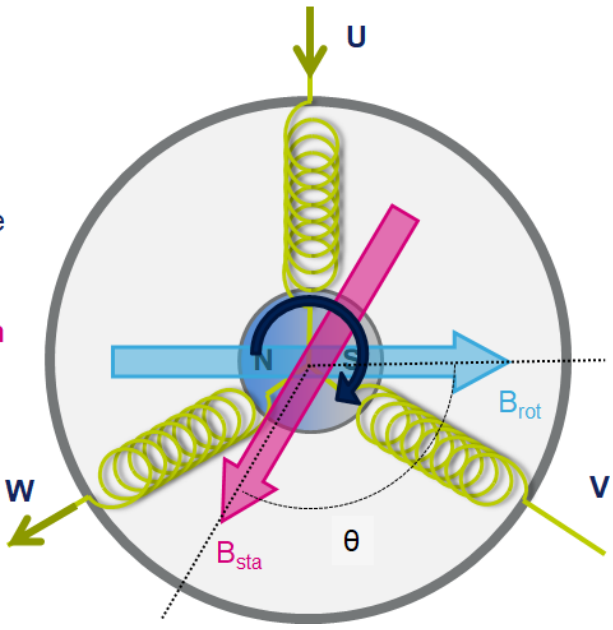
As the rotor has a permanent magnetic field, the rotating stator magnetic field creates a torque that moves the rotor. The maximum torque is obtained when the electrical angle between the rotor and the stator is 90°, so the step commutation is timed so that the torque is always close to 90°.

Figure 7. Motor torque

The torque applied to the motor is proportional to the sine of the load angle (θ).

When the rotor magnetic field approaches the stator one, the torque is reduced.

In order to keep the motor in motion it is necessary to change the direction of the stator magnetic field.



4 6-step firmware algorithms

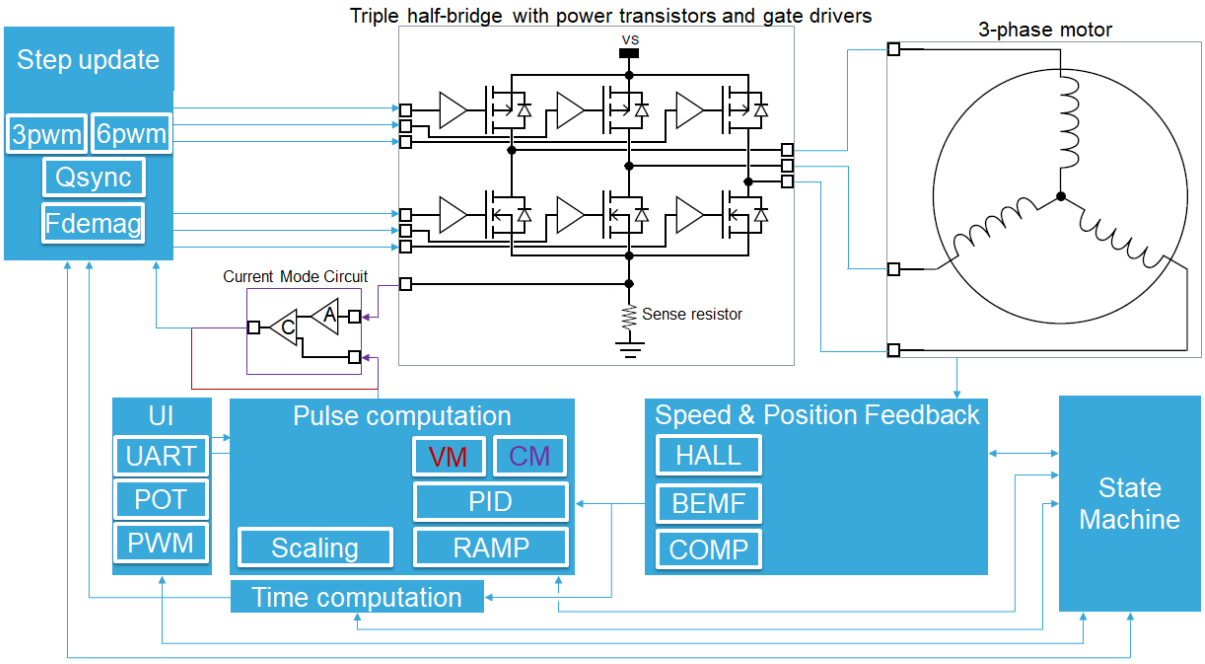
4.1 Overview

The 6-step firmware senses the position of the motor rotor with an electrical 60° accuracy and uses this information to compute the timing for the next step commutation and a new duty cycle for the PWM signals the control the amount of current pushed into the motor via a triple half bridge with power transistors. The user interface can issue a command to directly or indirectly change the duty cycle of these PWM signals.

4.1.1 Components

The 6-step firmware can be viewed as a set of components running under different tasks and interacting with each other and with the motor hardware and electronic circuitry. The different components are denoted by the blue boxes in the following figure.

Figure 8. High level architecture block diagram



There are different implementations of a component depending on the 6-step firmware setup. The functionality of each component is described below:

Speed and Position Feedback component

either BEMF for sensors-less algorithm or HALL for hall sensors algorithm or COMP for sense comparators algorithm is used

Pulse computation component

either current mode (CM) or voltage mode (VM) is used. A proportional-integral-derivative (PID) controller algorithm is used when the speed loop control feature is present. A setpoint ramping (RAMP) algorithm is used when the set point ramping feature is present

time computation component

user interface component

either a serial communication (UART) or a potentiometer voltage (POT) or a PWM pulse duration (PWM) is used to control the motor

step update component

there is the capability to use either Three-PWM (3pwm) or Six-PWM (6pwm) driving, anyway the EVSPIN32F0601S1/602S1 boards are driven in 6pwm mode only. Either synchronous or quasi-synchronous (Qsync) is used. Either normal or fast demagnetization (Fdemag) is used

state machine component

each component operates according to the state machine status and is also able to change the status

4.1.2 Tasks

The interrupt-based 6-step firmware runs several tasks using the various firmware components. Not all the tasks are applied in a given 6-step firmware setup.

All the tasks fall into one of the following categories:

- Zero-crossing (ZC) task
- High frequency (HF) task
- Medium frequency (MF) task
- Low frequency (LF) task
- Communication task
- Background task

These tasks are prioritized according to the 6-step firmware setup. The exact role of each task is described under each different 6-step sensing algorithm.

4.2 Driving modes

4.2.1 Voltage driving mode

The current injected into the motor is controlled by setting the duty cycle of the PWMs connected to the motor phases. The Current Mode circuit and the sense resistor shown in Figure 8. High level architecture block diagram are not used.

4.2.2 Current driving mode

The Current Mode circuit and the sense resistor shown in Figure 8. High level architecture block diagram are used. The Current Mode circuit has an amplifier A and a comparator C.

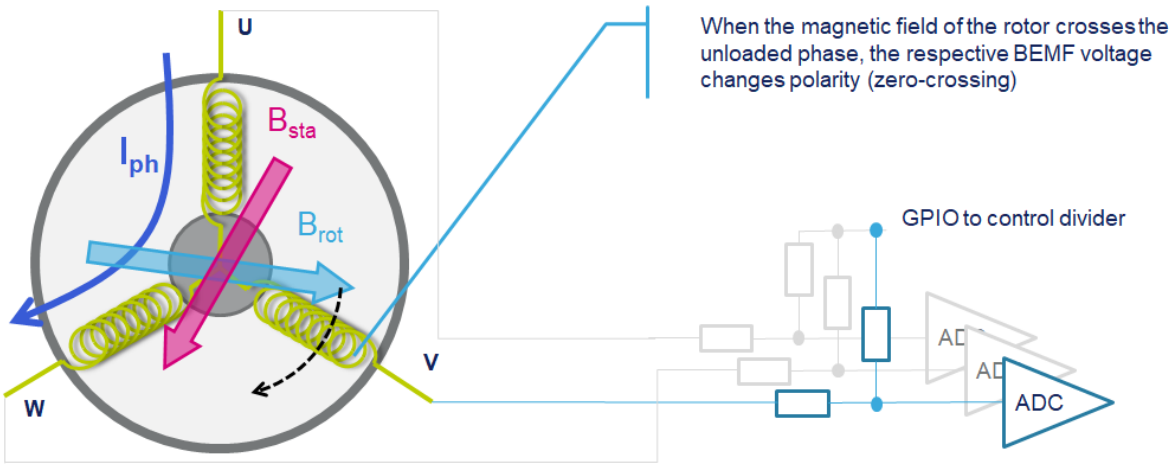
The current injected into the motor is controlled by setting the duty cycle of a reference PWM used to build a reference voltage. Using the comparator, this reference voltage is compared with an amplified sense resistor voltage which is proportional to the current circulating into the motor. This sense resistor is often designated as a shunt resistor. It is connected between the source of the lower side power transistor of each half-bridge and the ground.

The output of the comparator is used to switch off the PWMs connected to the motor phases when the amplified sense resistor voltage is greater than the reference voltage. If the amplified sense resistor voltage drops below the reference voltage, the PWMs connected to the motor phases are switched on during their next period.

4.3 Sensors-less algorithm

Sensors-less mode is based on sensing the Back Electro Magnetic Force (BEMF) zero-crossing on each motor coil by measuring the voltage on the floating phase using an ADC. The BEMF voltage is scaled at ADC input by a resistor bridge controlled by a GPIO. When the GPIO output is low, the resistor bridge is in operation and divides the voltage coming from the motor phase.

Figure 9. **Motor with sensors-less circuit**



In the 6-step firmware HF task which is run under the ADC call-back, the zero crossing is assessed through two possible thresholds related to whether the BEMF voltage when it is rising or falling. When measured during the HF PWM off time, each threshold is slightly above zero to ensure some noise protection. When measured during the HF PWM on time, each threshold must take the divided contribution of the power supply voltage into account.

When a motor phase becomes floating, its inductor must be demagnetized before assessing the BEMF voltage on this phase to avoid measuring the combination of the BEMF voltage and the demagnetization voltage from the demagnetization current. As there is no current discontinuity in an inductor and as there was current flowing just before disconnecting the power supply, the current on the new floating phase is not instantaneously zero and decreases down to zero at a rate that depends on the circuit connections to ground or supply through either the power transistor on the resistor or the power transistor body diode.

When this current is zero, the BEMF voltage can be assessed. The 6-step firmware allows assessment of the BEMF voltage after a demagnetization time set by the user in tenths of percentage of step time. This value must be lower than 400 (40% of a step), as the zero-crossing detection should occur around half a step. A good starting point for this value is 250 (a quarter of the step time).

When a BEMF voltage zero-crossing is detected, the firmware re-programs an MCU timer designated as an LF timer with a period designed to expire half a step time (or lower) after the zero-crossing detection. Then in the LF task, running under the LF timer call-back, the 6-step firmware executes the code to apply the next step to the motor. The time between the zero-crossing detection and the step commutation is adjustable.

The MF task running under the system tick call-back is used to interact with the user interface and to update the duty cycle controlling the amount of current pushed into the motor.

The BEMF voltage increases with speed; there is no BEMF when the motor is stopped and the BEMF measured voltage is not reliable when the speed is too low. To overcome these issues at start, the firmware first pushes current in the motor phases to set the rotor to a known position. This is the alignment state where the HF timer is started. The firmware then enters the start-up state where it starts the LF timer and consequently moves the motor and modifies the steps in order to achieve the speed linear acceleration. When the motor reaches a speed that can produce a reliable BEMF voltage, the firmware enters a validation state and, when it determines that the BEMF is reliable, it commutes the step according to BEMF zero crossing detection and the motor runs normally until a stop command is issued or a fault occurs.

4.3.1 Sensor-less MCU hardware resource requirements and usage

- An ADC is used with three different channels to measure the BEMF on the three motor phases and other channels for user measurement.
  - It must be possible to trigger the ADC by a timer trigger output pulse falling or rising edge.
  - The triggering time is expressed as a time in the HF timer period.
  - The timer that triggers the ADC may be the HF timer or a timer synchronized with the HF timer.
- One timer with four channels to generate three PWM signals and one signal to trigger the ADC.
  - Each PWM channel shall have its complementary counterpart.
  - This timer is designated the High Frequency (HF) timer.

- A timer is used to generate a time base for step commutation.
  - This timer is designated as the Low Frequency (LF) timer.
- In case of current mode, another timer is required to generate the PWM for the reference voltage.
  - This timer is designated as the Reference (REF) timer.
- In case of voltage mode, if the HF timer cannot directly trigger the ADC, a further timer triggered by the LF timer is used to trigger the ADC.
  - This timer is also designated as the REF timer.
  - In this mode, the LF timer is the master of the HF timer and the REF timer to these timers are synchronized and have the same counter values.
  - Both HF and REF timer must be programmed with the same pre-scaler.
  - The triggering resolution of the ADC is a counter cycle of the timer.

4.3.2 Sensor-less tasks

4.3.2.1 Sensor-less high frequency task

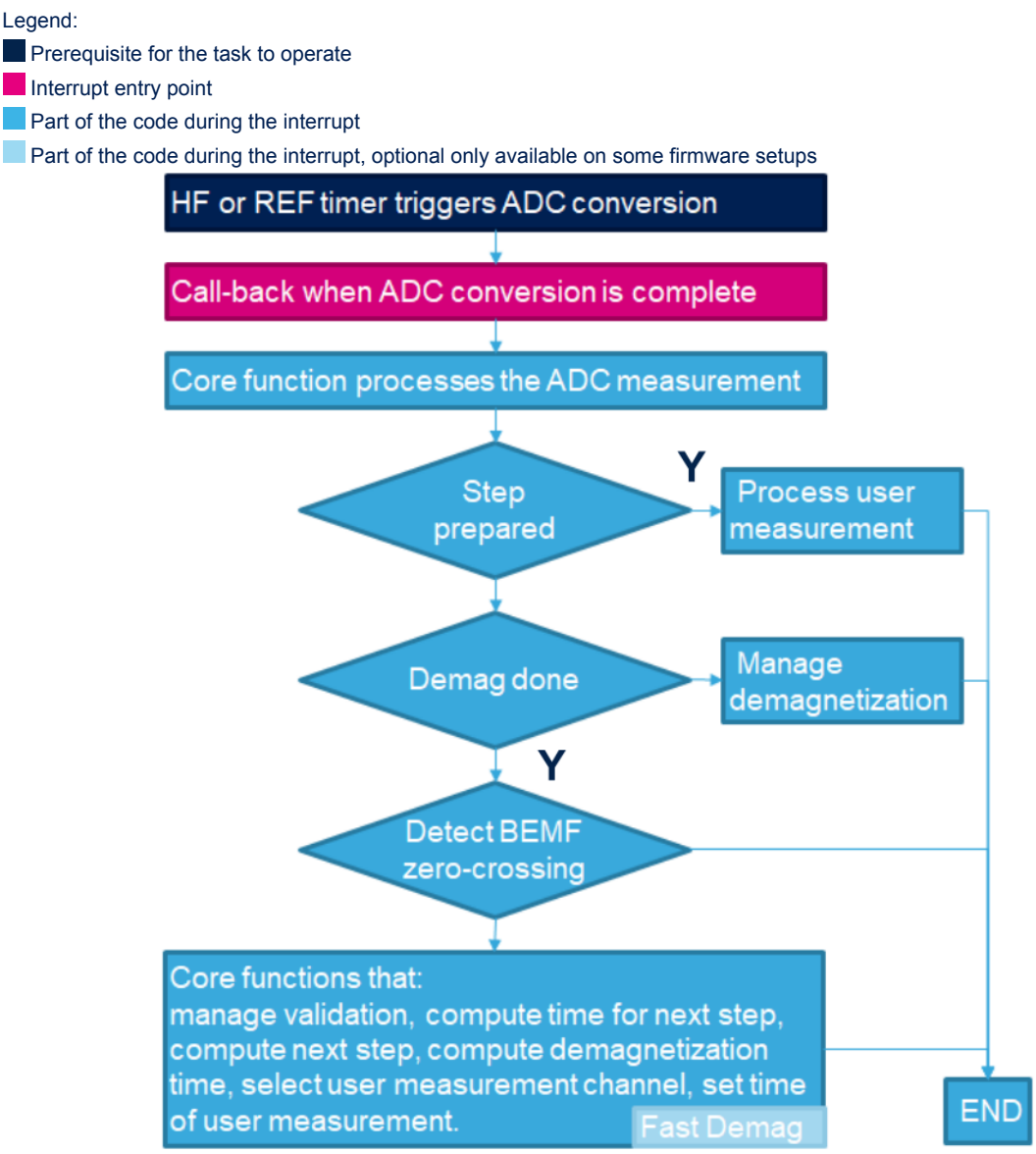
This high priority (0) task initiates when the ADC is triggered by a timer. When the ADC conversion is complete, a call-back calls a 6-step core function to process the ADC BEMF or user measurement.

The core function begins by evaluating is a Step has been prepared: if no, it is a BEMF measurement; if yes, it is a user measurement.

If it is a BEMF measurement, the task evaluates whether demagnetization has been performed: If not, it proceeds with demagnetization.

The task then proceeds to detect BEMF zero-crossing: if a BEMF zero-crossing is detected, the task calls several functions to manage the transition from the `MC_VALIDATION` to the `MC_RUN` state in order to compute the time for the next step, to determine the next step, to compute the demagnetization time, to select the user measurement channel, and to set user measurement time.

Figure 10. **Sensors-less high frequency task call graph**



4.3.2.2 *Sensor-less medium frequency task*

This lowest priority (3) task occurs every 1 ms and manages the duty cycle (hence the pulse) of the HF PWMs or REF PWM every control loop time milliseconds.

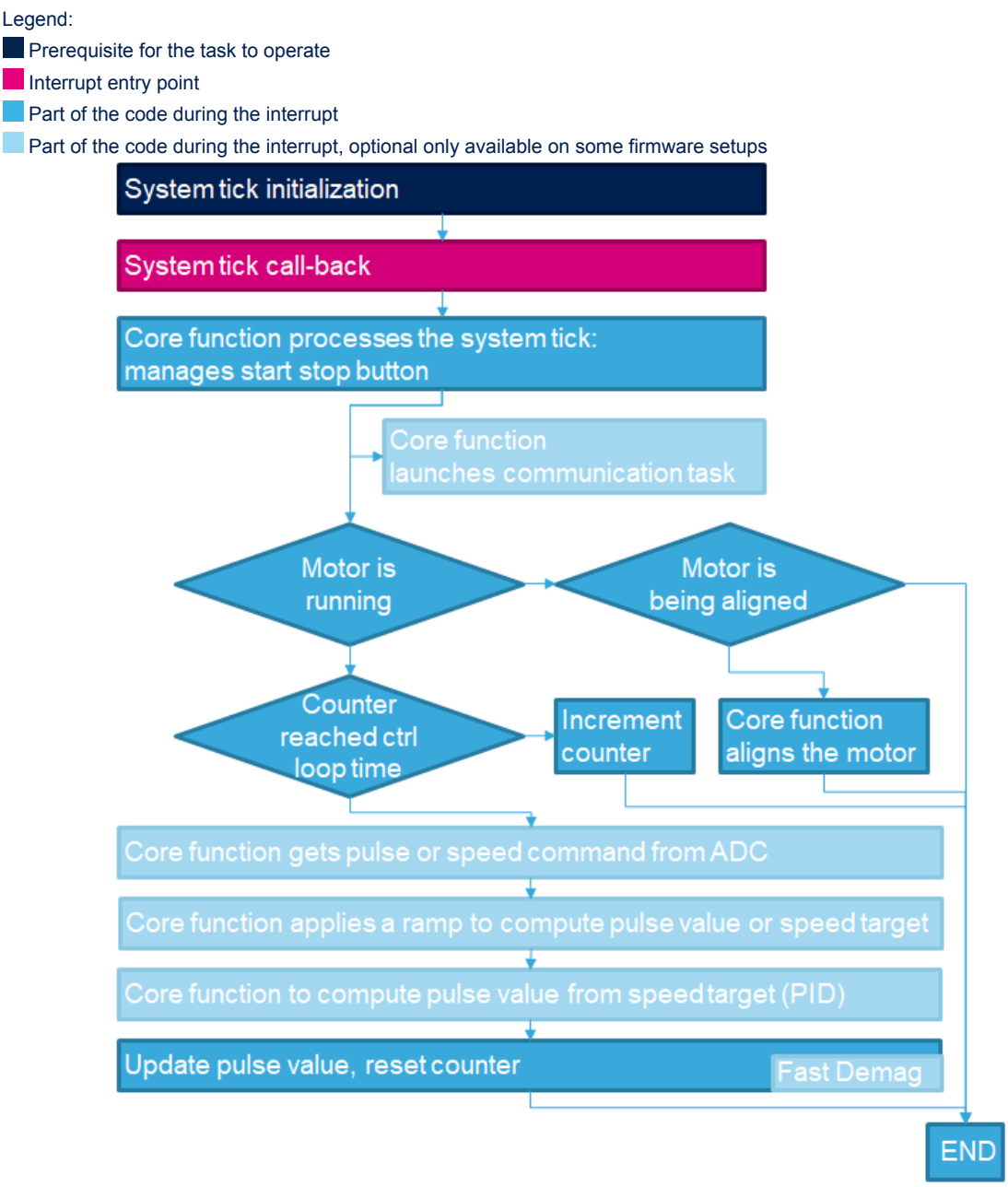
If the firmware includes the UART component, the task launches the serial communication task.

If the firmware includes the POT component, the medium frequency task uses the ADC to infer a pulse or speed command.

If the firmware includes the PID component, the medium frequency task manages the speed loop by calling a function with a PID regulator.

The light blue boxes are optional operations that depend on whether they are included in the firmware setup.

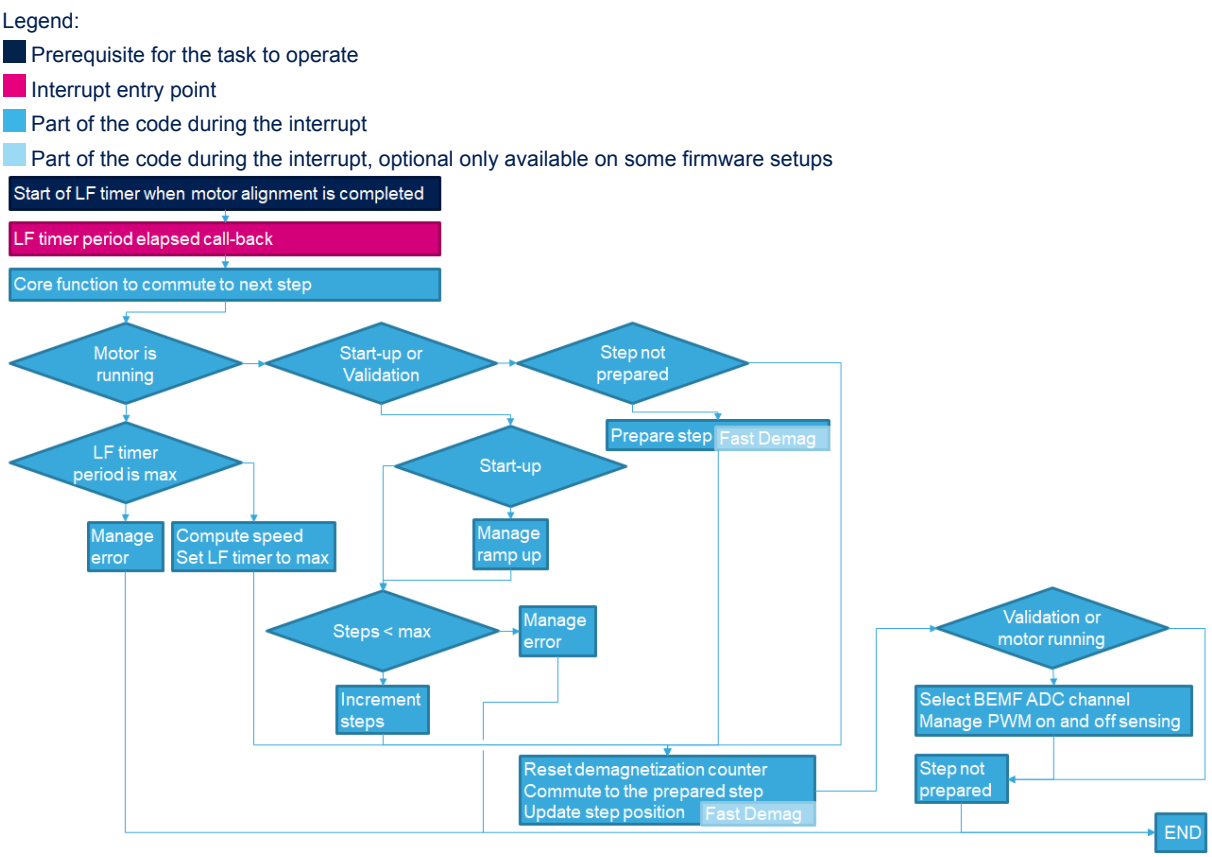
Figure 11. Sensors-Less medium frequency task call graph



4.3.2.3 *Sensor-less low frequency task*

This task is medium priority (1). The frequency of this task depends on the speed of the motor and runs under an LF timer interrupt. It commutes to a new step in the 6step sequence and computes the filtered speed feedback.

Figure 12. Sensors-Less low frequency task call graph



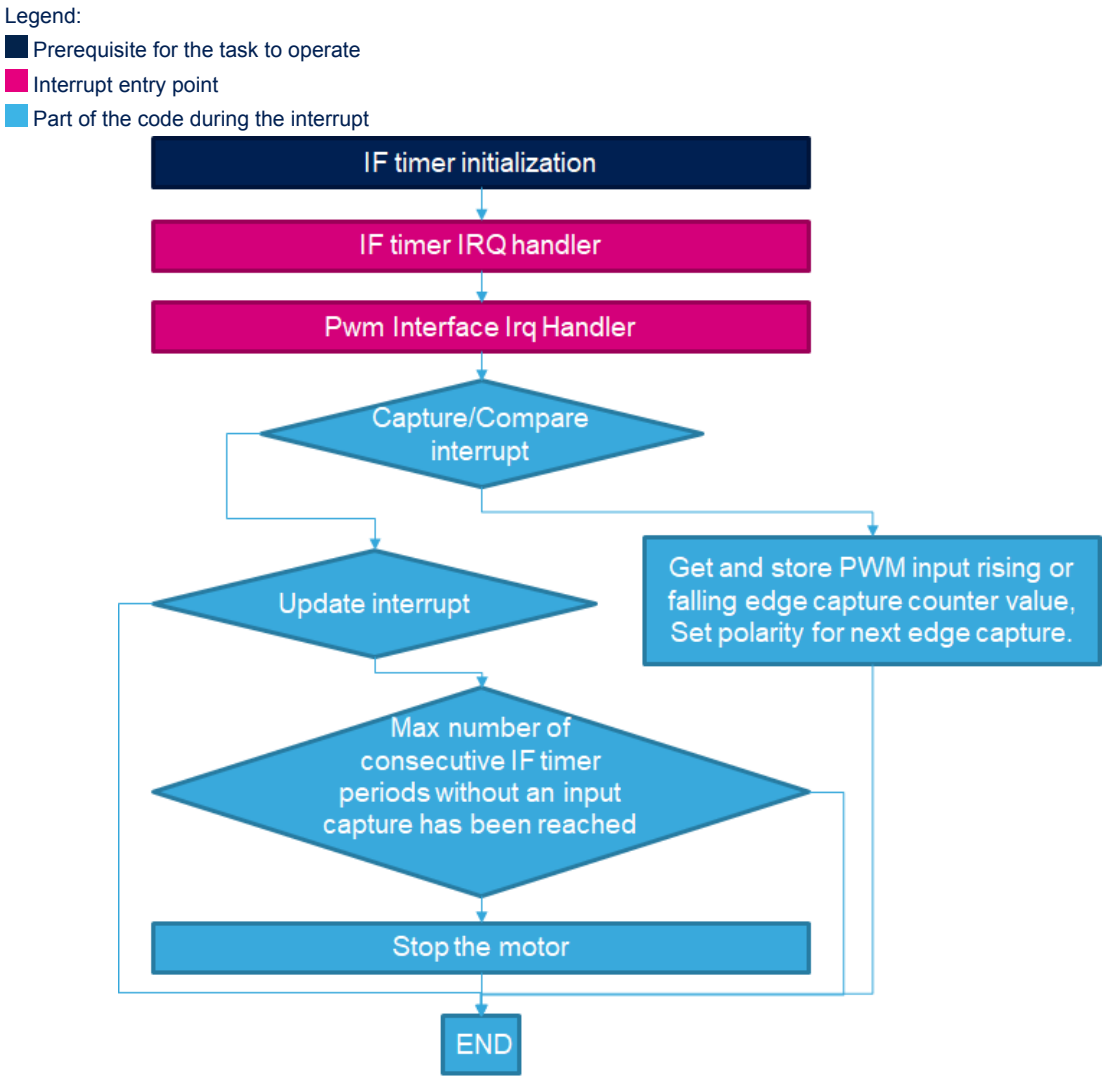
4.3.2.4 *Sensor-less communication task*

This is a low priority (2) task.

If the firmware includes the UART interface component, the task runs under the USART interrupt, which is enabled to receive several `MC_COM_RX_BUFFER_MAX` characters and, if the `McComReceiveFlag` is true, it is set to false in the task. The USART interrupt is disabled in the USART interrupt handler if the programmed number of characters has been received. When a LF character is received, the `McComReceiveFlag` is set to true.

If the firmware includes the PWM interface component, this task runs under the IF timer input capture interrupt and period elapsed interrupt. This task captures the rising edge and falling edge of the PWM signal used to control motor start, motor stop and motor speed (or torque) activity.

Figure 13. **PWM interface communication task**



4.3.2.5 *Sensor-less background task*

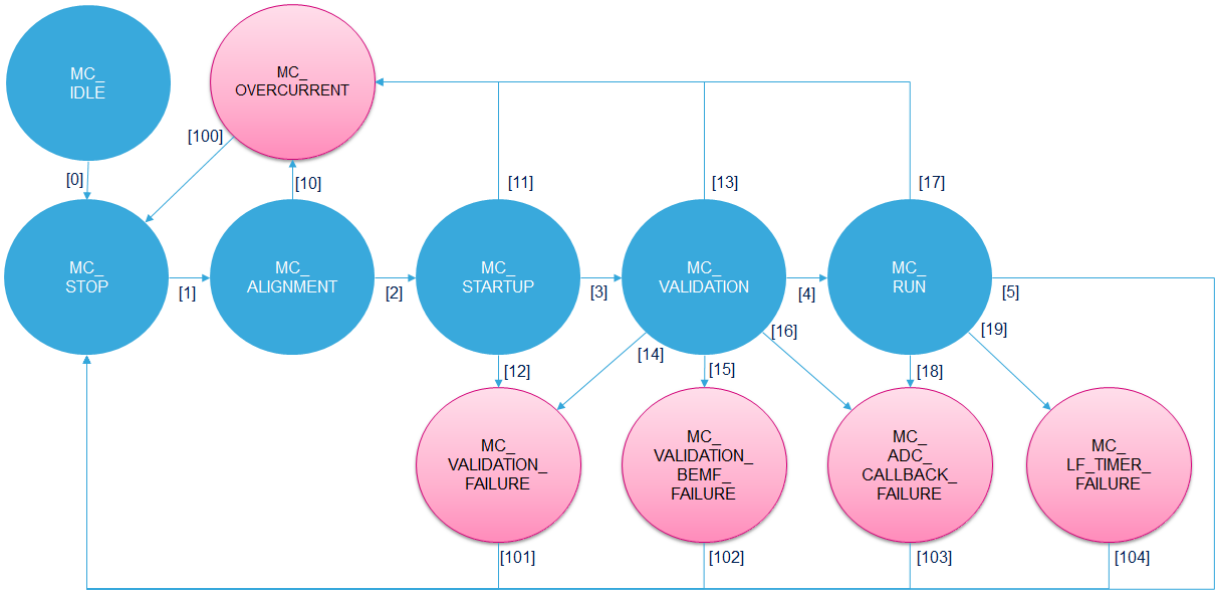
This task has no priority and is executed in the main file inside the while (1) loop. It is currently only used when a PWM pulse duration is used to control the motor. In this case, it processes the captured input values of the rising and falling edges of the PWM interface signal to control motor start, motor stop and motor speed (or torque) activity.

4.3.3 Sensor-less state machine

The 6-step firmware has several states depending on the execution schedule, on events and on user interventions.

Figure 14. 6-step sensors-less state machine

blue circles = expected states in normal operation
pink circles = error states



4.3.3.1 *Sensor-less normal states*

MC_IDLE

one-time state at the beginning of the 6-step firmware execution during its initialization

MC_STOP

the motor is stopped without current flowing

MC_ALIGNMENT

the motor is stopped and there is current flowing into its phases so that the rotor position is mid-way between two steps

MC_STARTUP

the motor runs in open loop and the step commutation time is set according to a speed linear acceleration

MC_VALIDATION

the motor runs in open loop and the step commutation time is constant; the firmware is monitoring BEMF zero crossing events trying to validate a closed loop operation

MC_RUN

the motor runs in closed loop and the step commutation time depends on the time of the last BEMF zero crossing detection

4.3.3.2 *Sensor-less error states*

The state machine enters an error state when abnormal behaviour is detected by the firmware, which calls an error function that is implemented in the user project. The default implementation of this function provided by the middleware template stops the motor.

MC_OVERCURRENT

the current flowing into the motor is too high

MC_VALIDATION_FAILURE

the start-up or the validation failed

MC_VALIDATION_BEMF_FAILURE

the validation failed because the BEMF zero-crossing detection is not reliable

MC_ADC_CALLBACK_FAILURE

there is an issue in the user ADC measurement

MC_LF_TIMER_FAILURE

speed feedback error probably due to a motor that is stalled or running too slowly

4.3.3.3 *Sensor-less transitions*

Referring to Figure 14. 6-step sensors-less state machine, the conditions for the numbered normal transitions are:

- [0] The firmware has been initialized.
- [1] The firmware has been ordered by the user to start the motor.
- [2] The alignment time is elapsed.
- [3] The start-up speed target has been reached: the programmed speed in open loop is equal or greater than the STARTUP_SPEED_TARGET.
- [4] The validation succeeded: the number of BEMF zero crossing events is equal or greater than the VALIDATION_ZERO_CROSS_NUMBER.
- [5] The firmware has been ordered by the user to stop the motor.

The conditions for the numbered error transitions are:

- [10], [11], [13] and [17] The firmware break interrupt has been called.
- [12] and [14] The number of executed steps is equal or greater than VALIDATION_STEPS_MAX.
- [15] The number of BEMF over threshold events is equal or greater than VALIDATION_BEMF_EVENTS_MAX.
- [16] and [18] The ADC channel to be processed during the ADC call-back is unexpected.
- [19] The period of the timer in charge of the step commutation has not be reprogrammed since the previous time the period elapsed call-back has been executed for this timer.

The conditions for the numbered post error transitions are:

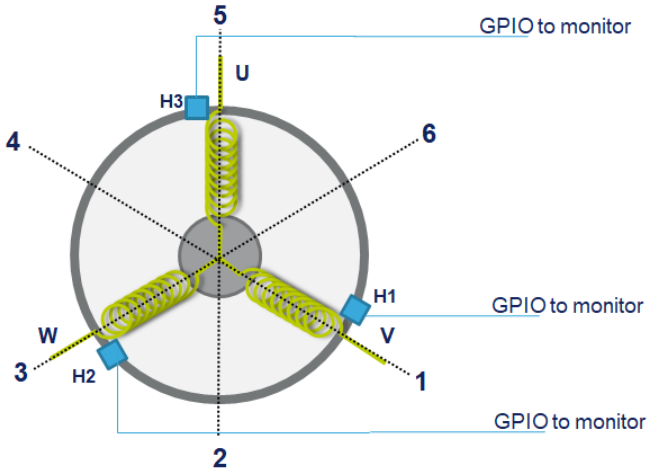
- [100], [101], [102], [103] and [104] The firmware calls an error function defined in the 6step configuration source file. By default, if the serial interface is available, the firmware reports the status to the user through this interface. Then the firmware stops the motor and changes the status accordingly.

4.4 Hall sensor algorithm

The three Hall effect sensors return digital signal values to indicate the rotor position. The firmware uses a configurable truth table to infer the next step number according to the hall status which is a pondered sum of the digital signal values from the Hall effect sensors.

Figure 15. **Motor with Hall effect sensors**

#	H1	H2	H3	Current
1	H	L	L	U → W
2	H	H	L	V → W
3	L	H	L	V → U
4	L	H	H	W → U
5	L	L	H	W → V
6	H	L	H	U → V



When the firmware is commanded to start the motor, the hall sensors are used to infer the position of the rotor so the firmware can program the next step according to the direction set by the user. When the motor does not move enough to change the hall status in the alignment time, the firmware either goes into an error state or tries to move to a different step depending on how the firmware is configured.

As soon as the rotor moves enough to change the hall status, the motor is assumed to be running. It keeps running until the user issues a stop command or an error occurs. If the motor is stalled while it is running (e.g., the rotor is inhibited for turning), the firmware will continue to try to reach the next step for a certain time, after which the firmware enters an error state and cuts off current to the motor. If the step reached during the run state is neither the next step nor the previous one for several consecutive times, the firmware enters an error state.

The step commutation for most motors with Hall sensors occurs as soon as the Hall status changes. In the 6-step firmware, the time between the hall status change and the step commutation is adjustable.

4.4.1 Hall sensor MCU hardware resource requirements and usage

The following MCU resources are required by the Hall sensor algorithm:

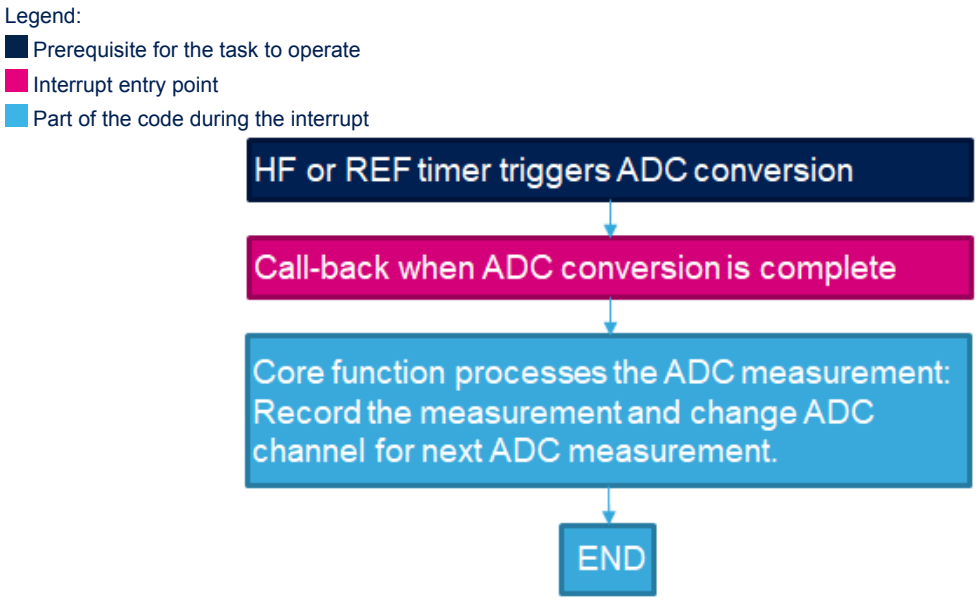
- A timer with four channels to generate three PWM signals and possibly one signal to trigger the ADC.
  - Each PWM channel has a counterpart or three GPIOs available to generate enable signals for each half bridge low side.
  - This timer is designated the high frequency (HF) timer.
- Three GPIOs connected to the digital Hall sensors.
- One interfacing timer to capture and time stamp the level transitions on these latter GPIOs.
  - This interfacing timer is also used to generate an event to generate a step commutation by changing the configuration of the HF timer.
  - This timer is designated the low frequency (LF) timer
  - The HF timer is triggered by the LF timer.
- A further timer for Current Mode to generate the PWM to build a reference voltage.
  - This timer is designated the reference (REF) timer.
- An ADC with several channels for user measurements.
  - The ADC must be able to be triggered by a timer trigger output pulse falling or rising edge.
  - The triggering time is expressed as a time in the HF timer period.
  - The timer triggering the ADC is possibly the HF timer or a timer synchronized with the HF timer.

4.4.2 Hall sensor tasks

4.4.2.1 Hall sensor high frequency task

This task is lowest priority (3) and is used for user ADC measurements.

Figure 16. **Hall sensors high frequency task call graph**



4.4.2.2 *Hall sensor medium frequency task*

This lowest priority (3) task occurs every 1 ms and manages the duty cycle (hence the pulse) of the HF PWMs or REF PWM every control loop time in milliseconds.

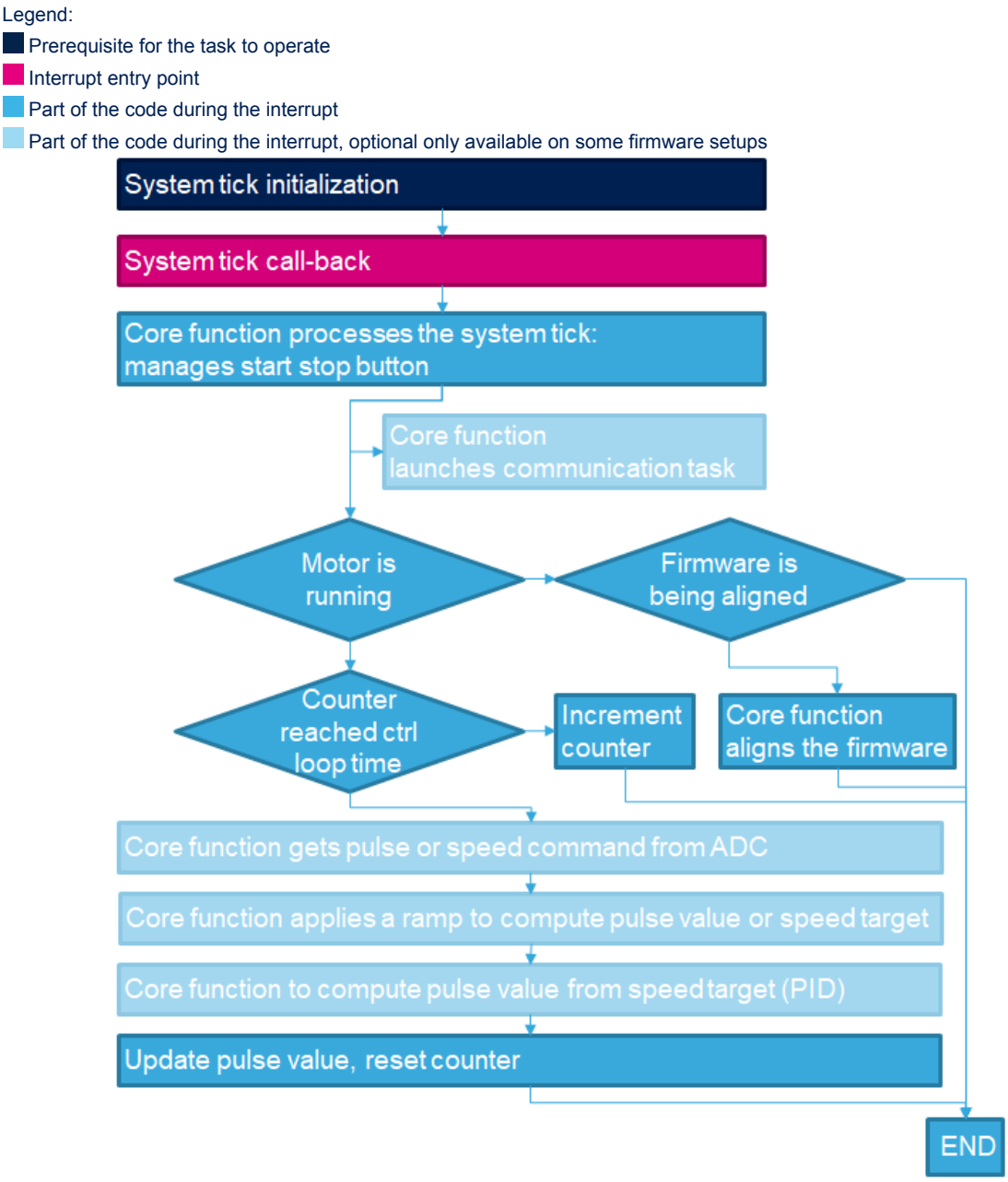
If the firmware includes the UART component, the task launches the serial communication task.

If the firmware includes the POT component, the task uses the ADC to infer a pulse or speed command.

If the firmware includes the PID component, the task manages the speed loop by calling a function with a PID regulator.

The light blue boxes are optional operations that depend on whether they are included in the firmware setup.

Figure 17. **Hall sensors medium frequency task call graph**

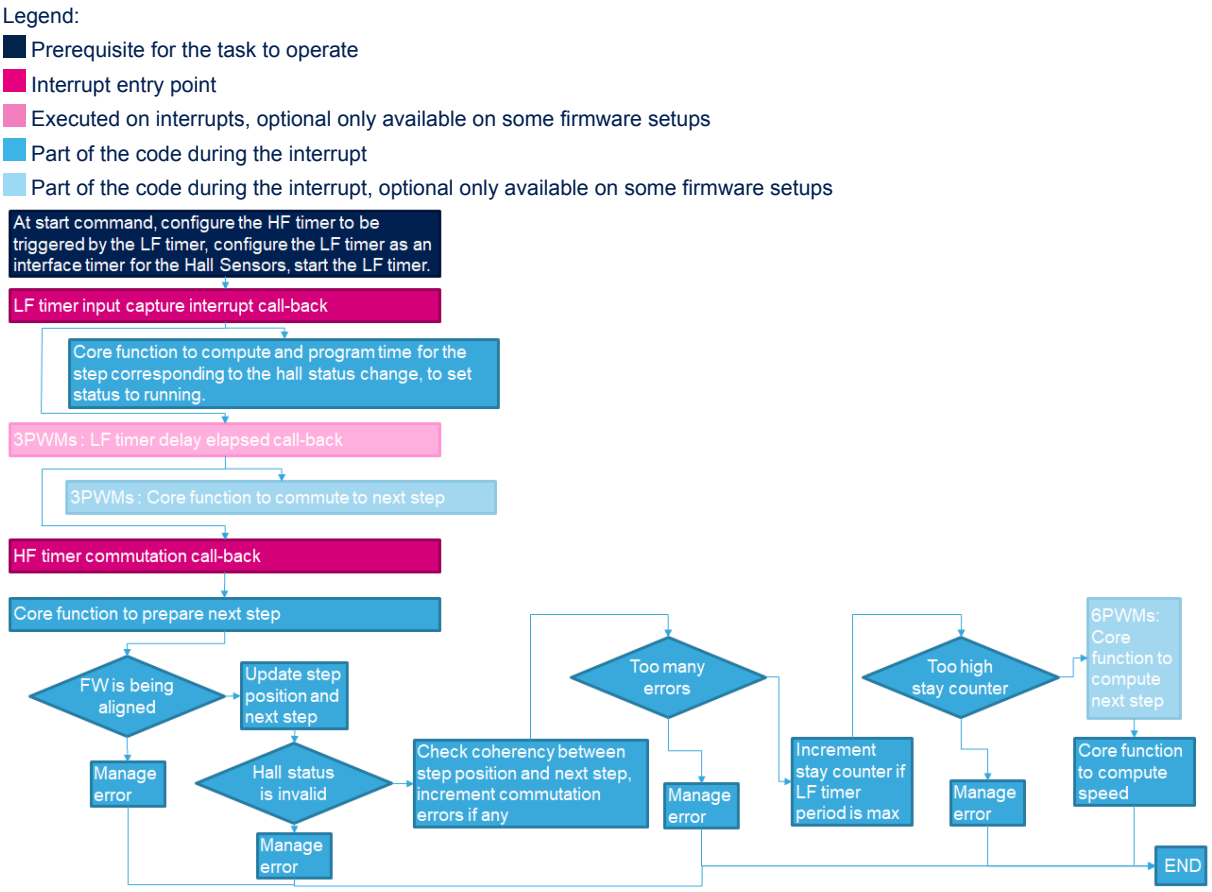


4.4.2.3 *Hall sensor low frequency task*

The frequency of this task depends on the current speed of the motor. It initially runs under a high priority (0) LF timer interrupt, and then under a medium priority (1) HF timer interrupt.

If the firmware is set up to include 3-PWM support, there is an additional LF timer interrupt used to manage the output level change of the GPIOs as these do not have preload registers. This interrupt occurs at the same time the commutation event is generated by the LF timer for the HF timer to transfer the content of the capture compare preload registers into the shadow registers. This way the change of the HF timer configuration and the 3 PWM GPIOs is almost simultaneous.

Figure 18. Hall sensors low frequency task call graph



4.4.2.4 *Hall sensor communication task*

This task is identical to the Sensor-less communication task. See Section 4.3.2.4 Sensor-less communication task.

4.4.2.5 *Hall sensor background task*

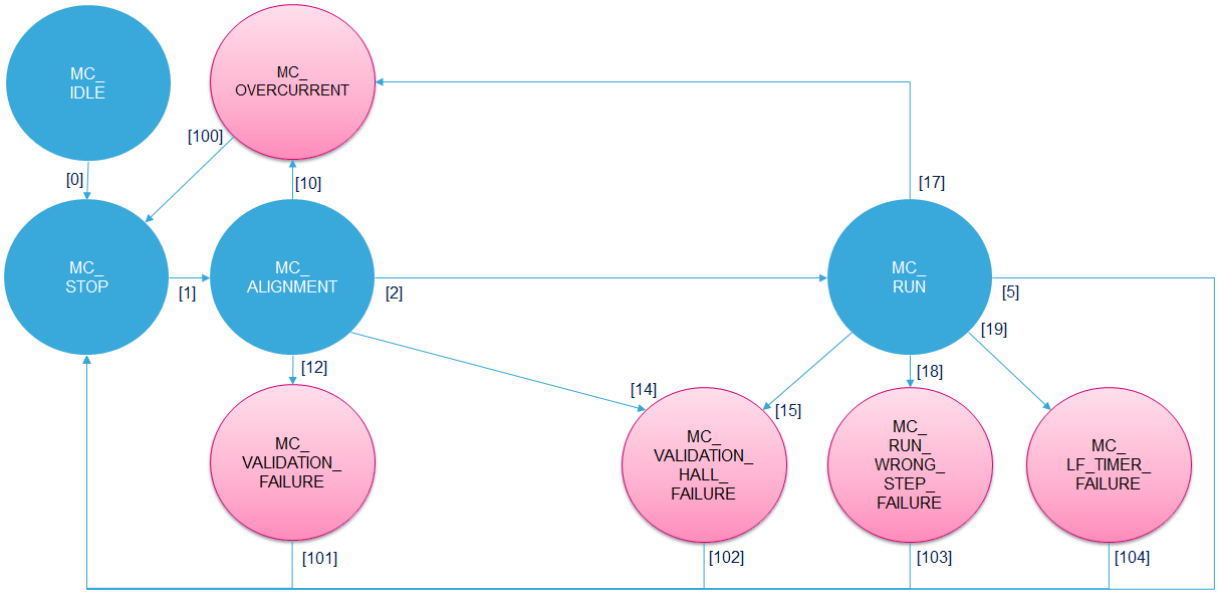
This task is identical to the Sensor-less background task. See Section 4.3.2.5 Sensor-less background task.

4.4.3 Hall sensor state machine

The 6-step firmware has several states depending on the execution schedule, on events and on user interventions.

Figure 19. 6-step hall sensor state machine

blue circles = expected states in normal operation
pink circles = error states



4.4.3.1 *Hall sensor normal states*

MC_IDLE

one-time state at the beginning of the 6-step firmware execution during its initialization

MC_STOP

the motor is stopped without current flowing

MC_ALIGNMENT

the motor is stopped. The firmware reads the hall sensor status and derives from it the step number. The firmware then tries to move the motor to the next step according to the direction set by the user. If the motor does not move enough to change the step status in the alignment time, the firmware either enters an error state or tries to move to a different step depending on the firmware setup.

MC_RUN

the motor runs in closed loop. The time of step commutation depends on the time of the last hall status change

4.4.3.2 *Hall sensor error states*

The state machine enters an error state when abnormal behaviour is detected by the firmware, which calls an error function that is implemented in the user project. The default implementation of this function provided by the middleware template stops the motor.

MC_OVERCURRENT

the current flowing into the motor is too high

MC_VALIDATION_FAILURE

the motor fails to move to the expected state in the required time

MC_VALIDATION_HALL_FAILURE

the hall status is invalid

MC_RUN_WRONG_STEP_FAILURE

the current step is neither the expected step nor the previous one

MC_LF_TIMER_FAILURE

speed feedback error probably due to a motor that is stalled or running too slowly

4.4.3.3 *Hall sensor transitions*

Referring to Figure 19. 6-step hall sensor state machine, the conditions for the numbered normal transitions are:

- [0] The firmware has been initialized.
- [1] The firmware has been ordered by the user to start the motor.
- [2] The motor moved such as the hall status changed to a valid value.
- [5] The firmware has been ordered by the user to stop the motor.

The conditions for the numbered error transitions are:

- [10] and [17] The firmware break interrupt has been called.
- [12] The motor did not move or moved such that the capture call-back of the timer in charge of the step commutation has not been executed.
- [14] and [15] The hall status value is invalid.
- [18] The hall status is valid, but the corresponding step is nor the expected step nor the previous one for more than RUN_COMMUTATION_ERRORS_MAX times.
- [19] The capture call-back of the timer in charge of the step commutation has not been executed while its period elapsed call-back has been executed the number of times corresponding to RUN_STAY_WHILE_STALL_MS time.

The conditions for the numbered post error transitions are:

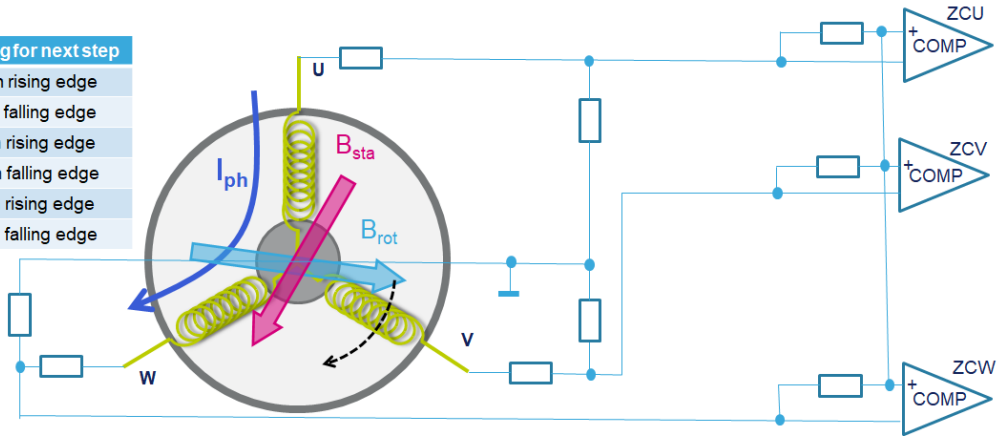
- [100], [101], [102], [103] and [104] The firmware calls an error function defined in the 6step configuration source file. By default, if the serial interface is available, the firmware reports the status to the user through this interface. The firmware then stops the motor and changes the status accordingly.

4.5 Sense comparators algorithm

In Sense Comparators Mode, the BEMF zero-crossing on each motor coils is detected by comparing voltage on the floating phase with a reconstructed center-tap voltage.

Figure 20. Motor with sense comparators circuit

Step	Monitoring for next step
1	ZCW on rising edge
2	ZCV on falling edge
3	ZCU on rising edge
4	ZCW on falling edge
5	ZCV on rising edge
6	ZCU on falling edge



The BEMF zero-crossing corresponds to either a rising or falling edge of one of the three comparator outputs. As in the sensors-less algorithm, the step commutation occurs half a step time after or before. The time between the zero-crossing detection and the step commutation is adjustable. There is no need to manage high frequency PWM on and off time to detect the BEMF zero-crossing.

The output of the comparators output is read according to the table in figure20 during the call-back of a dedicated timer, designated as the ZC timer. The task running under this call-back is designated as the zero-crossing task, which runs at a higher frequency than the HF task, allowing a higher time resolution in the BEMF zero-crossing detection.

The step commutation is performed in the LF task, which is executed under the call-back of the LF timer programmed in the ZC task.

To start a motor, the firmware enters an alignment state where it pushes current into the motor to set the rotor at a predefined step position. The high frequency PWM is started at the beginning of the alignment phase and the low frequency timer is started at the end.

The firmware then performs an open loop speed linear acceleration until it reaches a predefined target speed, or a valid zero-crossing detection is performed. In the first case, the firmware only starts monitoring BEMF zero-crossing when the target speed is reached; in the second case. it does so as soon as the motor starts to accelerate.

The start of the ZC timer is the BEMF zero-crossing monitoring start. On the first BEMF zero-crossing detection, the enters the running state and the next step commutation is computed according to the time of the BEMF zero-crossing. On each subsequent BEMF zero-crossing, the firmware computes a new step commutation until a stop command is issued or an error occurs.

4.5.1 Sense Comparator MCU hardware resource requirements and usage

The following MCU resources are required by the Sense Comparator algorithm:

- Three GPIOs connected to the comparator outputs.
- One timer to generate a time base to monitor these GPIOs.
  - This timer is designated as the Zero Crossing (ZC) timer.
- One timer with four channels to generate three PWM signals and possibly one signal to trigger the ADC.
  - Each PWM channel must have a counterpart or there must be three GPIOs available to generate enable signal for each half bridge low side.
  - This timer is designated as the High Frequency (HF) timer.
- One timer to generate a time base managing the steps commutation.
  - This timer is designated as the Low Frequency (LF) timer.
- In case of current mode, a further timer to generate the PWM to build a reference voltage.
  - This timer is designated as the Reference (REF) timer.

4.5.2 Sense Comparator tasks

4.5.2.1 Sense Comparator zero-crossing task

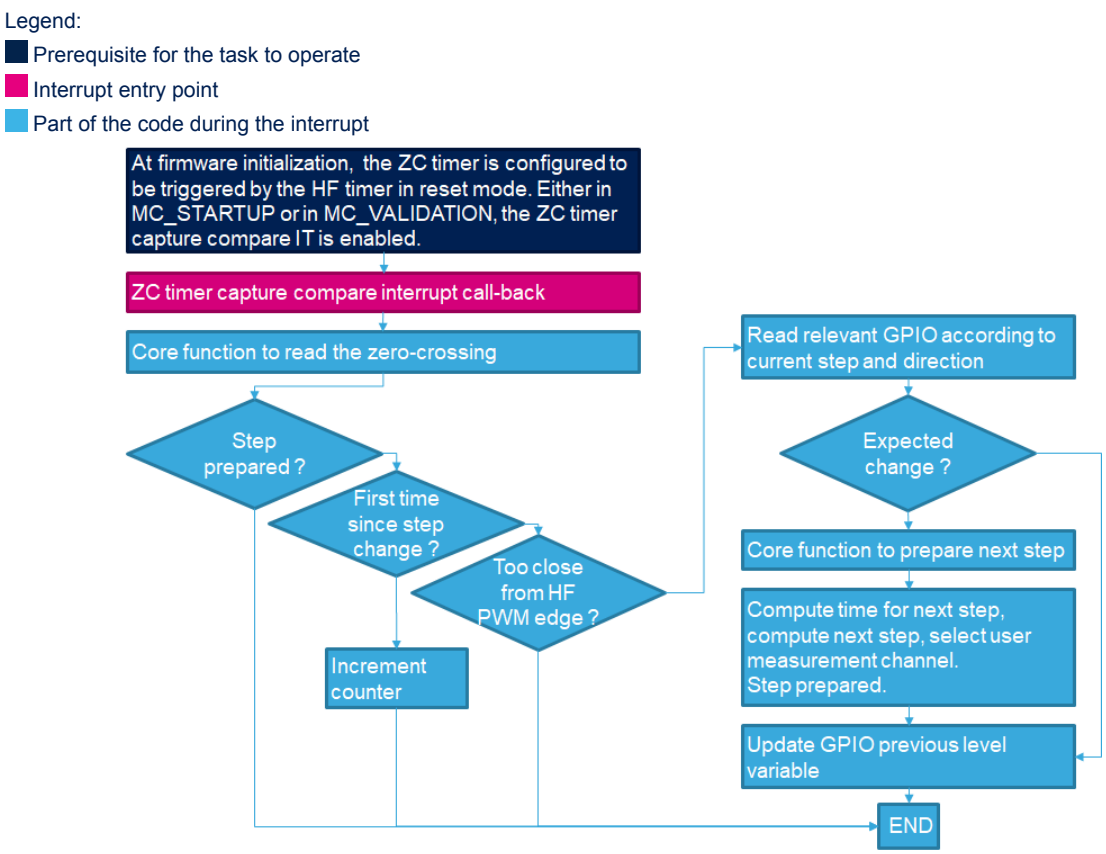
This medium priority (1) task occurs periodically at the highest frequency of all timers. It polls the status of the GPIO input connected to the sense comparator outputs. The higher the frequency of this task, the better the time resolution of the zero-crossing detection. This is one of the most demanding tasks on the microcontroller processing time.

To ensure reliability, zero-crossing detection is not attempted in the following circumstances:

- When the zero-crossing has already been found and the corresponding step has not been performed.
- The first time the task is executed after a step commutation.
- When the zero-crossing task is executed too close to a rising or a falling edge of the HF PWM. For this purpose, three guard times are defined and may have to be tuned by the user depending on the application:
  - Time in ZC timer counter cycles after the HF timer PWM edge to read the comparator output
  - Time in HF timer counter cycles after the HF timer PWM edge to read the comparator output
  - Time in HF timer counter cycles before the HF timerPWM edge to read the comparator output

The zero crossing is found when the relevant GPIO has a level change as shown in the table in Figure 20. Motor with sense comparators circuit. When the zero-crossing is found, the next step is prepared.

Figure 21. Sense comparators zero-crossing task call graph



4.5.2.2 *Sense Comparator high frequency task*

This lowest priority (3) task is used for the user ADC measurements in the same way as the Hall sensor high frequency task; see Section 4.4.2.1 Hall sensor high frequency task.

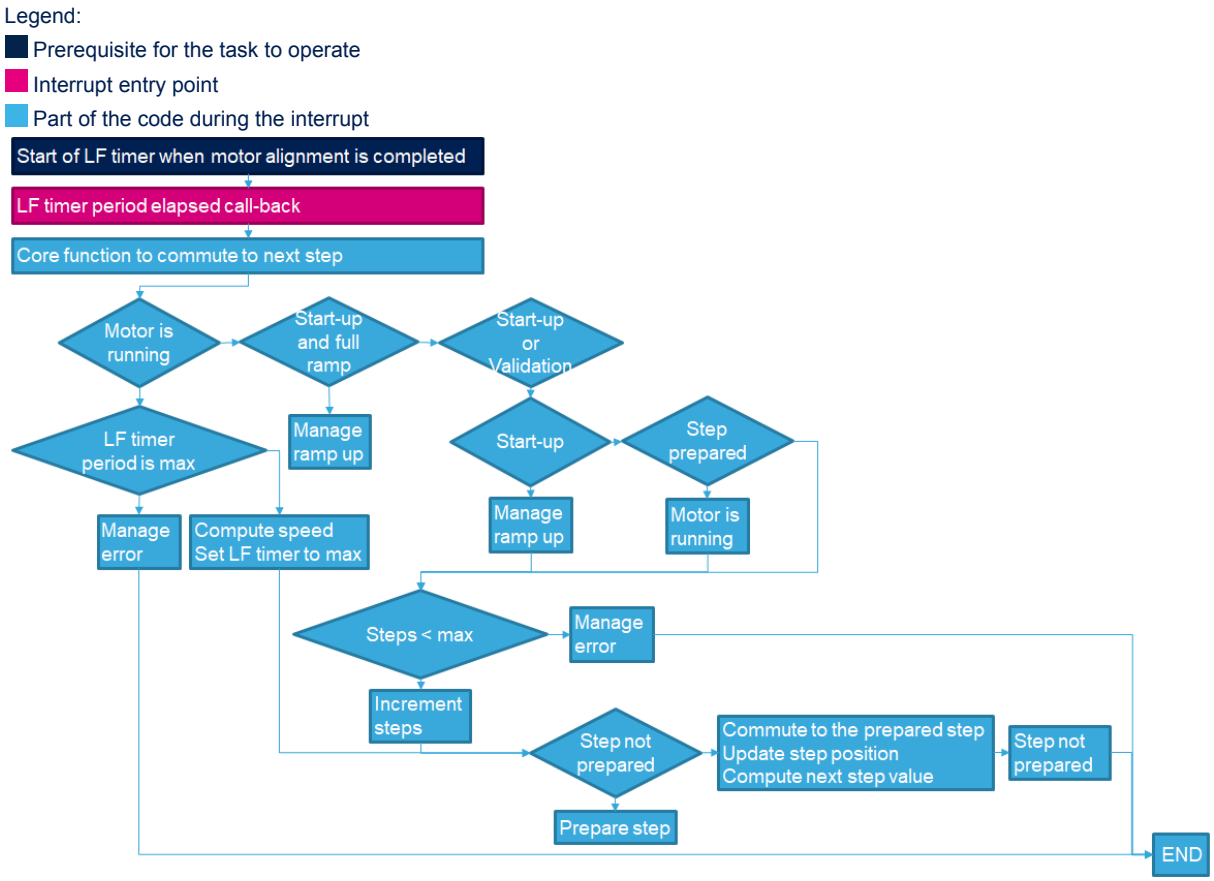
4.5.2.3 *Sense Comparator medium frequency task*

This lowest priority (3) task is very similar to the Sensors-less counterpart, except that no fast demagnetization management is required; see Section 4.3.2.2 Sensor-less medium frequency task.

4.5.2.4 *Sense Comparator low frequency task*

The frequency of this high priority (0) task is depends on the current speed of the motor. It runs under an LF timer interrupt and commutes to a new step in the 6step sequence. It computes the filtered speed feedback.

Figure 22. Sense comparators low frequency task call graph



4.5.2.5 *Sense Comparator communication task*

This task is identical to the Sensor-less communication task. See Section 4.3.2.4 Sensor-less communication task.

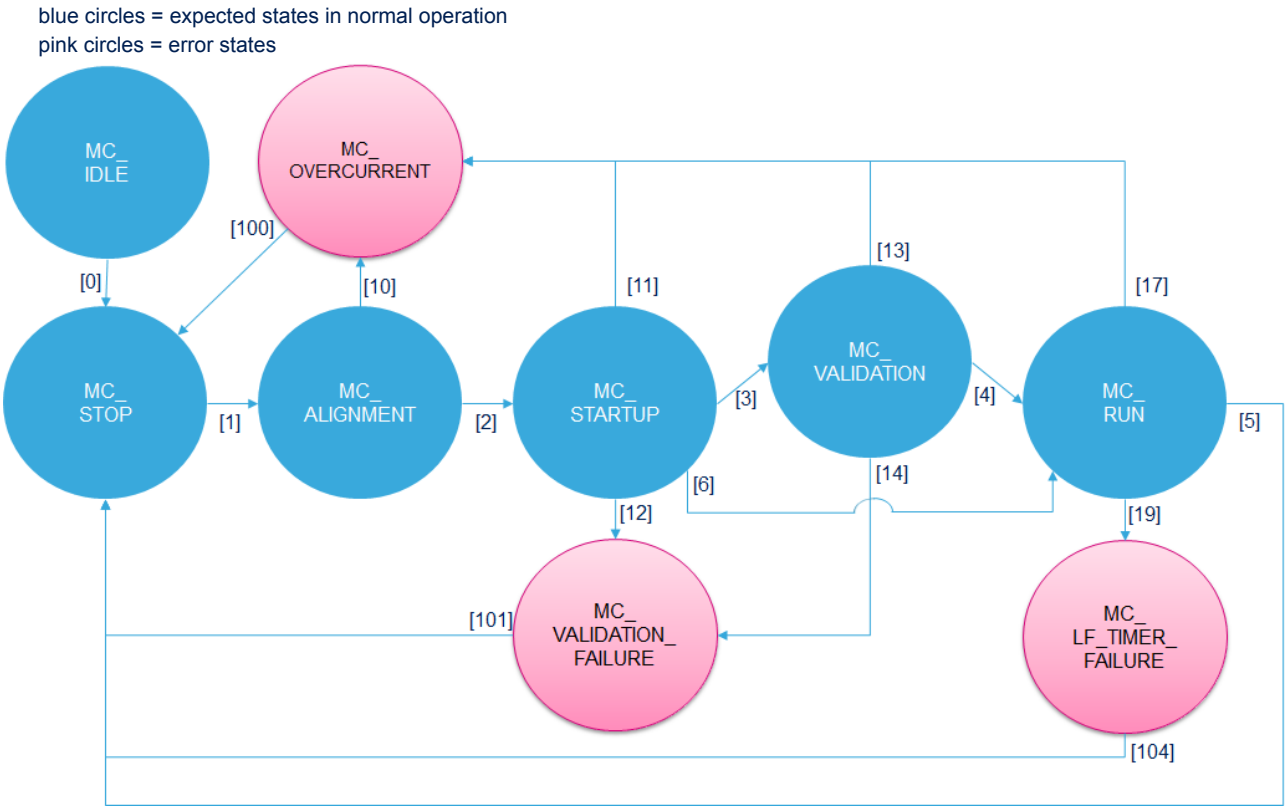
4.5.2.6 *Sense Comparator background task*

This task is identical to the Sensor-less background task. See Section 4.3.2.5 Sensor-less background task.

4.5.3 Sense Comparator state machine

The 6-step firmware has several states depending on the execution schedule, on events and on user interventions.

Figure 23. **6-step sense comparators state machine**



4.5.3.1 *Sense Comparator normal states*

MC_IDLE

one-time state at the beginning of the 6-step firmware execution during its initialization

MC_STOP

the motor is stopped without current flowing

MC_ALIGNMENT

the motor is stopped and there is current flowing into its phases such that the rotor position corresponds to a step

MC_STARTUP

the motor is running in open loop and the time of step commutations is programmed according to a speed linear acceleration

MC_VALIDATION

the motor is running in open loop and the time of step commutation is constant. The firmware is monitoring the output of the sense comparators to detect a zero crossing to validate a closed loop operation

MC_RUN

the motor is running in closed loop. The step commutation time depends on the time of the last zero crossing detection from the sense comparator output readings

4.5.3.2 *Sense Comparator error states*

The state machine enters an error state when abnormal behaviour is detected by the firmware, which calls an error function that is implemented in the user project. The default implementation of this function provided by the middleware template stops the motor.

MC_OVERCURRENT

the current flowing into the motor is too high

MC_VALIDATION_FAILURE

the firmware fails to move the motor to the expected state in the required time

MC_LF_TIMER_FAILURE

speed feedback error probably due to a motor that is stalled or running too slowly

4.5.3.3 *Sense Comparator transitions-con*

Referring to Figure 23. 6-step sense comparators state machine, the conditions for the numbered normal transitions are:

- [0] The firmware has been initialized.
- [1] The firmware has been ordered by the user to start the motor.
- [2] The alignment time is elapsed.
- [3] The full ramp start is enabled in the firmware and the start-up speed target has been reached: the programmed speed in open loop is equal or greater than the STARTUP_SPEED_TARGET.
- [4] The validation succeeded: the next step has been prepared during the interrupt of the timer in charge of detecting the zero crossing of the BEMF.
- [5] The firmware has been ordered by the user to stop the motor.
- [6] The full ramp start is disabled in the firmware and the validation succeeded: the next step has been prepared during the interrupt of the timer in charge of detecting the zero crossing of the BEMF.

The conditions for the numbered error transitions are:

- [10], [11], [13] and [17] The firmware break interrupt has been called.
- [12] The start-up speed target has been reached: the programmed speed in open loop is equal or greater than the STARTUP_SPEED_TARGET.
- [14] The number of executed steps is equal or greater than VALIDATION_STEPS_MAX.
- [19] The period of the timer in charge of the step commutation has not been reprogrammed since the previous time the period elapsed call-back has been executed for this timer.

The conditions for the numbered post error transitions are:

- [100], [101] and [104] The firmware calls an error function defined in the 6step configuration source file. By default, the firmware reports the status to the user through the serial interface if it is available. The firmware then stops the motor and changes the status accordingly.

5 User guide

5.1 6-step firmware setup

The 6-step firmware can come in different setup configurations in terms of the rotor position sensing method and driving modes that govern how the current is injected into the motor phases. There are also several other features that help adapt the 6-step firmware to specific application requirements.

To build the 6-step middleware for an application, the user needs to make choices regarding the following options:

- Driving mode:
  - Voltage. See Section 4.2.1 Voltage driving mode.
  - current. See Section 4.2.2 Current driving mode.
- Sensing method (mutually exclusive):
  - sensors-less; see Section 4.3 Sensors-less algorithm
  - hall sensors; see Section 4.4 Hall sensor algorithm
  - sense comparators; see Section 4.5 Sense comparators algorithm.

Rectification:

- Speed loop:
  - Enabled: the duty cycle of the HF PWM driving the current injection into the motor inductors or the duty cycle of the REF PWM used as a current reference is controlled by the output of a PID regulator
  - Disabled: the duty cycle is directly controlled by the user
- Set point ramping:
  - Enabled: the user command for the speed reference or for the HF or REF duty cycle is linearly ramped from the value at the time the command is issued to the commanded value
  - Disabled: the speed reference or the HF or REF duty cycle is updated each control loop time with the acceleration
- Three-PWM:
  - Enabled: three PWM signals with three enabling signals (GPIOs with constant level during a step) are needed to drive the motor
  - Disabled: three PWM signals with three complementary ones are needed
- Fast demagnetization:
  - Enabled: the MOSFET side where the driving PWM is fed is changed at each step commutation from the high to the low side or from the low to the high side to have the highest inverse voltage on the motor floating inductor and so the lowest time needed for demagnetization. This allows to monitor earlier the zero-crossing of the BEMF in the sensors-less algorithm.
- User interface (mutually exclusive):
  - UART (serial communication): the motor start, stop, direction, operating point (speed or torque through the PWM duty cycle), gate driver frequency, initial torque, BEMF sensing location and USER ADC measurement location are controlled though a terminal window using set commands
  - potentiometer (ADC measurement): the motor operating point is a linear function of the voltage read on the potentiometer through the ADC. Motor start and stop operations require a button
  - PWM: the motor start, stop and operating point are commanded through the duty cycle of an interface timer as specified in the ESC communication protocols listed below.
    - PWM (1000-2000 µs, tested with 480 Hz frequency) - compatible with ST ESC interface described in UM2197
    - ONESHOT125 (tested with 2 kHz frequency)
    - ONESHOT42 (tested with 4 kHz frequency)
    - MULTISHOT (tested with 4 kHz frequency)

5.2 API presentation and description

The firmware has a set of functions that the user can use to simply build its own application. By default, the first parameter of every function described herein is a motor control handle (`MC_Handle_t *pMc`) corresponding to the motor being controlled. This is a provision to allow the control of several motors with the same micro-controller.

The pointers to the start of the structure of a timer are pointers on `uint32_t` and the timer structure is a `TIM_HandleTypeDef` definition from HAL.

5.2.1 API common to all firmware setup

5.2.1.1 MC_Com_Init

This function initializes the serial communication task. If the serial communication has been selected in the firmware setup, this function has to be called after the `MC_Core_Init` function.

Number of parameters: 1

uint32_t *pMcCom

Pointer cast from `UART_HandleTypeDef` handle definition from HAL.

5.2.1.2 MC_Core_AssignTimers

This function assigns timers to the motor control structure. The `pHfTimer` and `pLfTimer` must be non-null pointers as they are mandatory for the operation of the motor whatever the chosen algorithm. The `pRefTimer` is mandatory in case of the driving mode is Current. The `pRefTimer` can also be used in the Voltage Mode as a mean to trig the ADC. The `pZcTimer` is mandatory for the Sense Comparators algorithm and it has no other use. In case there is no use of a timer, the NULL pointer must be used for its corresponding parameter.

Number of parameters: 1+4

uint32_t *pHfTimer

pointer to the start of the structure of the high frequency timer used to generate the PWM signals which drive the motor phases.

uint32_t *pLfTimer

pointer to the start of the structure of the low frequency timer used to generate a step commutation.

uint32_t *pRefTimer

pointer to the start of the structure of the reference timer used to generate the voltage reference in current mode and possibly an ADC trigger.

uint32_t *pZcTimer

pointer to the start of the structure of the zero-crossing timer used to find when the BEMF voltage zero crossing occurs with the help of sense comparators.

5.2.1.3 MC_Core_AssignUserMeasurementToSpeedDutyCycleCommand

This function assigns one of the five possible user measurements of the `MC_UserMeasurements_t` structure to a variable in the motor control handle indicating that this measurement is the one to be used to compute to compute either a speed or a duty cycle command for the motor. A user measurement must be linked to an ADC and an ADC channel with the `MC_Core_ConfigureUserAdcChannel` function.

Number of parameters: 1+1

MC_UserMeasurements_t UserMeasurement

User measurement used to build the speed command or the duty cycle command.

5.2.1.4 MC_Core_BackgroundTask

The only use of this function is when the PWM is selected as a User Interface in the firmware setup. This function has then to be called in the main program while (1) loop. Its purpose is to manage the lowest priority part of the communication PWM interface by processing the timer input rising and falling captures to prepare the computation of a speed or pulse command.

Number of parameters: 1+0

5.2.1.5 *MC_Core_ConfigurePwmInterface*

The only use of this function is when the PWM is selected as a User Interface in the firmware setup. This function has then to be called in the main program before the `MC_Core_Init` and the `MC_Core_BackgroundTask`. Its purpose is to configure the PWM interface.

Number of parameters: 1+9

uint32_t *pIfTimer

pointer to the start of the timer used for the pwm interface

uint32_t IfTimerChannel

channel of the timer used for the pwm interface. This parameter has to be cast from a `TIM_LL_EC_CHANNEL` low layer definition such as `LL_TIM_CHANNEL_CH1`.

uint16_t PulseUsForMaximumThrottle

captured pulse duration in us for max command

uint16_t PulseUsForMinimumThrottle

captured pulse duration in us for min command

uint32_t TimerPulseMaxValue

pulse value of the timer used for the pwm interface

uint8_t ArmingCnt

when the motor is stopped, number of consecutive pulses below minimum command duration needed to check pulses in the minimum to maximum command range to possibly start the motor

uint8_t StartCnt

number of consecutive pulses above min command duration needed to start the motor

uint16_t StopCnt

number of consecutive pulses below min command duration needed to stop the motor

uint16_t NoSignalStopMs

time in ms without pulse after which the motor stops

5.2.1.6 *MC_Core_ConfigureUserAdc*

This function configures the User ADC by assigning to it a timer and one channel of this timer. The assigned timer channel is used to trigger the ADC conversion. The period of the assigned timer is set equal to the period of the HF timer.

Number of parameters: 1+3

uint32_t *pTrigTimer

pointer to the timer used to trig the ADC

uint16_t TrigTimerChannel

channel of the timer used to trig the ADC. This parameter must be cast from a `TIM_LL_EC_CHANNEL` low layer definition such as `LL_TIM_CHANNEL_CH4`

uint8_t NumberOfUserChannels

number of channels to be used for the user measurements, maximum 5

5.2.1.7 *MC_Core_ConfigureUserAdcChannel*

This function links a user measurement to an ADC channel and programs the sampling time for this ADC channel.

Number of parameters: 1+4

uint32_t* pAdc

pointer on the ADC to be selected.

uint32_t AdcChannel

ADC channel to be selected. Use the channel (_CH) definition in the `<power board or inverter board>_conf.h` file corresponding to the type of measurement to be performed; i.e., `STEVAL_SPIN3204_ADC_VBUS_CH`

uint32_t SamplingTime

ADC sampling time to be selected. Use the sampling time (_ST) definition in the `<power board or inverter board>_conf.h` file corresponding to the type of measurement to be performed; i.e., `STEVAL_SPIN3204_ADC_VBUS_ST`

MC_UserMeasurements_t UserMeasurement

User measurement to map to ADC channel.

5.2.1.8 *MC_Core_ConfigureUserButton*

This function configures a user button and its debounce time. This button can be used as start/stop button or anything else. The callback implementing the operations performed upon pressing the button is the function void `HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin )` in the `6step_conf.c`.

Number of parameters: 1+2

uint16_t ButtonPin

cast from a GPIO pin definition corresponding to the button. Use a definition from the `main.h` file where are located the definitions generated with CubeMx; i.e., `USER1_LED_BUTTON_Pin GPIO_PIN_0`

ButtonDebounceTimeMs

button debounce time in ms

5.2.1.9 *MC_Core_GetGateDriverPwmFreq*

This function returns the gate driver frequency stored in the motor control structure.

Number of parameters: 1+0

5.2.1.10 *MC_Core_GetMotorControlHandle*

This function returns the motor control handle corresponding to the `MotorDeviceId` number (from 0 to `NUMBER_OF_DEVICES`-1)

Number of parameters: 1

uint8_t MotorDeviceId

5.2.1.11 *MC_Core_GetSpeed*

This function returns the filtered speed (Hz) feedback stored in the motor control structure.

Number of parameters: 1+0

5.2.1.12 *MC_Core_GetStatus*

This function returns the motor status (`MC_Status_t` type as shown below) stored in the motor control structure.

Number of parameters: 1+0

```
typedef enum {
  MC_IDLE = ((uint8_t) 0),
  MC_STOP,
  MC_ALIGNMENT,
  MC_STARTUP,
  MC_VALIDATION,
  MC_RUN,
  MC_OVERCURRENT,
  MC_VALIDATION_FAILURE,
  MC_VALIDATION_BEMF_FAILURE,
  MC_VALIDATION_HALL_FAILURE,
  MC_RUN_WRONG_STEP_FAILURE,
  MC_LF_TIMER_FAILURE,
/* PWM INTERFACE BEGIN 1 */
  MC_ADC_CALLBACK_FAILURE,
  MC_PWM_INTERFACE_FAILURE
/* PWM INTERFACE END 1 */
}MC_Status_t;
```

5.2.1.13 *MC_Core_Init*

This function must be called after all the timer assignments, ADC configurations, measurement configurations and the user interface configuration, except the serial communication (`MC_Com_Init`).

This function instances a motor control handle per call, with a handle id of 0 for the first call and increments the id value by 1 for each subsequent call. It initializes the motor control structure content with device status and values which are not expected to change during run-time such as the number of motor pole pairs.

Moreover, it calls the `MC_Core_Reset` function to initialize values that may change during run-time.

Finally, it starts the PWM interface if it exists.

Number of parameters: 1+0

5.2.1.14 *MC_Core_Reset*

This function initializes values that may change during run-time. It is called by the `MC_Core_Stop` and the `MC_Core_Init` functions. It is not intended to be called directly by the user, but it may be useful in some cases.

Number of parameters: 1+0

5.2.1.15 *MC_Core_SetAdcUserTrigTime*

This function sets inside the HF timer PWM period the trig time of the ADC. Note that the timer used to trig the ADC is either the HF timer or a timer that has the same period as the HF timer and is synchronized with it.

Number of parameters: 1+1

uint32_t DutyCycleToSet

Duty cycle in 1/1024 of PWM period.

5.2.1.16 *MC_Core_SetDirection*

This function sets the direction of the motor rotation.

Number of parameters: 1+1

uint32_t DirectionToSet

direction to set where 0 is for positive speed (clockwise direction) and 1 for negative speed (counterclockwise direction).

5.2.1.17 *MC_Core_SetDutyCycle*

This function sets the pulse command corresponding to the duty cycle parameter taking into account the timer period. The timer on which the pulse is set is either the HF timer for voltage mode driving or the REF timer for current mode driving.

Number of parameters: 1+1

uint32_t DutyCycleToSet

Duty cycle in 1/1024 of PWM period.

5.2.1.18 *MC_Core_SetGateDriverPwmFreq*

This function sets the gate driver PWM frequency. If this function is called while the motor is not stopped, it stops the motor as its first operation. This function also sets all the timer periods depending on the gate driver PWM frequency and calls the `MC_Core_Reset` function.

Number of parameters: 1+1

uint32_t FrequencyHzToSet

frequency in Hz to be set.

5.2.1.19 *MC_Core_SetSpeed*

This function sets the speed command with the `SpeedToSet`. The speed command is taken into account in the medium frequency task every control loop time. The previous speed command becomes immediately the speed target value eventually bypassing the acceleration mechanism.

Number of parameters: 1+1

uint32_t SpeedToSet

speed command to be set in Hz

5.2.1.20 *MC_Core_SetStartupDutyCycle*

This function sets the startup reference corresponding to the duty cycle parameter taking into account the timer period. The timer, on which the pulse is set, is either the HF timer for voltage mode driving or the REF timer for current mode driving. The startup reference is used during `MC_ALIGNEMENT` and `MC_STARTUP` states.

Number of parameters: 1+1

uint32_t DutyCycleToSet

Duty cycle in 1/1024 of PWM period.

5.2.1.21 *MC_Core_Start*

This function starts the motor if the motor is stopped. It launches also the ADC calibration, programs the timer pulse to trig the ADC, configures the commutation event for the HF timer (this event triggers the transfer of the preload register content to the shadow register), and if the REF timer exists programs its pulse value and starts it.

Number of parameters: 1+0

5.2.1.22 *MC_Core_Stop*

This function stops the motor by stopping LF and HF timers. It stops also the ADC interrupts and the REF timer if it exists. In case of the Sense Comparators algorithm, it stops the ZC timer. Finally, it resets all the motor control variables by calling the `MC_Core_Reset` function.

Number of parameters: 1+0

5.2.2 API dedicated to sensors-less operation

5.2.2.1 *MC_Core_ConfigureBemfAdc*

This function configures an ADC by assigning to it a timer and one channel of this timer. The assigned timer channel is used to trigger the ADC conversion for the BEMF measurement. The period of the assigned timer is set equal to the period of the HF timer. The same ADC is used for the three BEMF phases.

Number of parameters: 1+3

uint32_t * pAdc

pointer on the ADC used for BEMF

uint32_t *pTrigTimer

pointer on the handle of the timer used to trig the ADC

uint16_t TrigTimerChannel

channel of the timer used to trig the ADC. This parameter must be cast from a `TIM_LL_EC_CHANNEL` low layer definition such as `LL_TIM_CHANNEL_CH4`

5.2.2.2 *MC_Core_ConfigureBemfAdcChannel*

This function links a motor BEMF phase to an ADC channel and programs the sampling time for this ADC channel.

Number of parameters: 1+3

uint32_t AdcChannel

ADC channel to be selected. Use the channel (_CH) definition in the `<power board or inverter board>_conf.h` file corresponding to the `BemfPhase`; i.e., `STEVAL_SPIN3204_ADC_BEMF_CH1`

uint32_t SamplingTime

ADC sampling time to be selected. Use the definition in the `<power board or inverter board>_conf.h` file corresponding to the BEMF sampling time (i.e., `STEVAL_SPIN3204_ADC_BEMF_ST`)

MC_BemfPhases_t BemfPhase

BEMF phase to map to ADC channel

```
typedef enum {
  MC_BEMF_PHASE_1   = ((uint8_t) 0),
  MC_BEMF_PHASE_2   = ((uint8_t) 1),
  MC_BEMF_PHASE_3   = ((uint8_t) 2)
}MC_BemfPhases_t;
```

5.2.2.3 *MC_Core_SetAdcBemfTrigTime*

This function sets inside the HF timer PWM period the trig time of the ADC. Note that the timer used to trig the ADC is either the HF timer or a timer that has the same period as the HF timer and is synchronized with it.

Number of parameters: 1+1

uint32_t DutyCycleToSet

Duty cycle in 1/1024 of PWM period.

5.2.3 API dedicated to sense comparators

5.2.3.1 *MC_Core_ConfigureSenseComparators*

This function assigns the GPIO pins and ports to the dedicated motor control handle variables.

Number of parameters: 1+6

pPortU

pointer to the port of the GPIO used to monitor the U motor phase comparator output

PinU

GPIO pin of the GPIO port used to monitor the U motor phase comparator output

pPortV

pointer to the port of the GPIO used to monitor the V motor phase comparator output

PinV

GPIO pin of the GPIO port used to monitor the V motor phase comparator output

pPortW

pointer to the port of the GPIO used to monitor the W motor phase comparator output

PinW

GPIO pin of the GPIO port used to monitor the W motor phase comparator output

5.2.3.2 *MC_Core_GetZcReadFreq*

This function returns the zero-crossing reading frequency stored in the motor control structure.

Number of parameters: 1+0

5.2.3.3 *MC_Core_SetZcReadFreq*

This function must be used only when the motor is stopped. It sets the zero-crossing reading frequency. After this function has been called, the `MC_Core_Reset` function must then be called to set other variables depending on the zero-crossing read frequency.

Number of parameters: 1+1

uint32_t FrequencyHzToSet

frequency in Hz to be set

5.3 Firmware parameters

5.3.1 Common parameters

MOTOR_NUM_POLE_PAIRS

Number of Motor Pole pairs

ALIGNMENT_TIME

Time for alignment in millisecond

STARTUP_SPEED_TARGET

Target speed in RPM during startup in open loop

STARTUP_DUTY_CYCLE

PWM on time in 1/1024 of a PWM period - HF timer in `VOLTAGE_MODE`, REF timer in `CURRENT_MODE`. This parameter is used during the alignment to control the current pushed into the motor.

STARTUP_DIRECTION

Motor rotation direction, 0 for clockwise (positive speed), 1 for counterclockwise (negative speed)

RUN_CONTROL_LOOP_TIME

Periodicity in ms of the loop controlling the HF timer PWMs or the REF timer PWM

RUN_DUTY_CYCLE

PWM on time in 1/1024 of PWM period - HF timer in `VOLTAGE_MODE`, REF timer in `CURRENT_MODE`. The duty cycle of the relevant PWM is ramped linearly from the `STARTUP_DUTY_CYCLE` to the `RUN_DUTY_CYCLE` during the startup in case of the sensors-less algorithm or the sense comparators algorithm with a full ramp.

RUN_ACCELERATION

Acceleration during `RUN` state per control loop time in 1000/1024 RPM/s when the speed loop has been selected in the firmware setup or in 1/1024 of PWM period.

RUN_SPEED_ARRAY_SHIFT

The speed feedback is computed from an averaging of 2^`RUN_SPEED_ARRAY_SHIFT` LF timer periods.

USER_ADC_TRIG_TIME

1/1024 of PWM period elapsed. The ADC measurement is triggered at this time inside the HF PWM period.

5.3.2 Current mode parameters

STARTUP_PEAK_CURRENT

Peak current in mA. This current is used by a macro to compute the startup duty cycle.

STARTUP_SENSE_RESISTOR

Sense resistor in mΩ (See Figure 8. High level architecture block diagram). A definition shall be in the BSP inverter or power board conf template header.

STARTUP_SENSE_GAIN

Sense gain in thousandths (See Figure 8. High level architecture block diagram). A definition shall be in the BSP inverter or power board conf template header.

STARTUP_REF_DIV_RATIO

Reference PWM divider ratio in thousandths (See Figure 8. High level architecture block diagram). A definition shall be in the BSP inverter or power board conf template header.

RUN_HF_TIMER_DUTY_CYCLE

PWM on time in 1/1024 of PWM period elapsed. This is the duty cycle programmed on the HF timer and hence the maximum duty cycle achieved when there is no switch off of the PWM by the current control circuitry (See Figure 8. High level architecture block diagram – current mode circuit).

5.3.3 Torque parameters

Some torque parameters are common to all algorithm and thus already described in Section 5.3.1 Common parameters.

RUN_DUTY_CYCLE_MIN

In RUN state, minimum PWM on time in 1/1024 of PWM period - HF timer in `VOLTAGE_MODE`, REF timer in `CURRENT_MODE`. This is the duty cycle commanded when the potentiometer voltage is minimum or when the PWM interface duty cycle is minimum and above stop threshold.

RUN_DUTY_CYCLE_MAX

In RUN state, maximum PWM on time in 1/1024 of PWM period - HF timer in `VOLTAGE_MODE`, REF timer in `CURRENT_MODE`. This is the duty cycle commanded when the potentiometer voltage is maximum or when the PWM interface duty cycle is maximum.

5.3.4 Speed loop parameters

RUN_SPEED_TARGET

Target speed in RPM during run state. This target can be changed during run-time using the selected user interface.

RUN_SPEED_MIN

In RUN state, minimum speed command in RPM. This is the speed commanded when the potentiometer voltage is minimum or when the PWM interface duty cycle is minimum and above stop threshold.

RUN_SPEED_MAX

In RUN state, maximum speed command in RPM. This is the speed commanded when the potentiometer voltage is maximum or when the PWM interface duty cycle is maximum.

PID_KP

Kp parameter for the PID regulator

PID_KI

Ki parameter for the PID regulator. This parameter is scaled internally with the control loop time:
$pMc->pid\_parameters.ki = ((PID\_KI)*(pMc->control\_loop\_time))$;

PID_KD

Kd parameter for the PID regulator. This parameter is scaled internally with the control loop time:
$pMc->pid\_parameters.kd = ((PID\_KD)/(pMc->control\_loop\_time))$;

PID_SCALING_SHIFT

Common Kp, Ki, Kd scaling for the PID regulator, from 0 to 15.

PID_OUTPUT_MIN

Minimum output value of the PID regulator in tenths of percentage of the HF or REF timer period.

PID_OUTPUT_MAX

Maximum output value of the PID regulator in tenths of percentage of the HF or REF timer period.

5.3.5 Sensor-less algorithm parameters

ALIGNMENT_STEP

Alignment is done to this step.

STARTUP_SPEED_MINIMUM

Minimum speed in RPM for the first step.

STARTUP_ACCELERATION

Acceleration during startup in RPM/s.

VALIDATION_DEMAGN_DELAY

Demagnetization delay in number of HF timer periods elapsed before a first BEMF ADC measurement is processed to detect the BEMF zero crossing.

VALIDATION_ZERO_CROSS_NUMBER

Number of zero crossing events during the startup for closed loop control begin.

VALIDATION_BEMF_EVENTS_MAX

In open loop, maximum number of events where BEMF is over `RUN_BEMF_THRESHOLD_DOWN` during expected BEMF decrease. If this number is reached, the status is changed to `MC_VALIDATION_BEMF_FAILURE` and the `MC_Core_LL_Error` function is called.

VALIDATION_STEPS_MAX

In open loop, maximum number of steps since `MC_STARTUP` beginning. If the number of steps reaches this value, the status is changed to `MC_VALIDATION_FAILURE` and the `MC_Core_LL_Error` function is called.

RUN_LF_TIMER_PRESCALER

LF timer prescaler value used in validation and run states. There is a tradeoff for this value: the higher the prescaler, the lower the speed achievable and the lower the prescaler, the better the accuracy at high speed (minimization of the frequency difference between two consecutive period values).

RUN_BEMF_THRESHOLD_DOWN

BEMF voltage threshold for zero crossing detection when BEMF is expected to decrease. 12bits ADC value with 3.3V full scale. Should be near zero but not too close for some noise immunity. The default value is 200 (161mV).

RUN_BEMF_THRESHOLD_UP

BEMF voltage threshold for zero crossing detection when BEMF is expected to increase. 12bits ADC value with 3.3V full scale. Should be near zero but not too close for some noise immunity. The default value is 200 (161mV).

RUN_BEMF_THRESHOLD_DOWN_ON

Voltage mode only. BEMF voltage threshold during PWM ON time for zero crossing detection when BEMF is expected to decrease. This threshold should be corresponding to a voltage close to half of the motor power supply divided by the resistor bridge ratio (the bridge is controlled by a GPIO as shown in Figure 9. Motor with sensors-less circuit).

RUN_BEMF_THRESHOLD_UP_ON

Voltage mode only. BEMF voltage threshold during PWM ON time for zero crossing detection when BEMF is expected to increase. This threshold should be corresponding to a voltage close to half of the motor power supply divided by the resistor bridge ratio (the bridge is controlled by a GPIO as shown in Figure 9. Motor with sensors-less circuit).

RUN_ZCD_TO_COMM

Zero Crossing detection to commutation delay in 15/128 degrees. In example, 256 corresponds to 30°. A low value helps to reach higher maximum speed. Default value is 200.

RUN_DEMAGN_DELAY_MIN

Demagnetization delay in number of HF timer periods elapsed before a first BEMF ADC measurement is processed to detect the BEMF zero crossing.

RUN_SPEED_THRESHOLD_DEMAG

Speed threshold above which the `RUN_DEMAGN_DELAY_MIN` is applied. This avoid a division in the FW when speed increases and hence reduces microcontroller load.

RUN_DEMAG_TIME_STEP_RATIO

Tenths of percentage of step time allowed for demagnetization. Default value is 270, meaning the demagnetization is 27% of a step rounded to a few HF timer periods.

BEMF_ADC_TRIG_TIME

1/1024 of PWM period elapsed. The ADC measurement is triggered at this time inside the HF PWM period when the BEMF is measured during the PWM OFF time.

BEMF_ADC_TRIG_TIME_PWM_ON

Voltage mode only. 1/1024 of PWM period elapsed. The ADC measurement is triggered at this time inside the HF PWM period when the BEMF is measured during the PWM ON time.

BEMF_PWM_ON_ENABLE_THRES

Voltage mode only. If the HF PWM pulse value is above this value definition * HF timer period / 1024, the time computed from the `BEMF_ADC_TRIG_TIME_PWM_ON` is used to trig the ADC.

BEMF_PWM_ON_DISABLE_THRES

Voltage mode only. If the HF PWM pulse value is below this value definition * HF timer period / 1024, the time computed from the BEMF_ADC_TRIG_TIME is used to trig the ADC.

5.3.6 Hall-sensors algorithm parameters

ALIGNMENT_FORCE_STEP_CHANGE

If not equal to 0, force a step change when the motor does not start with the value reported by the hall sensors

VALIDATION_HALL_STATUS_DIRECTx_STEPn

X = [0,1] (direction), n= [1-6] (step). The FW uses this table to select the next step position.

RUN_HALL_INPUTS_FILTER

Setting for input capture digital filter. The digital filter is made of an event counter in which N consecutive events are needed to validate a transition on the output.

RUN_COMMUTATION_DELAY

Delay between a hall sensors status change and a step commutation with an LF timer counter resolution.

RUN_COMMUTATION_ERRORS_MAX

Maximum number of consecutive wrong commutations. When this number is exceeded, the FW status is changed to `MC_RUN_WRONG_STEP_FAILURE` and the `MC_Core_LL_Error` function is called.

RUN_STAY_WHILE_STALL_MS

While the motor is stalled, stay in MC_RUN state during this value in ms. If the motor is stalled beyond this time, the firmware status is changed to `MC_LF_TIMER_FAILURE` and the `MC_Core_LL_Error` function is called.

5.3.7 Sense comparators algorithm parameters

ALIGNMENT_STEP

Alignment is done to this step

STARTUP_SPEED_MINIMUM

Minimum speed in RPM for the first step

STARTUP_ACCELERATION

Acceleration during startup in RPM/s

STARTUP_FULL_RAMP

0 to shorten the startup ramp as much as possible, 1 to execute the startup ramp until the startup speed target is reached.

VALIDATION_STEPS_MAX

In open loop, maximum number of steps since `MC_STARTUP` beginning. If the number of steps reaches this value, the status is changed to `MC_VALIDATION_FAILURE` and the `MC_Core_LL_Error` function is called.

RUN_LF_TIMER_PRESCALER

LF timer prescaler value used in validation and run states. There is a tradeoff for this value: the higher the prescaler, the lower the speed achievable and the lower the prescaler, the better the accuracy at high speed (minimization of the frequency difference between two consecutive period values).

RUN_ZCD_TO_COMM

Zero Crossing detection to commutation delay in 15/128 degrees. In example, 256 corresponds to 30°. A low value helps to reach higher maximum speed. Default value is 200.

RUN_ZC_READ_GUARD_TIME_AFTER_PWM_EDGE_ZC_TIME_NS

Time in nanoseconds after the HF timer trigger output rising edge. When this time is elapsed the `MC_Core_ZeroCrossingRead` function is called to check if there is a valid zero-crossing. See Figure 24. Example of MC_Core_ZeroCrossingRead timing.

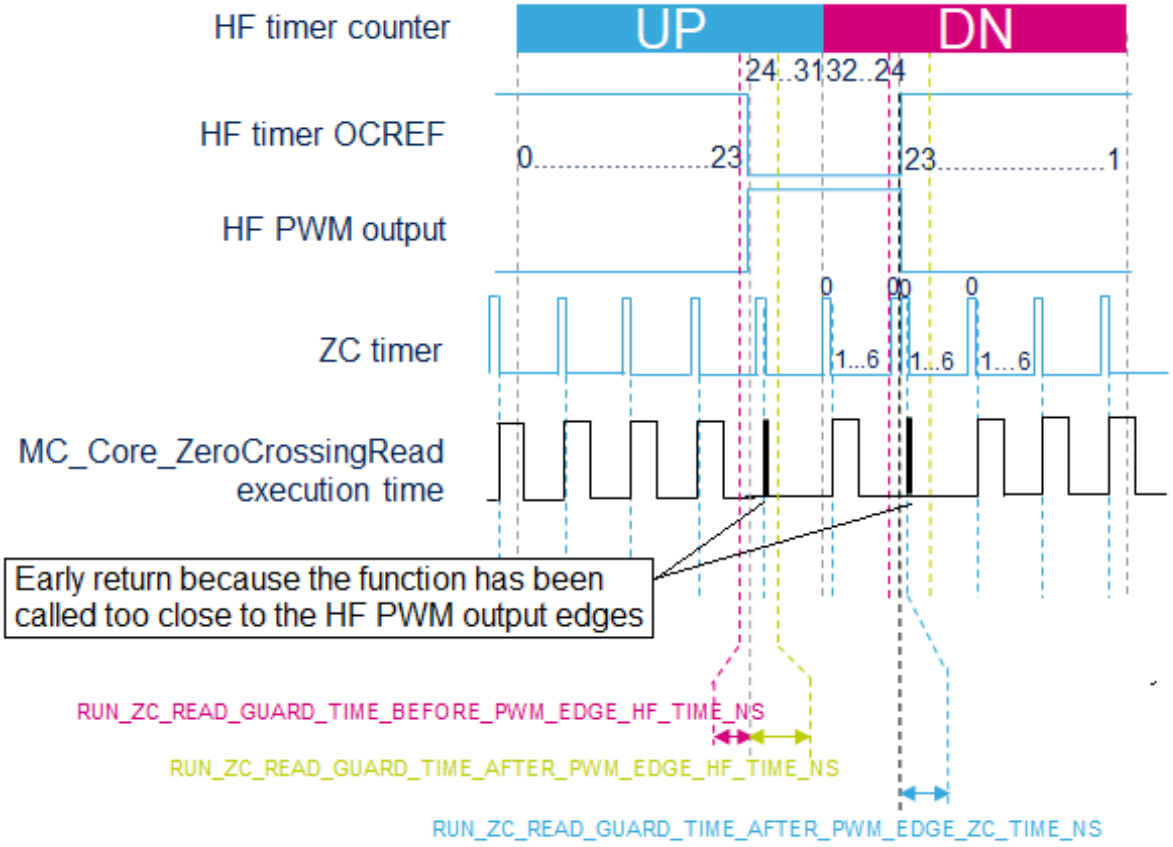
RUN_ZC_READ_GUARD_TIME_AFTER_PWM_EDGE_HF_TIME_NS

Time in nanosecond after the HF PWM output edge from which the MC_Core_ZeroCrossingRead function will not return without having check if there is a valid zero-crossing. See Figure 24. Example of MC_Core_ZeroCrossingRead timing.

RUN_ZC_READ_GUARD_TIME_BEFORE_PWM_EDGE_HF_TIME_NS

Time in nanosecond before the HF PWM output edge until which the MC_Core_ZeroCrossingRead function will not return without having check if there is a valid zero-crossing. See Figure 24. Example of MC_Core_ZeroCrossingRead timing.

Figure 24. **Example of MC_Core_ZeroCrossingRead timing**



Revision history

Table 2. **Document revision history**

Date	Version	Changes
13-Jul-2020	1	Initial release.

Contents

List of figures

List of tables