Getting started with the motor control six-step firmware example for STEVAL-PTOOL2V1

Introduction

The STSW-PTOOL2V1 firmware provides low voltage three-phase brushless DC motor control with the STEVAL-PTOOL2V1 reference design board based on the advanced STSPIN32F0252 BLDC controller with STM32 MCU.

The package includes a sample implementation to drive a BLDC motor with Hall sensors position feedback. It is preconfigured for a trapezoidal driving technique and speed loop with constant PWM duty cycle, known as voltage mode. External potentiometer support allows run-time variation of the target speed.

This software is based on the STM32CubeHAL hardware abstraction layer for the STM32 microcontroller. Motor parameters and speed loop controller parameters are easily accessed and edited in the user configuration file.

The package can be easily downloaded onto the STSPIN32F0252 controller via SWD connection.

1 STSW-PTOOL2V1 firmware package

1.1 Code architecture

The STSW-PTOOL2V1 firmware package supports the STEVAL-PTOOL2V1 board but can be adapted to other boards embedding the STSPIN32F0252 SIP or variant.

To correctly work with the board, the six-step library scalar firmware has to be configured for the STSPIN32F0252 3-phase controller to drive a BLDC motor in open and closed loop.

The library features the following control modes:

- sensorless voltage mode
- sensorless current mode
- voltage mode with Hall effect sensor feedback
- current mode with Hall effect sensor feedback

Important: The STSW-PTOOL2V1 firmware package includes the last two modes only.

Drivers abstract low-level hardware information allowing the middleware components and applications to fully manage the STSPIN32F0252 through a complete set of APIs which send commands to the motor driver. The package includes, in the Projects subfolder, an application example for the STEVAL-PTOOL2V1 board to drive a medium or low voltage three-phase BLDC motor and several motor control parameter files to be used directly with the corresponding motor or as templates for similar ones.

The firmware library and example are written in C programming language and use the embedded STM32Cube HAL abstraction-layer or optimized access to STM32F031 resources. To use the library, a basic knowledge of C programming language, 3-phase motor drives and power inverter hardware is required. In-depth know-how of STM32 functions is required only to customize existing modules and add new ones for a thorough application development. The supported IDE is IAR Workbench.

The firmware project has been developed using the STM32CubeMX configuration tool.

The code architecture is based on STM32Cube Ecosystem and uses the STM32Cube package related to the inverter board MCU.

The six-step firmware package consists of a main STSW-PTOOL2V1-HS folder containing Drivers, Middlewares and Projects subfolders.

1.2 Package folders

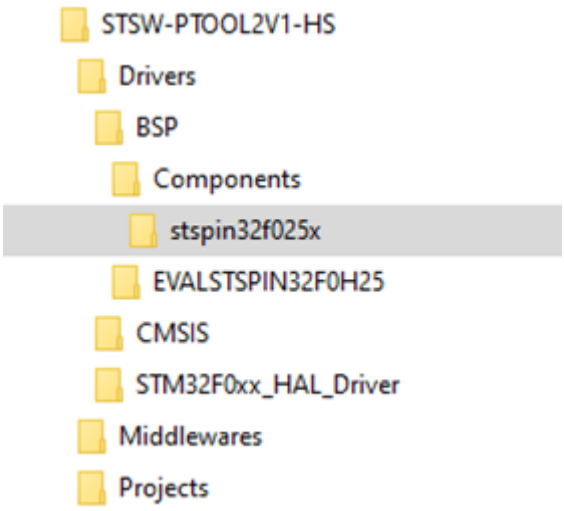
1.2.1 Drivers

This folder contains the source code of the STSPIN32F0252 series components, the EVALSTSPIN32F0H25 board (the board series generic name), STM32Cube HAL and CMSIS drivers.

The interface file for the component drivers contains very specific structures for each component. The hardware resource mapping is handled by the customization of the six-step middleware configuration templates and the mapping from the STM32CubeMX project.

The interface file for the board drivers contains definitions specific to the board, such as resistor values or ADC channel redefinition corresponding to board functions (for example, voltage reading on a potentiometer to set a speed command).

Figure 1. Drivers folder content



1.2.2 Middlewares

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 (i.e., 6step_hs_vm_spdlp_spr.c).

There is also a set of files dedicated to the serial interface (UART): 6step_com.c, 6step_com_sl.c, 6step_com_hs.c, 6step_com_sc.c. The UART is present but not enabled in the reference software. The code for the other available interfaces is embedded in the first cited files (i.e. 6step_hs_vm_spdlp_spr.c).

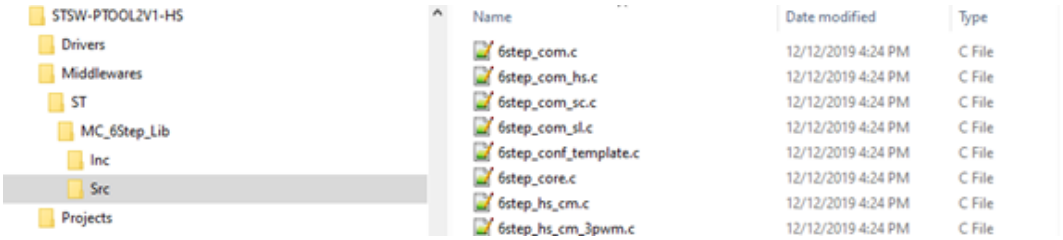
The middleware contains template source and header files to be copied and renamed without “_template” into the user project. The files can be customized according to the user application.

The 6step_conf_template.c file contains:

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

The 6step_conf_<sensing method>_<drive mode>_template.h, 6step_conf_<sensing mode>_<drive mode>_spdlp_template.h are middleware template files containing the definitions for the example application motor control parameters.

Figure 2. **Middlewares folder content**

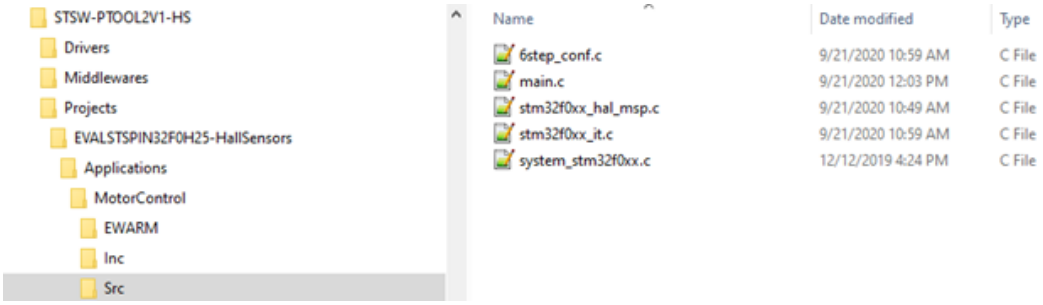


1.2.3 Projects

The “Projects” folder is related to the inverter board hardware configuration. The project works with IAR Embedded Workbench version 8.30.1 or later.

STM32CubeMX has been used to configure all the peripherals needed in the motor control application as well as to map all hardware resources. The ioc file, the src and inc folders generated by this tool have been copied into the Projects\EVALSTSPIN32F0H25-HallSensors\Applications\MotorControl folder.

Figure 3. **Projects folder content**



The “Src” folder contains:

- 6step_conf.c: configuration file where functions can be customized to adapt the six-step middleware to a specific hardware
- main.c : main file containing the calls to the functions which:
  - initialize the MCU peripherals (timers, ADCs, GPIOs, etc.)
  - initialize the middleware
  - configure the communication interface according to the user choice

  This file also contains the “6step_core.h” to link the user level to the motor middleware and make all the API functions available.
- stm32f0xx_hal_msp.c: standard STM32Cube HAL file for MCU peripheral initialization and de-initialization
- stm32f0xx_it.c: STM32Cube HAL file for MCU interrupt request and handling function (the interrupt handler implementation is customized in the 6step_conf.c file)
- system_stm32f0xx.c: standard CMSIS system source file

The “Inc” folder contains:

- 6step_conf.h: a set of definitions that select the six-step middleware building the sensing method, control mode and other features. This file is a customization for the user application of the middleware 6step_conf_template.h file
- 6step_conf_<sensing method>_<drive mode>.h, 6step_conf_<sensing method>_<drive mode>_spdlp.h: customization files of the corresponding middleware _template.h files. They contain the definitions for the control parameters of the user application motor.
- EVALSTSPIN32F0H25_conf.h: customization of the corresponding Drivers BSP _template.h file. It has to be modified to include header files related to the control board chosen by the user or to the inverter board MCU. It can be also modified to take into account power board or inverter board modifications, such as changes of components
- main.h: containing definitions created by STM32CubeMX used for the MCU peripheral initialization

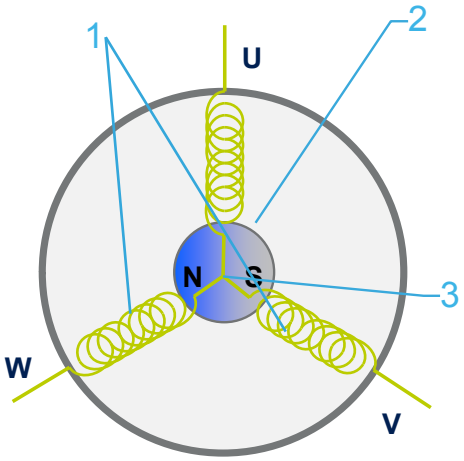
- stm32f0xx_hal_conf.h, stm32f0xx_it.h: header files generated by STM32CubeMX

2 Brushless DC motor basics

A brushless three-phase motor consists of a fixed part (stator) made of a set of three windings and a mobile part containing an internal permanent magnet (rotor) which may have several pole pairs evenly distributed around the stator.

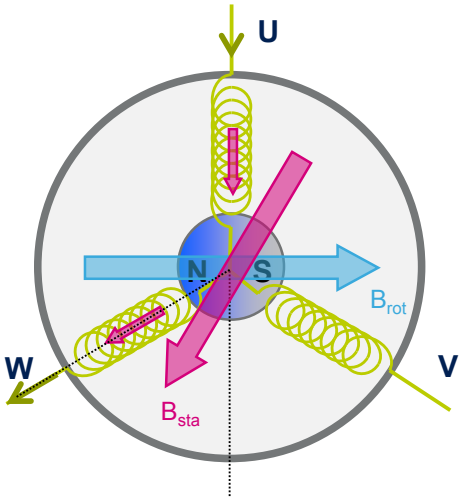
Figure 4. Motor stator and rotor arrangement

1. Stator composed by three coils (or phases), positioned at 120° from each other
2. Permanent magnet generating the rotor magnetic field
3. Windings connected by one side. The sum of the currents is zero



In six-step driving, the electrical cycle is split into six commutation steps. For each step, the bus voltage is applied to one of the three motor phase windings while the ground is applied to a second winding, generating low current in these two windings as well as a stator magnetic field as shown in the figure below. The third winding remains open.

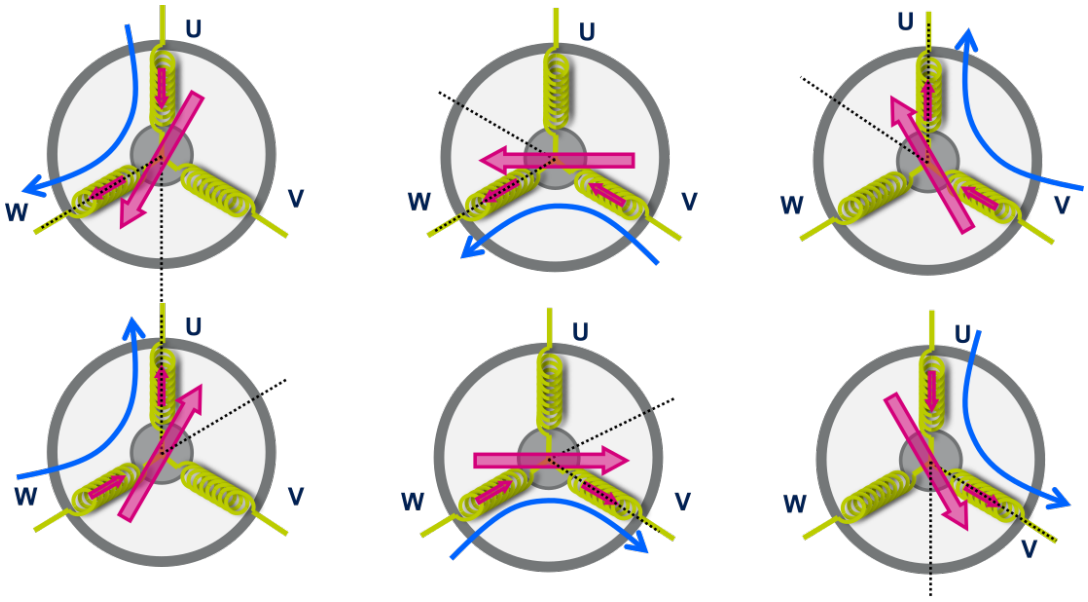
Figure 5. Motor stator and rotor magnetic fields



The rotor magnetic field is always present and is generated by a permanent magnet. When current flows from a motor phase to another, the magnetic fields merge generating the stator field.

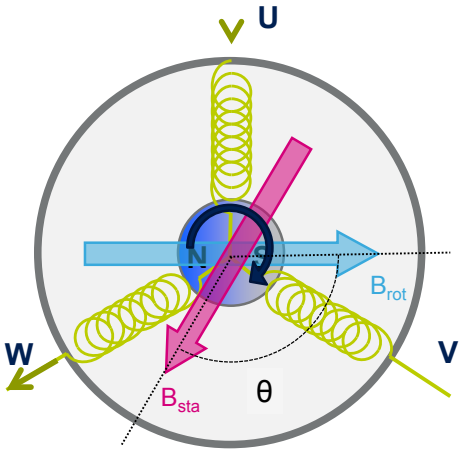
The subsequent steps are executed in the same way but the bus voltage and the ground are applied to different motor phase windings to generate a rotating stator magnetic field with six different discrete positions as shown below.

Figure 6. **Motor stator magnetic field 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°, as shown in the figure below. The step commutation ensures the torque is always close to 90°.

Figure 7. **Motor stator magnetic field discrete positions**



The torque applied to the motor is proportional to the sine of the load angle (θ). When the rotor magnetic field approaches the stator magnetic field, the torque is reduced.

Important: *To keep the motor in motion it is necessary to change the stator magnetic field direction.*

3 Six-step firmware algorithms

3.1 Overview

The six-step firmware senses the position of the motor rotor with an electrical 60° accuracy. It computes the time of the next step commutation and duty cycle for the PWM signals which control the amount of current pushed into the motor by using a triple half bridge with power transistors.

The six-step firmware is composed of a set of components running under different tasks which interact with each other, with the motor hardware and the electronic circuitry.

3.1.1 Components

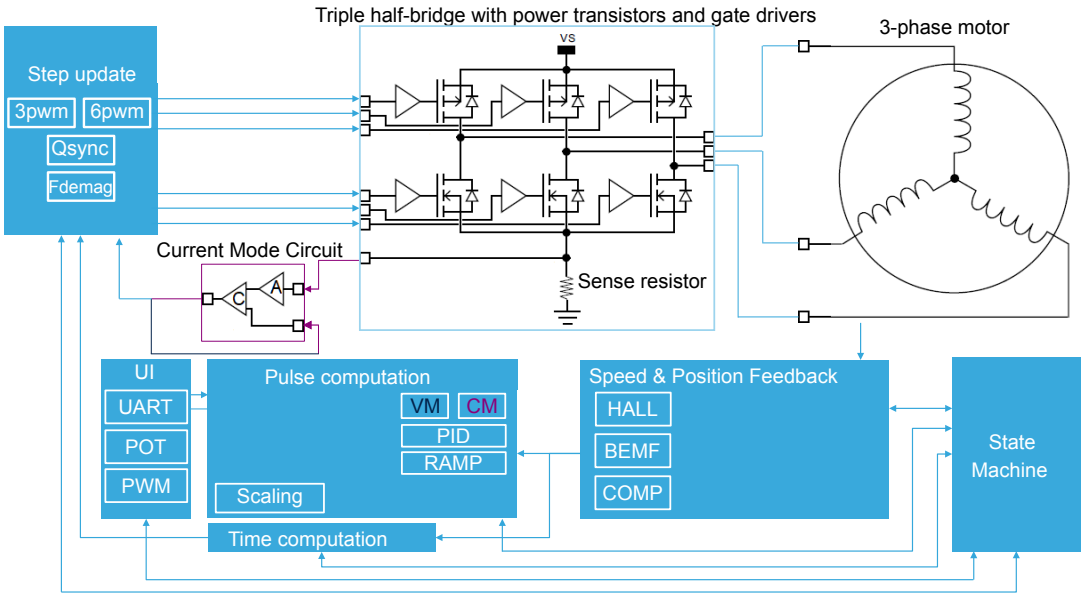
The six-step firmware can be split into:

- speed and feedback component: BEMF for sensorless algorithm, HALL for Hall sensor algorithm or COMP for sense comparator algorithm is used. In the STSW-PTOOL2V1 example only Hall sensor feedback is implemented
- pulse computation component: 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 set point ramping (RAMP) algorithm is used when the set point ramping feature is present
- time computation component
- user interface component: a serial communication (UART), a potentiometer voltage (POT) or a PWM pulse duration (PWM) is used to control the motor. In the STSW-PTOOL2V1 example only the potentiometer input is implemented
- step update component: three-PWM (3pwm) or six-PWM (6pwm) driving can be implemented, but the STEVAL-PTOOL2V1 board is driven in 6pwm mode only. Synchronous or quasi-synchronous (Qsync) rectification and normal or fast demagnetization (Fdemag) can be used
- state machine component

There are different implementations of a component depending on the six-step firmware setup.

Each component operation depends on the state machine status and can change it as well.

Figure 8. Six-step firmware high level architecture block diagram



3.1.2 Tasks

The six-step firmware is based on interrupts and runs several tasks which use the components described in Section 3.1.1 Components.

The supported tasks are:

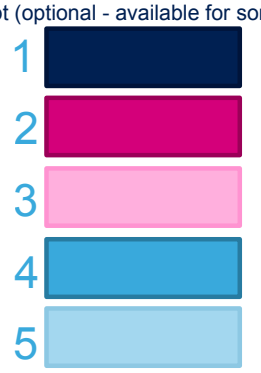
- High frequency (HF) task
- Medium frequency (MF) task
- Low frequency (LF) task

Tasks are prioritized according to the six-step firmware setup.

The figure below shows how tasks are identified by color in the next section diagrams.

Figure 9. Color code for tasks

1. Pre-requisites
2. Interrupt entry point
3. Execution on interrupts (optional - available for some firmware setup only)
4. Part of the code executed during the interrupt
5. Part of the code executed during the interrupt (optional - available for some firmware setup only)



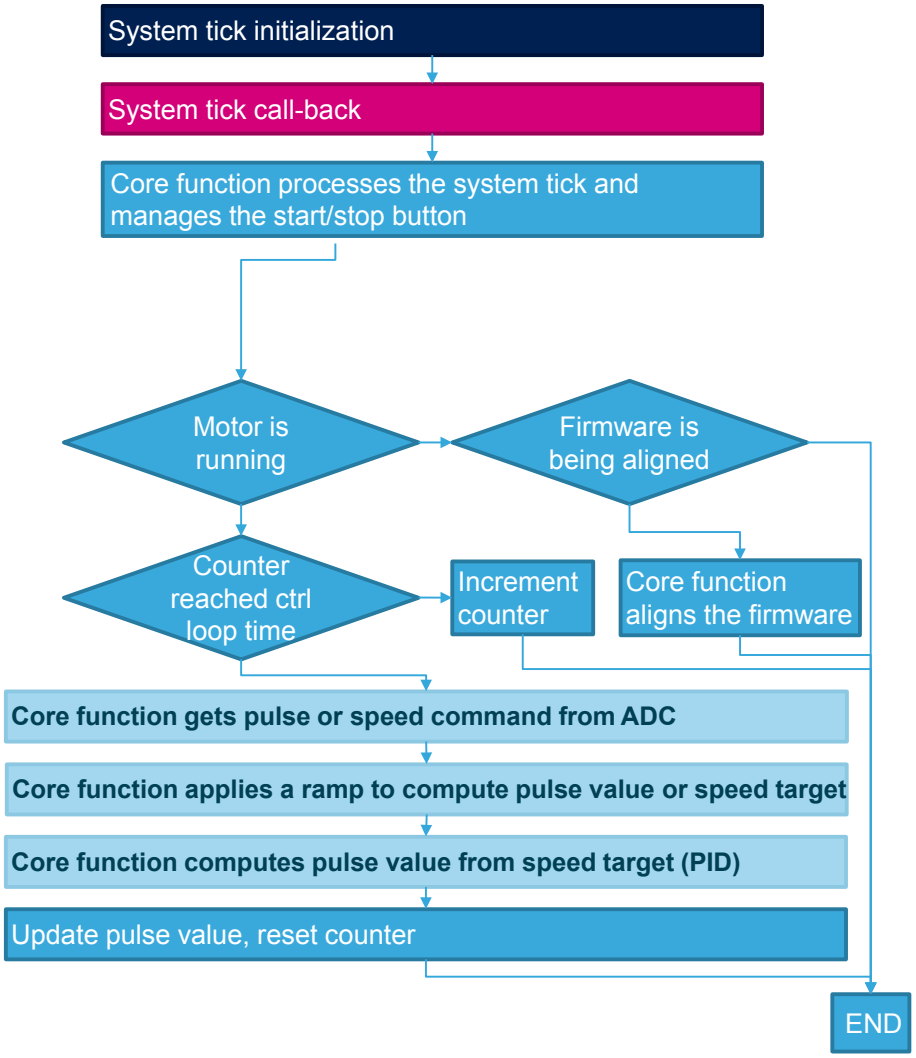
3.1.2.1 *Medium frequency task*

This is a low priority task. It occurs every 1 ms and manages the duty cycle (the pulse) of the HF PWMs.

If the firmware has been built with the POT component, the medium frequency task uses the ADC to infer a pulse or speed command.

The medium frequency task manages the speed loop by calling a function with a PID regulator.

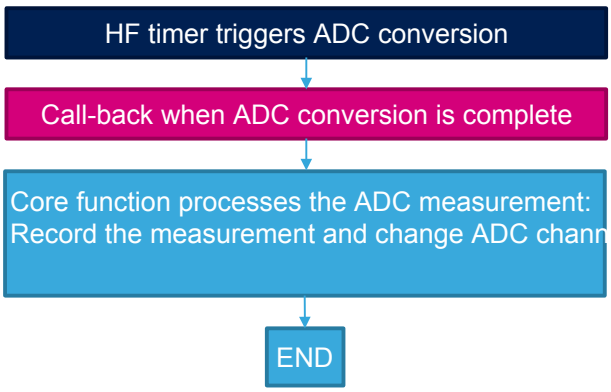
Figure 10. **Medium frequency task diagram**



3.1.2.2 *High frequency task*

This is a low priority task. It performs ADC measurements.

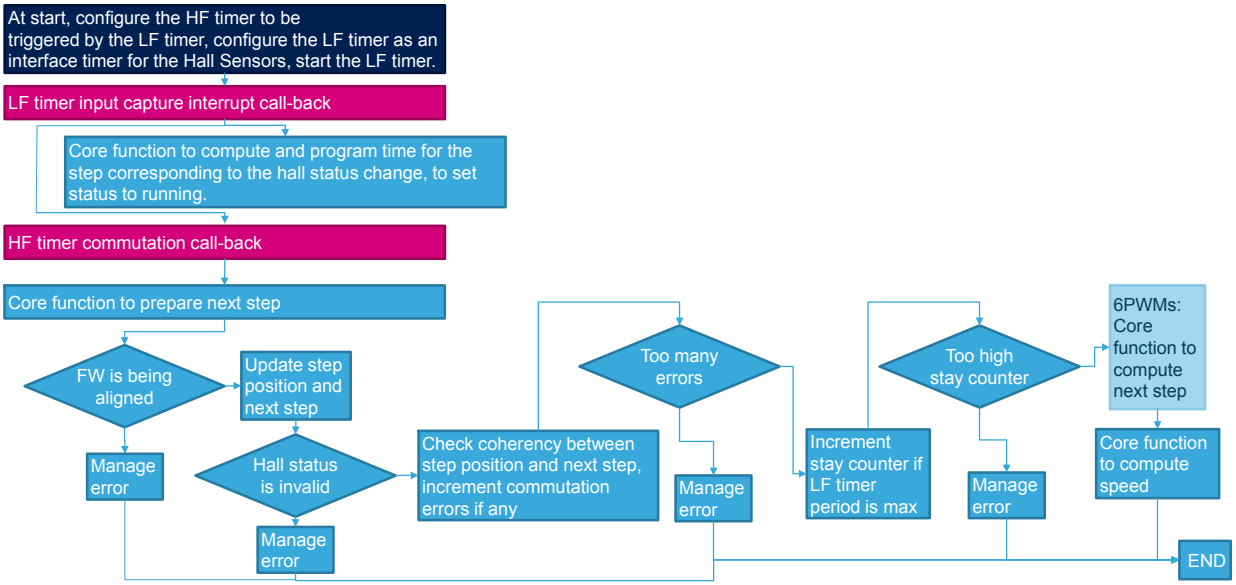
Figure 11. **High frequency task diagram**



3.1.2.3 *Low frequency task*

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

Figure 12. **Low frequency task diagram**



3.2 Driving modes

3.2.1 Voltage

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 are not used.

3.2.2 Current

In this driving mode, the Current Mode circuit and the sense resistor shown in Figure 8 are used. The Current Mode circuit has an A amplifier and a C comparator.

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

The comparator output 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.

3.3 Hall sensor algorithm

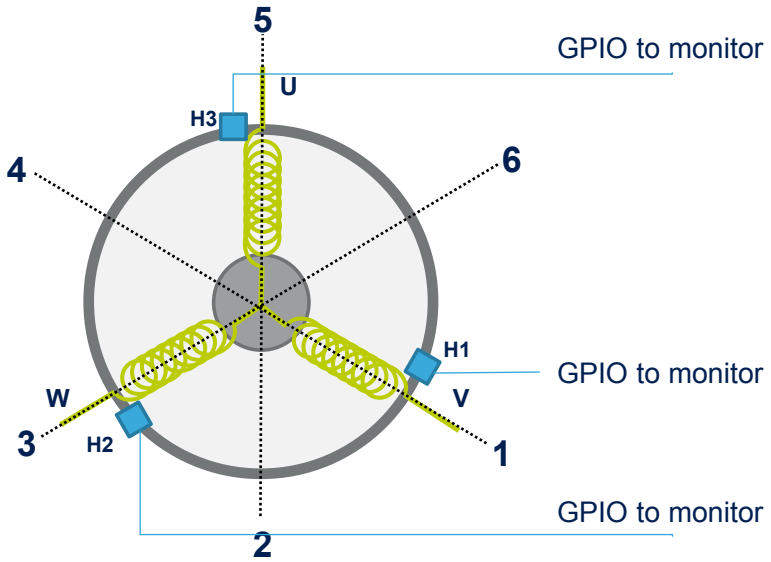
The three Hall effect sensors detect the rotor position returning digital signal values.

The firmware uses a configurable truth table to infer the next step number according to the Hall status which is a pondered sum of Hall effect sensor digital signal values as shown below.

Table 1. Six-step firmware truth table

#	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

Figure 13. Motor with Hall effect sensors



When the firmware receives a command to start the motor, the Hall sensors infer the rotor position and the firmware programs the next step according to the direction chosen by the user.

If the motor does not move enough to change the Hall status in the alignment time, the firmware signals an error or tries to move to a different step depending on the firmware setup.

If the rotor moves enough to change the Hall status, the motor is assumed to be running. It keeps running until you choose to stop it or an error occurs.

If the motor is stalled (for instance, someone is holding the rotor), the firmware tries to reach the next step for some time: if time elapses and the motor is still stalled, the firmware goes into error state and stops pushing current into the motor by actually stopping it. If, while running, the step reached is not the next step nor the previous one for several consecutive times, the firmware goes into error state.

For most motors with Hall sensors, the step commutation occurs as soon as the Hall status changes. In the six-step firmware, the time between the Hall status change and the step commutation can be customized.

3.3.1 MCU hardware requirements and use

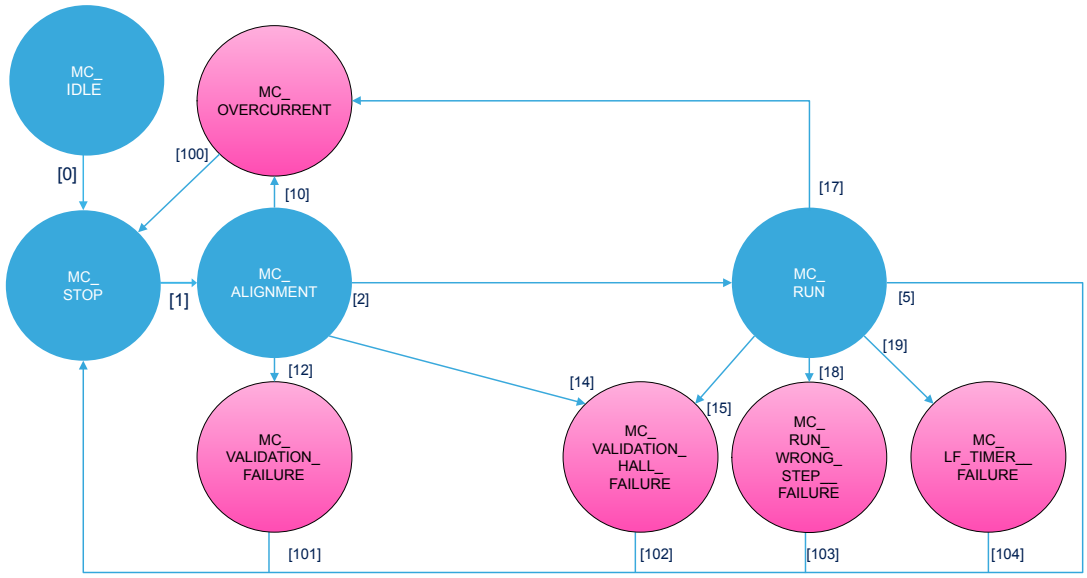
To use the Hall sensor algorithm, you need the following resources:

- a four-channel high frequency (HF) timer to generate PWM signals to trigger the ADC. Each PWM channel has its complementary counterpart or three GPIOs are available to generate the enable signal for each half bridge low side
- three GPIOs connected to the digital Hall sensors
- an interfacing low frequency (LF) timer to capture and time stamp the level transitions on the GPIOs. It is used also to generate a step commutation by changing the configuration of the HF timer
- In case of current mode, an additional reference (REF) timer to generate the PWM to build the reference voltage
- an ADC with several channels for measurements. The ADC can be triggered by a timer 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 the HF timer or a timer synchronized with the HF timer

3.3.2 State machine

The six-step firmware has several states depending on the execution schedule, on events and on user intervention. In the figure below, the light blue identifies the expected states during normal operation while the pink identifies error states.

Figure 14. Six-step Hall sensor state machine



3.3.2.1 *Normal states*

- `MC_IDLE`: one-time state at the beginning of the six-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 deduces the step number. Then, it tries to move the motor to the next step according to chosen direction. When the motor does not move enough to change the step status during the alignment time, the firmware goes into 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

3.3.2.2 *Error states*

When an abnormal behavior is detected, the firmware goes from normal to error state.

The firmware calls an error function in the user project. The default implementation of this function, provided by the middleware, 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_VALIDATION_HALL_FAILURE: the Hall status is invalid
- MC_RUN_WRONG_STEP_FAILURE: the current step is not the expected step nor the previous one
- MC_LF_TIMER_FAILURE: the motor is stalled or running too slowly

3.3.2.3 *Transitions*

Referring to Figure 14, the conditions for the numbered normal transitions are:

- [0] → the firmware has been initialized
- [1] → the user has sent a command to the firmware to start the motor
- [2] → the motor moved, changing the Hall status to a valid value
- [5] → the user has sent a command to the firmware 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 it moved but 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 not 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 a 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 six-step configuration source file. By default, if the serial interface is available, it reports the status and the firmware stops the motor changing the status accordingly

4 Six-step firmware setup

The six-step firmware setup can vary in terms of sensing method (the means used to monitor the position of the rotor) and driving modes (the means used to control the current injection into the motor phases).

To build the six-step middleware for your application, you can select:

- Driving mode: voltage (see Section 3.2.1) or current (see Section 3.2.2)
- Sensing method: sensorsless, Hall sensors or sense comparators. Only the Hall sensor method is supported by the firmware package (see Section 3.3)
- Rectification: synchronous or quasi-synchronous. In synchronous rectification, the current decay in the motor is performed using the Power MOSFET R_{ds} ON resistance, whereas quasi-synchronous conversion, the MOSFET body diode is used.
- Speed loop: enabled or disabled. When the speed loop is enabled, the HF PWM duty cycle, which drives the current injection into the motor inductors, or the REF PWM duty cycle, used as current reference, is controlled by a PID regulator output. Otherwise, the duty cycle is directly controlled by the user.
- Set point ramping: enabled or disabled. If enabled, the user command for the speed reference, for the HF or REF duty cycle is linearly ramped from the value at the time the command is issued to the commanded value. The speed reference or the HF or REF duty cycle is updated each control loop time with the acceleration.
- Three-PWM: enabled or disabled. When enabled, three PWM signals with three enabling signals (GPIOs with constant level during a step) are needed to drive the motor. Otherwise three PWM signals with three complementary ones are needed.
- Fast demagnetization: enabled or disabled. When enabled, the MOSFET side, where the driving PWM is fed, is changed at each step commutation from high to low side or from low to high side to obtain the highest inverse voltage on the motor floating inductor and so the lowest time needed for demagnetization. This allows monitoring the BEMF zero-crossing in the sensorless algorithm.
- User interface: UART (serial communication), potentiometer (ADC measurement) or PWM. Only the potentiometer is actually implemented in the firmware package.
  - When the UART interface is selected, a terminal window uses predefined functions to command 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.
  - When the potentiometer interface is selected, the motor operating point is a linear function of the voltage read on the potentiometer through the ADC. A switch triggers the motor start and stop.
  - When the PWM interface is selected, the duty cycle of an interface timer commands the motor start, stop and operating point, as specified in the following ESC communication protocols:
    - 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 APIs

The STSW-PTOOL2V1 firmware features a set of functions to build and customize your own application.

By default, the first parameter of every function is a motor control handle (`MC_Handle_t *pMc`) corresponding to the motor controlled. This allows controlling several motors through the same microcontroller.

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

5.1 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

1st parameter: uint32_t *pMcCom — Pointer cast from `UART_HandleTypeDef` handle definition from HAL.

5.2 MC_Core_AssignTimers

This function assign timers to the motor control structure. The pHfTimer and pLfTimer must be non-null pointers as they are mandatory for motor operation indipendently of the chosen algorithm.

The pRefTimer is mandatory if the driving mode is Current. The pRefTimer can also be used in the Voltage Mode to trig the ADC. The pZcTimer is mandatory for the sense comparator algorithm. If a timer is not used, the NULL pointer has to be used for its corresponding parameter.

Number of parameters: 1+4

2nd parameter: uint32_t *pHfTimer — Pointer to the start of the structure of the high frequency timer used to generate the PWM signals to drive the motor phases.

3rd parameter: uint32_t *pLfTimer — Pointer to the start of the structure of the low frequency timer used to generate a step commutation.

4th parameter: uint32_t *pREFTimer — Pointer to the start of the structure of the reference timer used to generate the voltage reference in current mode and eventually an ADC trigger.

5th parameter: uint32_t *pZCTimer — Pointer to the start of the structure of the zero-crossing timer used to recognize the occurrence of the BEMF voltage zero crossing with the help of sense comparators.

5.3 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 has to be used to compute a speed or a duty cycle command for the motor. A user measurement has to be linked to an ADC and an ADC channel via the `MC_Core_ConfigureUserAdcChannel` function.

Number of parameters: 1+4

2nd parameter: MC_UserMeasurements_t — User measurement to build the speed command or the duty cycle command.

5.4 MC_Core_BackgroundTask

This function is used only when the PWM is selected as user interface in the firmware setup and has to be called in the main program “while (1)” loop.

It manages the lowest priority for 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

5.5 MC_Core_ConfigurePwmInterface

This function is used only when the PWM is selected as user interface in the firmware setup and has to be called in the main program before the `MC_Core_Init` and the `MC_Core_BackgroundTask`. It configures the PWM interface.

Number of parameters: 1+9

2nd parameter: uint32_t *pIfTimer — Pointer to the start of the timer used for the PWM interface.

3rd parameter: 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`.

4th parameter: uint16_t PulseUSForMaximumThrottle — Captured pulse duration in µs for max. command.

5th parameter: uint16_t PulseUSForMinimumThrottle — Captured pulse duration in µs for min. command.

6th parameter: uint32_t TimerPulseMaxValue — Pulse value of the timer used for the PWM interface.

7th parameter: uint8_t ArmingCnt — When the motor is stopped, the number of consecutive pulses below the min. command duration needed to check pulses in the min. to max. command range to start the motor.

8th parameter: uint8_t StartCnt — Number of consecutive pulses above the min. command duration need to start the motor.

9th parameter: uint8_t StopCnt — Number of consecutive pulses above the min. command duration need to stop the motor.

10th parameter: uint16_t NoSignalStopMs — Time in ms without pulse after which the motor stops.

5.6 MC_Core_ConfigureUserAdc

This function configures the user ADC by assigning a timer and one of its channels to it. The assigned timer channel triggers the ADC conversion. The period of the assigned timer is set equal to the period of the HF timer.

Number of parameters: 1+3

2nd parameter: uint32_t *pTrigTimer — Pointer to the timer used to trig the ADC.

3rd parameter: uint16_t TrigTimerChannel — Channel of the timer used to trig the ADC. This parameter has to be cast from a `TIM_LL_EC_CHANNEL` low layer definition such as `LL_TIM_CHANNEL_CH4`.

4th parameter: uint8_t NumberofUserChannels Number of channels for user measurements (max. 5).

5.7 MC_Core_ConfigureUserAdcChannel

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

Number of parameters: 1+4

2nd parameter: uint32_t *pAdc Pointer to the ADC to be selected.

3rd parameter: uint32_t AdcChannel ADC channel to be selected. It uses 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).

4th parameter: uint32_t SamplingTime ADC sampling time to be selected. It uses 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).

5th parameter: uint16_t MC_UserMeasurements_t User measurement to map to ADC channel.

5.8 MC_Core_ConfigureUserButton

This function configures a user button and its debounce time. The button can be used as start/stop button or for other functions. 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

2nd parameter: uint16_t ButtonPin Cast from a GPIO pin definition corresponding to the button. It uses a definition from the main.h file where the definitions generated with STM32CubeMX are located (i.e., USER1_LED_BUTTON_Pin GPIO_PIN_0).

3rd parameter: ButtonDebounceTimeMs Button debounce time in ms.

5.9 MC_Core_GetGateDriverPwmFreq

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

Number of parameters: 1+0

5.10 MC_Core_GetMotorControlHandle

This function returns the motor control handle corresponding to the `MotorDeviceId`.

Number of parameters: 1

1st parameter: uint8_t *MotorDeviceId A number from 0 to NUMBER_OF_DEVICES-1.

5.11 MC_Core_GetSpeed

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

Number of parameters: 1+0

5.12 MC_Core_GetStatus

This function returns the motor status (MC_Status_t type) stored in the motor control structure.

```
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;
```

Number of parameters: 1+0

5.13 MC_Core_Init

This function must be called after all the timer assignments, the ADC, the measurement 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, incrementing the ID value by 1 for each subsequent call. It initializes the motor control structure content with the device status and values which are not expected to change during runtime, such as the number of motor pole pairs. Moreover, it calls the `MC_Core_Reset` function to initialize values that might change during runtime. It also starts the PWM interface if present.

Number of parameters: 1+0

5.14 MC_Core_Reset

This function initializes values that might change during runtime. 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.

Number of parameters: 1+0

5.15 MC_Core_SetAdcUserTrigTime

This function sets the ADC trig time inside the HF timer PWM period.

Note: The timer used to trig the ADC is the HF timer or a timer that has the same period and is synchronized with it.

Number of parameters: 1+1

2nd parameter: uint32_t AdcUserTrigTimeToSet

5.16 MC_Core_SetDirection

This function sets the direction of the motor rotation.

Number of parameters: 1+1

2nd parameter: uint32_t DirectionToSet Direction to set where 0 is for positive speed (clockwise direction) and 1 for negative speed (counterclockwise direction).

5.17 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 the HF timer for voltage mode driving or the REF timer for current mode driving.

Number of parameters: 1+1

2nd parameter: uint32_t DutyCycleToSet Duty cycle in 1/1024 of PWM period.

5.18 MC_Core_SetGateDriverPwmFreq

This function sets the gate driver PWM frequency. If this function is called while the motor is still moving, 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

2nd parameter: uint32_t FrequencyHzToSet Frequency in Hz to be set.

5.19 MC_Core_SetSpeed

This function sets the speed command which is taken into account in the medium frequency task at every control loop. The previous speed command becomes immediately the speed target value bypassing the acceleration mechanism.

Number of parameters: 1+1

2nd parameter: uint32_t SpeedToSet Speed command to be set in Hz

5.20 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 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

2nd parameter: uint32_t DutyCycleToSet Duty cycle in 1/1024 of PWM period.

5.21 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 preloaded register content to the shadow register), and, if the REF timer exists, programs its pulse value and starts it.

Number of parameters: 1+0

5.22 MC_Core_Stop

This function stops the motor by stopping LF and HF timers. It also stops the ADC interrupts and the REF timer if present. In the sense comparator algorithm, it stops the ZC timer. Moreover, it resets all the motor control variables by calling the `MC_Core_Reset` function.

Number of parameters: 1+0

6 Firmware parameters

6.1 Common parameters

Table 2. STSW-PTOOL2V1 common parameters

Parameter	Description
MOTOR_NUM_POLE_PAIRS	Number of motor pole pairs
ALIGNMENT_TIME	Time for alignment in milliseconds
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 FW 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.

6.2 Current mode parameters

Table 3. STSW-PTOOL2V1 current mode parameters

Parameter	Description
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(1). A definition shall be in the BSP inverter or power board conf template header.
STARTUP_SENSE_GAIN	Sense gain in thousandths (1). A definition shall be in the BSP inverter or power board configuration template header.
STARTUP_REF_DIV_RATIO	Reference PWM divider ratio in thousandths (1). A definition shall be in the BSP inverter or power board configuration 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 ((1).

1. See *Figure 8. Six-step firmware high level architecture block diagram*

6.3 Torque parameters

Table 4. STSW-PTOOL2V1 torque parameters

Parameter	Description
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.

6.4 Speed loop parameters

Table 5. STSW-PTOOL2V1 speed loop parameters

Parameter	Description
RUN_SPEED_TARGET	Target speed in RPM during run state. This target can be changed during runtime 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.

6.5 Hall sensor algorithm parameters

Table 6. STSW-PTOOL2V1 Hall sensor algorithm parameters

Parameter	Description
ALIGNMENT_FORCE_STEP_CHANGE	If not equal to 0, it forces a step change when the motor does not start with the value reported by the Hall sensors
VALIDATION_HALL_STATUS_DIRECT0_STEP1	The value for each VALIDATION_HALL_STATUS_DIRECTx_STEPn definition shall be the Hall status corresponding to step “n” for direction “x”. The firmware uses this table to select the next step position.
VALIDATION_HALL_STATUS_DIRECT0_STEP2	
VALIDATION_HALL_STATUS_DIRECT0_STEP3	
VALIDATION_HALL_STATUS_DIRECT0_STEP4	
VALIDATION_HALL_STATUS_DIRECT0_STEP5	
VALIDATION_HALL_STATUS_DIRECT0_STEP6	
VALIDATION_HALL_STATUS_DIRECT1_STEP1	
VALIDATION_HALL_STATUS_DIRECT1_STEP2	
VALIDATION_HALL_STATUS_DIRECT1_STEP3	
VALIDATION_HALL_STATUS_DIRECT1_STEP4	
VALIDATION_HALL_STATUS_DIRECT1_STEP5	
VALIDATION_HALL_STATUS_DIRECT1_STEP6	
RUN_HALL_INPUTS_FILTER	Setting for input capture digital filter which consists in an event counter in which N consecutive events are needed to validate a transition on the output.
RUN_COMMUTATION_DELAY	Delay between a Hall sensor 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 firmware 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, it stays 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.

Revision history

Table 7. **Document revision history**

Date	Version	Changes
07-Oct-2020	1	Initial release.

Contents

List of figures

List of tables