

---

## Getting started with the motor control six-step firmware example for STEVAL-PTOOL1V1

### Introduction

The [STSW-PTOOL1V1](#) firmware provides low voltage three-phase brushless DC motor control with the [STEVAL-PTOOL1V1](#) reference design board based on the advanced [STSPIN32F0B](#) 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, speed loop controller parameters and Hall sensor decoding table values are easily accessed and edited in the user configuration file.

The package can be easily downloaded onto the [STSPIN32F0B](#) controller via SWD connection.

# 1 STSW-PTOOL1V1 firmware package overview

The [STSW-PTOOL1V1](#) firmware features:

- Sample application to drive a low voltage, 3-phase brushless motor using the [STEVAL-PTOOL1V1](#) motor control board
- Timer to generate PWM phase driving signals and current reference
- Management of parameters such as minimum and maximum speed and direction
- GPIO, PWM and IRQ configuration
- Speed control through potentiometer
- Motor positioning detection through Hall sensors reading and decoding
- Motor control by trigger switch
- Free, user-friendly license terms

The package consists of a main folder embedding three subfolders: Drivers, Middlewares and Projects.

## 1.1 Package folders

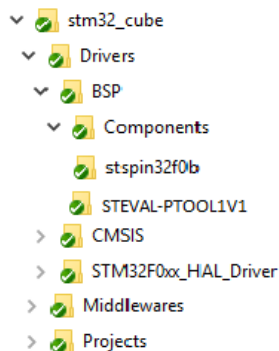
### 1.1.1 Drivers

This folder contains the [STSW-PTOOL1V1](#) source code as well as STM32Cube HAL and CMSIS drivers generated by [STM32CubeMX](#).

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



### 1.1.2 Middlewares

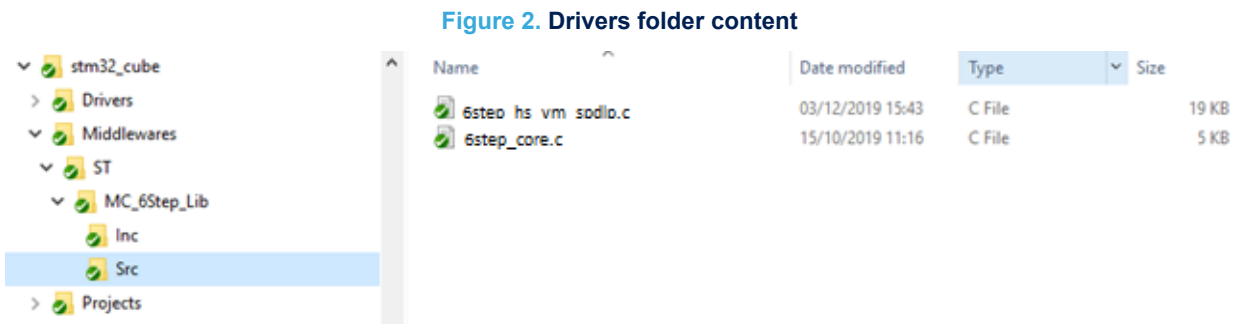
This folder contains the core code of the motor control algorithms. The code supports the control method with Hall sensors and speed loop with constant PWM duty cycle (6step\_hs\_vm\_spdlp.c).

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

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\_core.c file contains the data structure and the definition of the core functions of the algorithm.



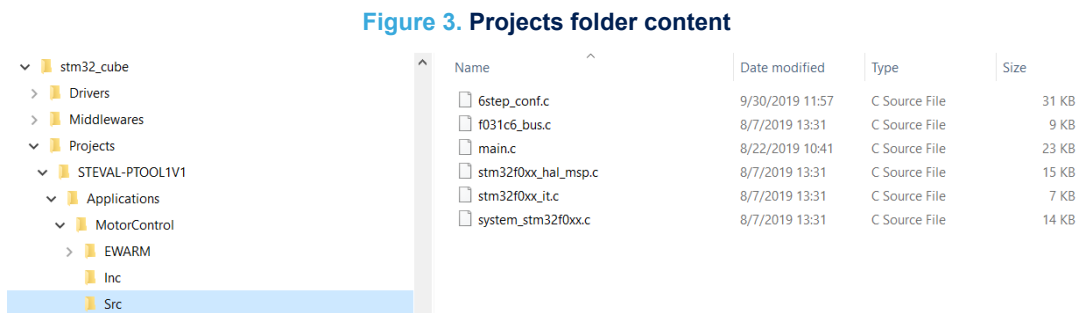
### 1.1.3 Projects

The Projects folder contains a subfolder including the firmware project of the [STEVAL-PTOOL1V1](#). The project has been implemented for:

- IAR Embedded Workbench v8.40.2 or later
- MDK-ARM v5.30.0 or later
- [STM32CubeIDE v1.2.0](#) or later

[STM32CubeMX](#) has been used to:

- configure all the peripherals needed in a motor control application
- map all hardware resources
- generate the src and inc folders copied in the `Projects\STEVAL-PTOOL1V1\Applications\MotorControl` folder



The “Src” folder contains:

- `6step_conf.c`: configuration file where functions can be customized to adapt the six-step middleware to a specific hardware
- `F031c6_bus.c`: optional file containing board specific functions, not handled by the middleware, for communication between the MCU and board components
- `main.c`: main file containing the calls to the functions which:
  - initialize the MCU peripherals (timers, ADCs, GPIOs, etc.)
  - configure the middleware according to the board characteristics
  - initialize the middleware
  - configure the communication interface according to the user choice
- `stm32f0xx_hal_msp.c`: standard STM32Cube HAL file for MCU peripherals initialization and de-initialization
- `stm32f0xx_it.c`: STM32Cube HAL file for MCU interrupt request and handling function
- `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_hs_vm_spdip.h`: holding the definitions for the control parameters of the user application motor
- `f031c6_bus.h`: optional file containing the function prototypes implemented in `f031c6_bus.c`

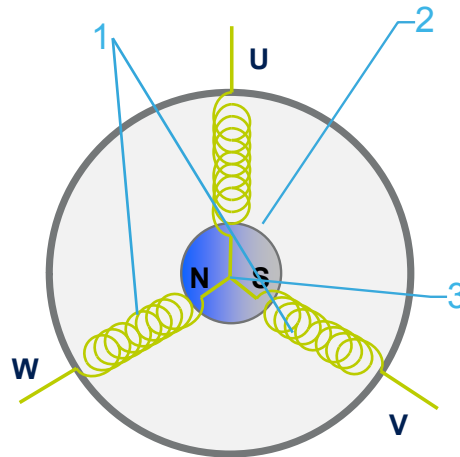
- f031c6\_errno.c: optional file containing the error code for the board specific functions
- steval-ptool1v1\_conf.h: configuration file that can be customized to take into account modifications of the board (for example, changes of a component value)
- main.h: containing definitions created by [STM32CubeMX](#) 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.

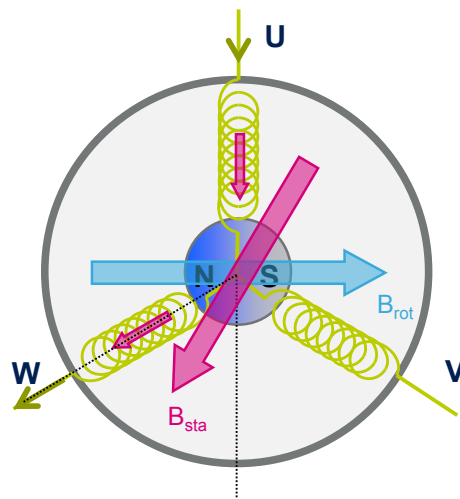
**Figure 4. Motor stator and rotor arrangement**

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.

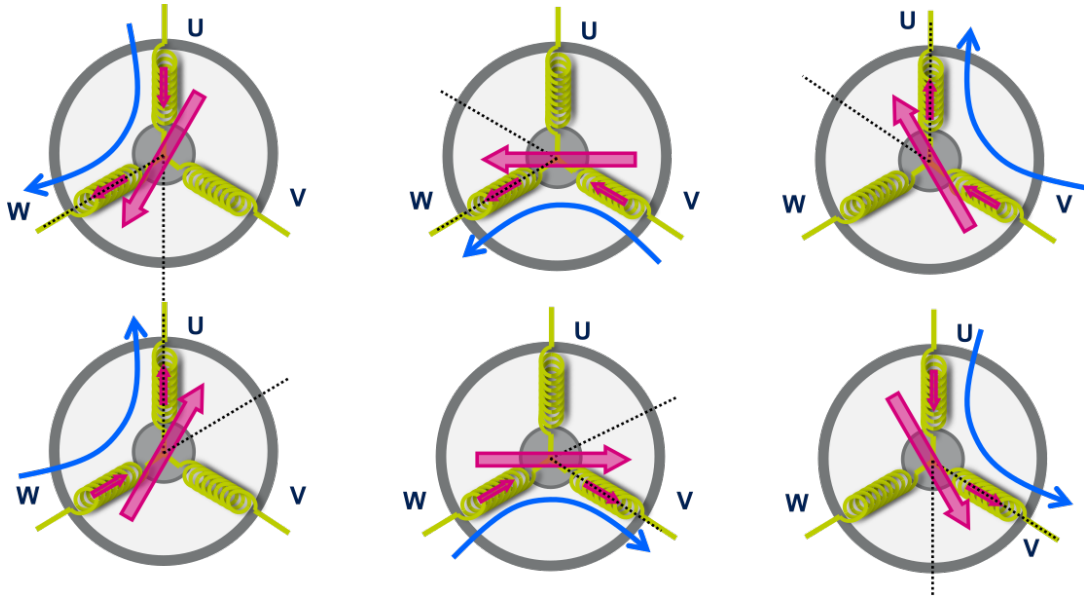
**Figure 5. Motor stator and rotor magnetic fields**



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.

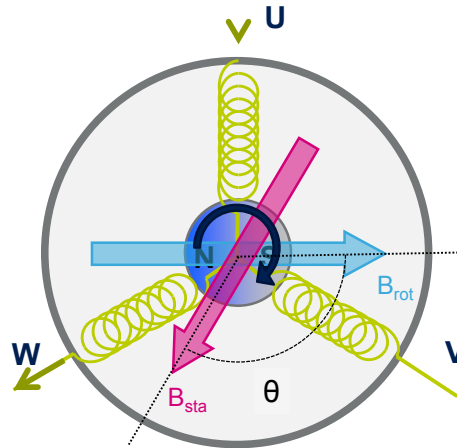
The subsequent steps are executed in the same way but the bus voltage and the ground are applied to different motor phase windings to generate a rotating stator magnetic field with six different discrete positions as shown below.

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  $90^\circ$ , as shown in the figure below. The step commutation ensures the torque is always close to  $90^\circ$ .

Figure 7. Motor stator magnetic field discrete positions



The torque applied to the motor is proportional to the sine of the load angle ( $\theta$ ). 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.

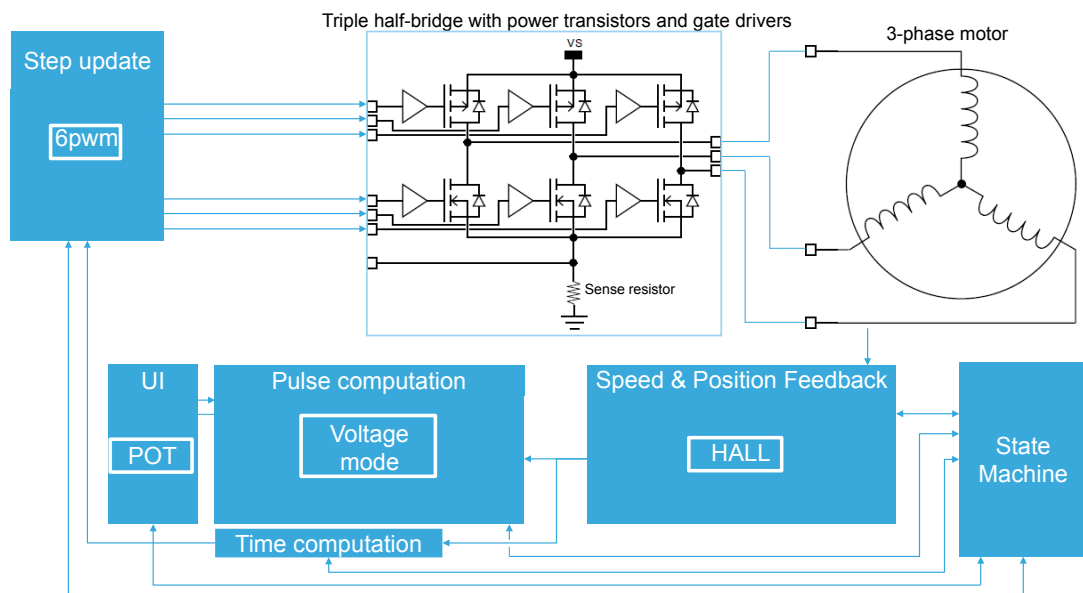
#### 3.1.1 Components

The six-step firmware can be split into:

- power on/off management – keep-alive circuit
- speed and feedback component
- pulse computation component (voltage mode - VM)
- proportional-integral-derivative (PID) controller algorithm for speed loop control feature
- time computation component
- user interface component - potentiometer voltage (POT)
- step update component - Six-PWM (6pwm)
- state machine component

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 Power on/off management - keep-alive circuit

As soon as the power is supplied and the firmware starts running, the PF0 GPIO is set to high in order to enable the STEVAL-PTOOL1V1 keep-alive circuit. The circuit maintains the power supply connected after switch-off signal reception to properly brake and stop the motor.

An external interrupt on the rising edge is configured on PF1 GPIO. When the logic status change (from low to high) is detected on PF1, the braking sequence starts. All the PWM on N-channels are enabled with duty cycle 0% activating all the power bridge low side MOSFETs.

PF0 GPIO is then set to low, disabling the keep-alive circuit and disconnecting the power supply.

### 3.1.3 Tasks

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

The supported tasks are:

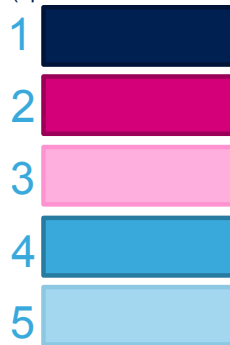
- High frequency (HF) task
- Medium frequency (MF) task
- Low frequency (LF) task

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)



#### 3.1.3.1 Medium frequency task

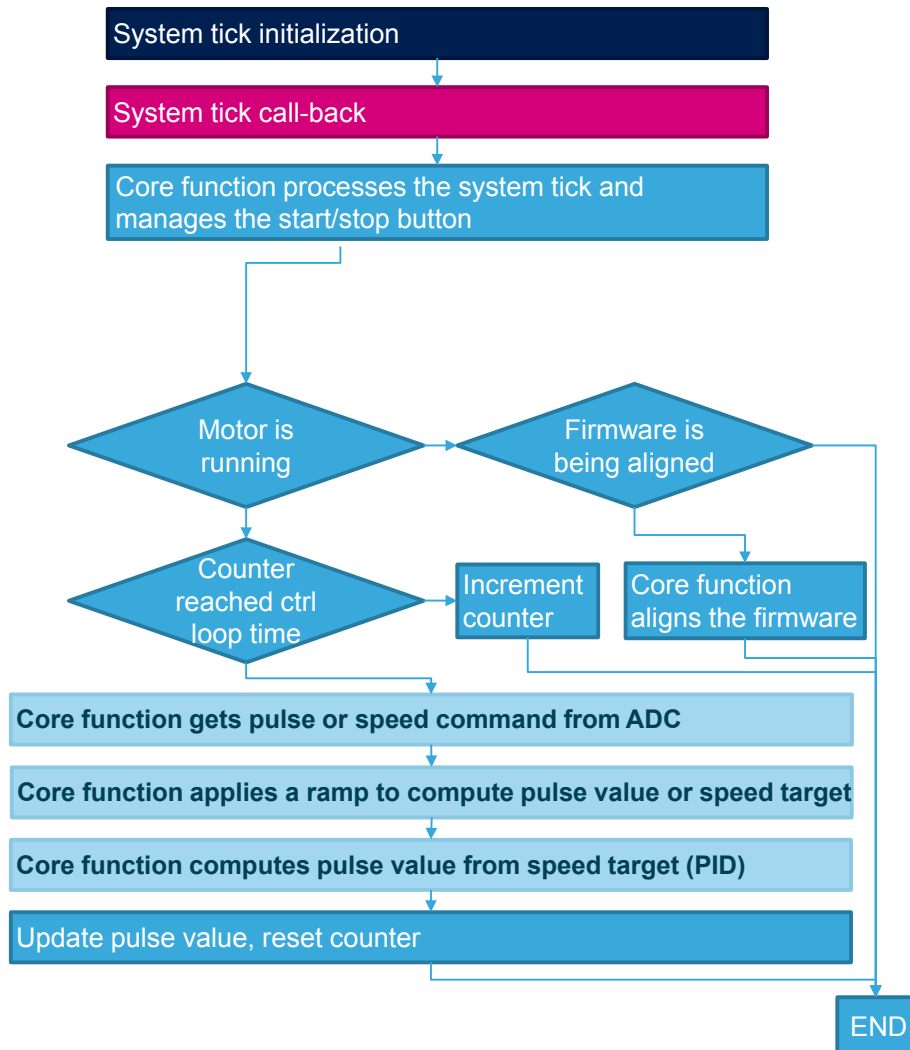
This is a low priority task. It occurs every 1 ms and manages the duty cycle (the pulse) of the HF PWMs.

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.



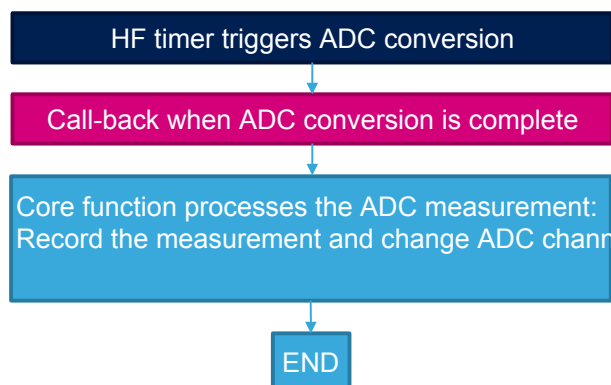
Figure 10. Medium frequency task diagram



### 3.1.3.2 High frequency task

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

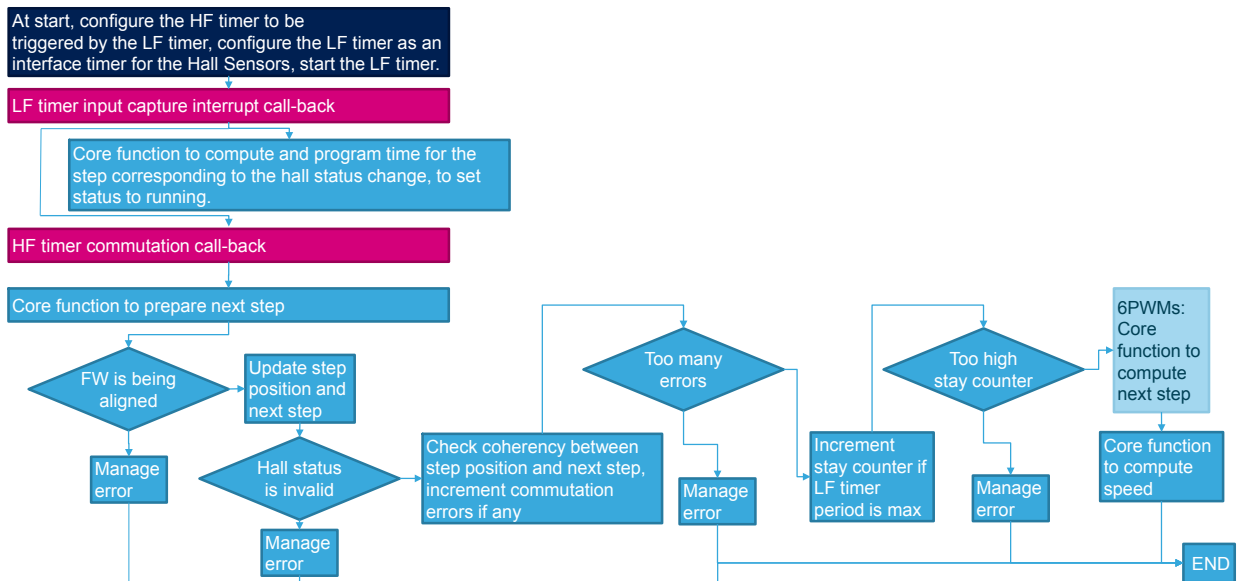
Figure 11. High frequency task diagram



### 3.1.3.3 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 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 are not used.

### 3.3 Hall sensor algorithm

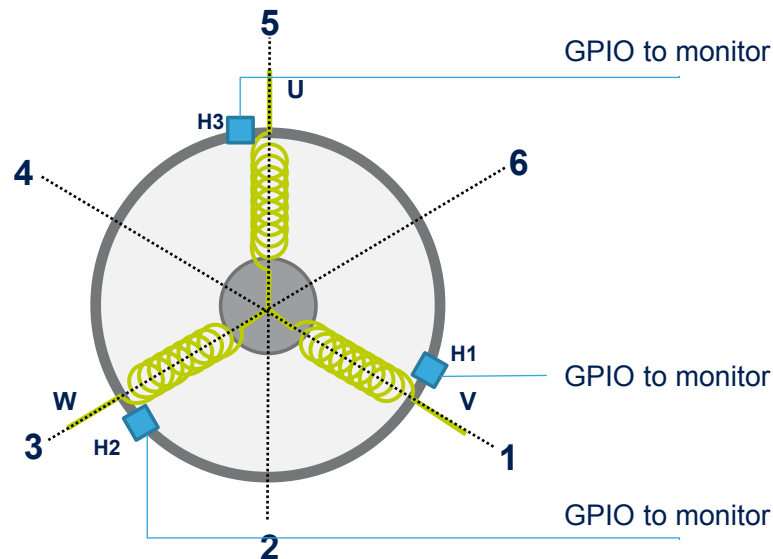
The three Hall effect sensors detect the rotor position returning digital signal values.

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

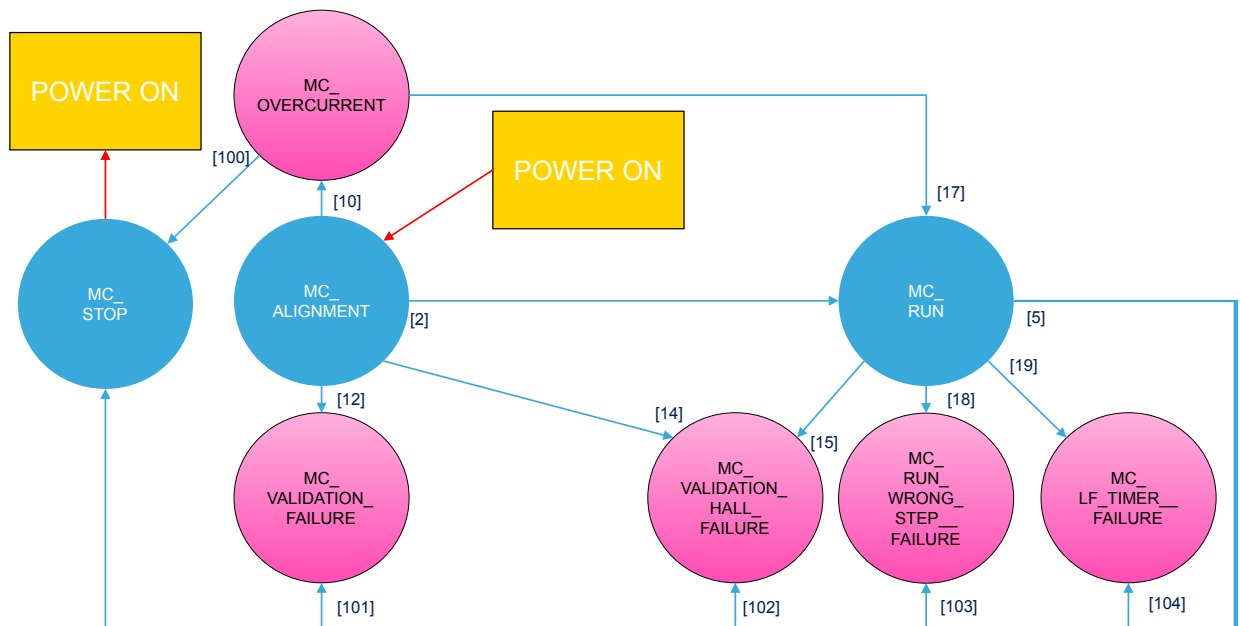
To use the Hall sensor algorithm, you need the following resources:

- 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
- 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

After power on and resource initialization, the firmware goes into alignment state:

- **MC\_ALIGNMENT**: 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
- **MC\_STOP**: the motor is stopped and the device is powered off

### 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 its application, you can select:

- Driving mode: voltage
- Sensing method: Hall sensors
- Speed loop: enabled
- Set point ramping: disabled
- Potentiometer interface: enabled or disabled

## Revision history

**Table 2. Document revision history**

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

## Contents

<b>1</b>	<b>STSW-PTOOL1V1 firmware package overview</b>	<b>2</b>
1.1	Package folders	2
1.1.1	Drivers	2
1.1.2	Middlewares	2
1.1.3	Projects	3
<b>2</b>	<b>Brushless DC motor basics</b>	<b>5</b>
<b>3</b>	<b>Six-step firmware algorithms</b>	<b>7</b>
3.1	Overview	7
3.1.1	Components	7
3.1.2	Power on/off management - keep-alive circuit	7
3.1.3	Tasks	8
3.2	Voltage driving mode	10
3.3	Hall sensor algorithm	10
3.3.1	MCU hardware requirements and use	11
3.3.2	State machine	11
<b>4</b>	<b>Six-step firmware setup</b>	<b>14</b>
	<b>Revision history</b>	<b>15</b>
	<b>Contents</b>	<b>16</b>
	<b>List of figures</b>	<b>17</b>
	<b>List of tables</b>	<b>18</b>



## List of figures

Figure 1.	Drivers folder content . . . . .	2
Figure 2.	Drivers folder content . . . . .	3
Figure 3.	Projects folder content . . . . .	3
Figure 4.	Motor stator and rotor arrangement . . . . .	5
Figure 5.	Motor stator and rotor magnetic fields . . . . .	5
Figure 6.	Motor stator magnetic field discrete positions . . . . .	6
Figure 7.	Motor stator magnetic field discrete positions . . . . .	6
Figure 8.	Six-step firmware high level architecture block diagram . . . . .	7
Figure 9.	Color code for tasks . . . . .	8
Figure 10.	Medium frequency task diagram . . . . .	9
Figure 11.	High frequency task diagram . . . . .	9
Figure 12.	Low frequency task diagram . . . . .	10
Figure 13.	Motor with Hall effect sensors . . . . .	11
Figure 14.	Six-step Hall sensor state machine . . . . .	12

## List of tables

<b>Table 1.</b>	Six-step firmware truth table .....	10
<b>Table 2.</b>	Document revision history .....	15

**IMPORTANT NOTICE – PLEASE READ CAREFULLY**

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, please refer to [www.st.com/trademarks](http://www.st.com/trademarks). All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2020 STMicroelectronics – All rights reserved