Getting started with the STSW-SPIN3202 and STSW-SPIN3204 firmware packages

Introduction

The STSW-SPIN3202 and STSW-SPIN3204 firmware packages implement examples for driving 3-phase permanent magnet or brushless DC motors using a six-step (trapezoidal) control algorithm.

The STSW-SPIN3202 is the example for the STEVAL-SPIN3202 board, based on the STSPIN32F0A system-in-package, while the STSW-SPIN3204 is for the STEVAL-SPIN3204 board, based on the STSPIN32F0B system-in-package.

For the differences between the two hardware solutions, please refer to the respective user manuals UM2278 and UM2572.

1 Acronyms and abbreviations

Table 1. Acronyms and abbreviations

Acronym	Description
6Step Lib	The six-step driving library
API	Application programming interface
BEMF	Back electromagnetic force
BLDC	Brushless direct current
CMSIS	Cortex® microcontroller software interface standard
HF_TIMx	High frequency timer used to generate the PWM input of the gate drivers
IDE	Integrated development environment
LF_TIMx	Low frequency timer used for the six-step commutation
PMSM	Permanent magnet synchronous motor
SIP	System-in-package

2 Firmware package overview

The firmware package includes:

- drivers for the STEVAL-SPIN3202 or STEVAL-SPIN3204 evaluation board, STSPIN32F0A or STSPIN32F0B SIP and embedded STM32
- middleware for the six-step driving library and the serial communication user interface
- sample motor control application project files

Note: The firmware packages can be adapted to suit other boards mounting the SIP of the SPIN32F0x family.

The 6Step Lib supports four different control modes:

1. sensorless voltage mode
2. sensorless current mode
3. voltage mode with Hall effect sensor feedback
4. current mode with Hall effect sensor feedback

In the sensorless modes, the rotor position is determined through detection of the BEMF zero-crossing.

The firmware library and the example are written in the C programming language and uses the STM32Cube HAL embedded abstraction-layer software or optimized access to the STM32F031 resources.

Note: To use this library, you should have a basic knowledge of C, 3-phase motor drives and power inverter hardware(In-depth knowledge of STM32 functions is only required to customize existing modules or to add new ones in a wider project development.).

The Drivers abstract low-level hardware information, so the middleware components and applications can fully manage the STSPIN32F0A/B through a complete set of APIs which send commands to the motor driver in a hardware-independent manner. The package includes an application example to drive a low voltage three-phase BLDC/PMSM motor and several motor control parameters files to be used directly with your corresponding motor or as template for similar ones.

2.1 Package content

Once package is unzipped, its contents are arranged into different folders under the main stm32_cube folder.

2.1.1 Drivers

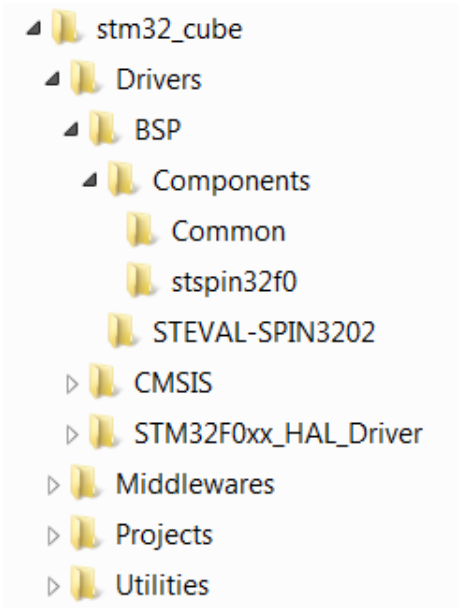
The Drivers folder contains the source code for:

- STSPIN32F0: the STSPIN32F0 code is located in the Components folder; the STSPIN32F0 code uses the HAL API.
- STEVAL-SPIN3202 / STEVAL-SPIN3204: board specific drivers
- STM32Cube HAL: STM32Cube HAL embedded abstraction-layer for the peripheral drivers.
- CMSIS drivers: vendor-independent hardware abstraction layer for the ARM® Cortex™-M processor series.

stspin32f0.h is the interface file for the HW resource mapping regarding the 3-phase controller circuit with embedded MCU (STSPIN32F0A/B). It includes the high frequency timer mapping between the embedded MCU and the embedded gate drivers. This file should not normally be changed.

STEVAL-SPIN3202.h / STEVAL-SPIN3204.h is the interface file for the HW resource mapping regarding the board. It includes the low frequency timer mapping associated with the driving methods chosen for the board (sensorless or Hall effect sensors, voltage or current mode). This file is updated according to any modifications performed on the evaluation board or to use a different board with the one of the STSPIN32F0x system-in-packages.

Figure 1. **Driver folder content (STSW-SPIN3202)**

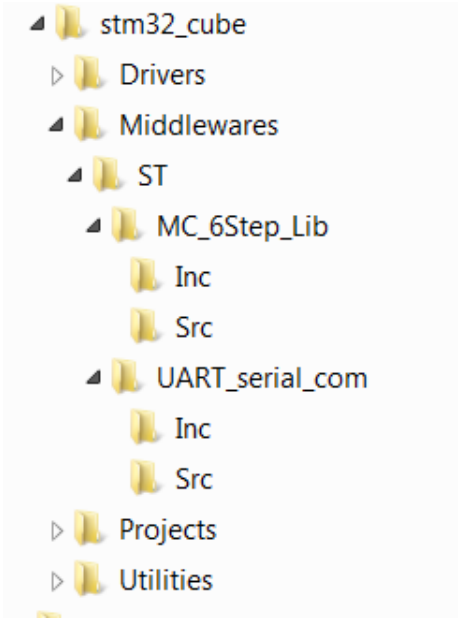


2.1.2 Middlewares

The Middlewares folder contains:

- the source code of the 6Step Lib (the core of the motor control algorithm): 6Step_Lib.c and 6Step_Lib.h
- the serial communication user interface (based on UART): UART_UI.c and UART_UI.h

Figure 2. **Middlewares folder content**



2.1.3 Projects

The Projects folder contains the project-specific source code demonstrating board functionality and IDE project files.

The Src sub-folder contains:

- **main_32F0.c**: the main file for FW initialization (peripherals, MC_6Step and UART communication). It contains all the MCU initialization for the timers, ADC, GPIOs, UART etc. and the entry point for the 6Step

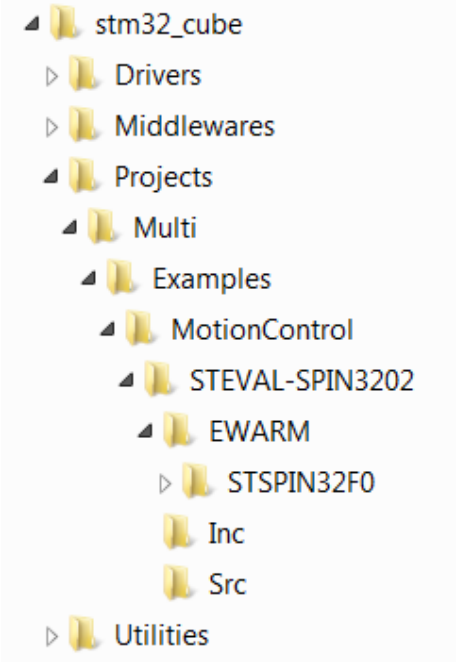
Library. With the include of the header file 6Step_Lib.h and the 6Step init call `MC_SixStep_INIT()`, the user level is linked with the motor library and all API functions are available.

- **stspin32f0_hal_msp.c**: the standard ST Cube HAL file for MCU configuration which also contains the HAL callbacks (i.e., the ADC callback).
- **stspin32f0_it.c**: the ST Cube HAL file for MCU interrupt request and handling functions. It defines all the interrupt handlers and contains the starting point for UART communication.

The Inc sub-folder contains:

- **main_32F0.h**: includes the header file 6Step_Lib.h.
- **stm32f0xx_hal_conf.h**: ST Cube HAL configuration, including the list of modules to be used in the HAL driver.
- **stspin32f0_it.h**: contains the headers of the interrupt handlers.
- **MC_SixStep_param_32F0.h**: includes the motor control parameters to drive a motor with the 6Step Lib.
- Several **MC_SixStep_param_<MotorIdentifier>.h**: motor control parameter files for one motor model example.

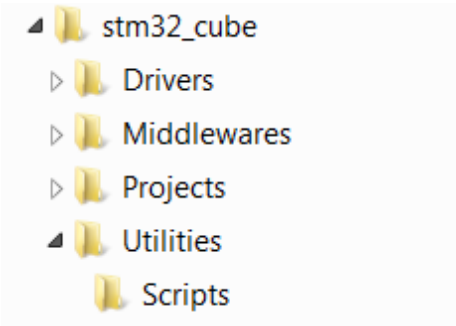
Figure 3. Projects folder content (STSW-SPIN3202)



2.1.4 Utilities

This folder contains python 2.7.13 scripts to test and tune the motor control firmware.

Figure 4. Utilities folder content



The **SpeedRecording.py** script generates speed versus time data with the granularity of the firmware speed loop, and interact with the firmware in real time as if it were through a serial communication terminal. When the motor is stopped, the test ends and the corresponding graph is created.

The **BemfRecording.py** script generates BEMF and step position versus time data. It is also possible to interact in real time with the firmware in real time as if it were through a serial communication terminal. When the motor is stopped, the test ends and the corresponding graph is created.

The **StartStopTest.py** reliability test to check whether the motor can be started and stopped when commanded.

2.2 Architecture

The example uses the following software layers:

- **Demonstration layer**: includes the software demonstrating the evaluation board features based on the middleware service layer, the low level abstraction layer (drivers) and the basic peripheral usage applications for board-based functions.
- **Middleware layer**: includes the user interface and the 6Step Lib. The 6Step Lib provides an API to control a 3-phases BLDC/PMSM motor. The user interface uses serial communication to send commands through a terminal to control a 3-phase BLDC/PMSM motor. The 6Step Lib and user interface interact by calling their respective APIs.
- **Driver layer**: includes the STM32Cube HAL sub-layer and the board support package (BSP) sub-layer.

The STM32Cube HAL sub-layer provides the low level drivers and the hardware interfacing methods to interact with the upper layers (application, libraries and stacks). It provides generic, multi-instance and function-oriented APIs to help reduce user application development time by providing ready-to-use processes.

For example, it provides APIs for communication peripheral (I²C, UART, etc.) initialization and configuration, data transfer management based on polling, interrupt or DMA processes, and communication error management.

The HAL Drivers APIs are split in two categories:

1. generic APIs providing common, generic functions to all the STM32 series
2. extension APIs with special functions for a specific family or part number

The BSP sub-layer offers a set of APIs relative to the hardware components in the hardware board. It is based on modular architecture so it can be ported to any hardware by simply implementing the low level routines.

The BSP sub-layer is composed of:

1. Component driver: for the external devices on the board (excluding the STM32), the component driver provides specific APIs to the external components of the BSP driver, and can be ported to any other board.
2. BSP driver: links the component driver to a specific board and provides a set of easy-to-use APIs.

Figure 5. **STSW-SPIN3202 architecture**

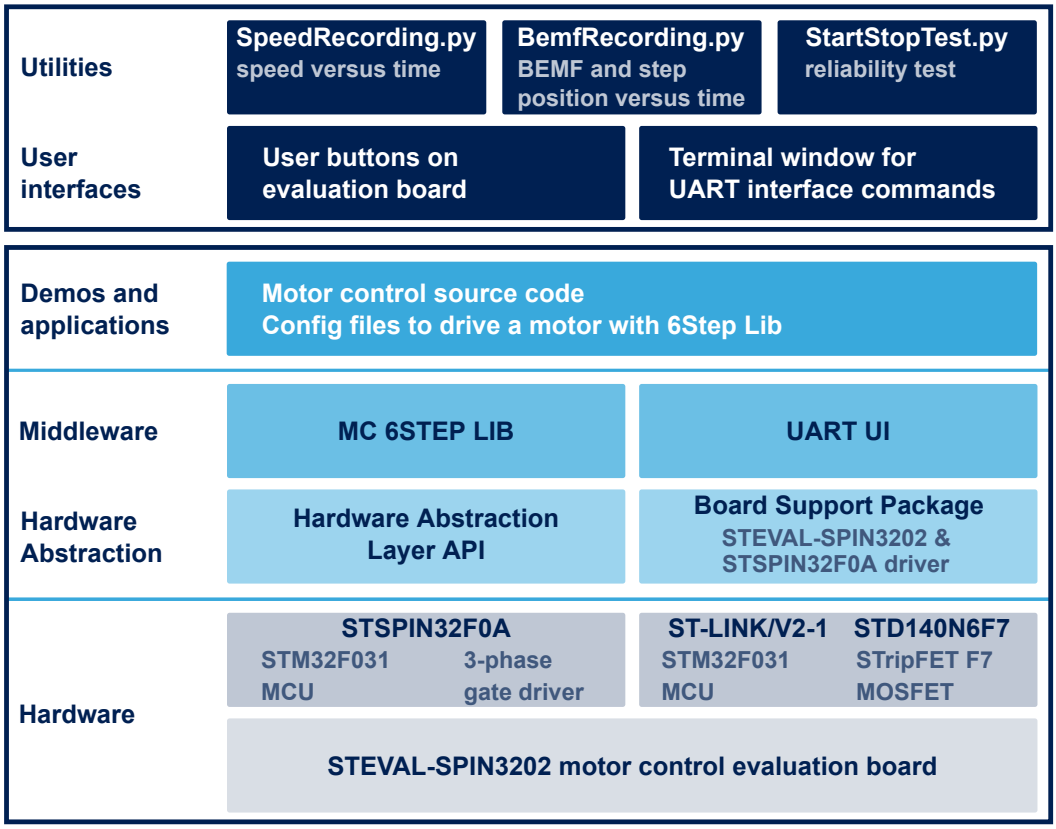
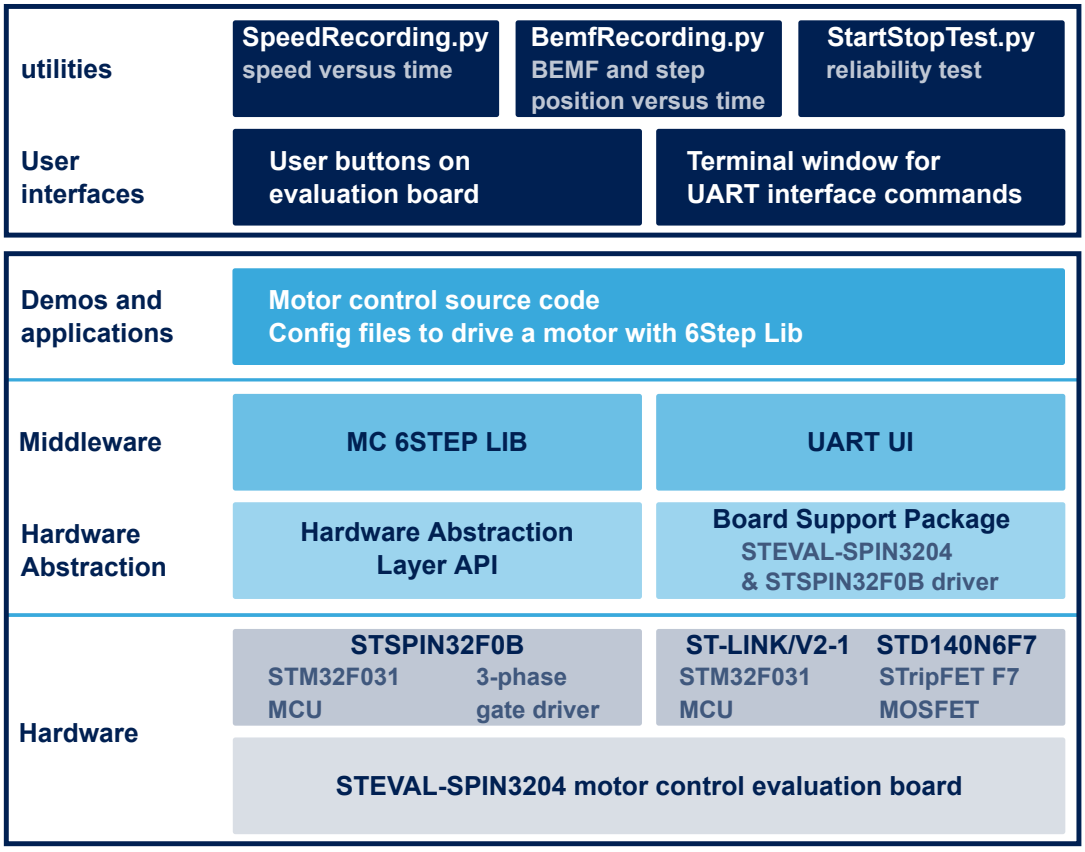


Figure 6. **STSW-SPIN3204 architecture**



2.3 6Step Lib features

- Six-step algorithm with:
  - sensorless mode: the back EMF (BEMF) voltage on the non-energized phase is monitored and used to trigger the commutation events
  - Hall-based sensor feedback: digital Hall effect sensors are used to derive the position and speed of the rotor
  - voltage mode: driving voltage is set directly through the PWM duty cycle applied by the power stage
  - current mode: this current control loop limits the current in the motor windings
- Speed loop control based on PI controller or optional PID controller.
- Speed ramping, sometimes referred as reference Setpoint ramping.
- User interface through serial communication.
- Start/Stop button and Fault LED.
- Optional speed reference control using potentiometer.
- Optional ADC measurement (phase current, DC bus voltage, chipset internal temperature).
- Fault handling and protections (overcurrent, speed feedback error etc.)

2.4 Flash and RAM requirements:

The Flash and RAM requirements are depending on the features and options selected for the compilation. The figures presented in the table below have been obtained with an optimization level high privileging speed over code size (using IAR IDE).

Table 2. **Flash and RAM requirements**

Speed reference	Sensing Mode	Control Mode	Flash (Kbytes)	RAM (Kbytes)
Potentiometer	Sensorless	Voltage	13.2	1.8
		Current	13.5	1.9
	Hall effect sensors	Voltage	12.1	1.7
		Current	12.4	1.8
Uart	Sensorless	Voltage	19	2.9
		Current	19.3	3
	Hall effect sensors	Voltage	16.4	2.9
		Current	16.7	2.9

2.5 System setup guide

2.5.1 Hardware setup

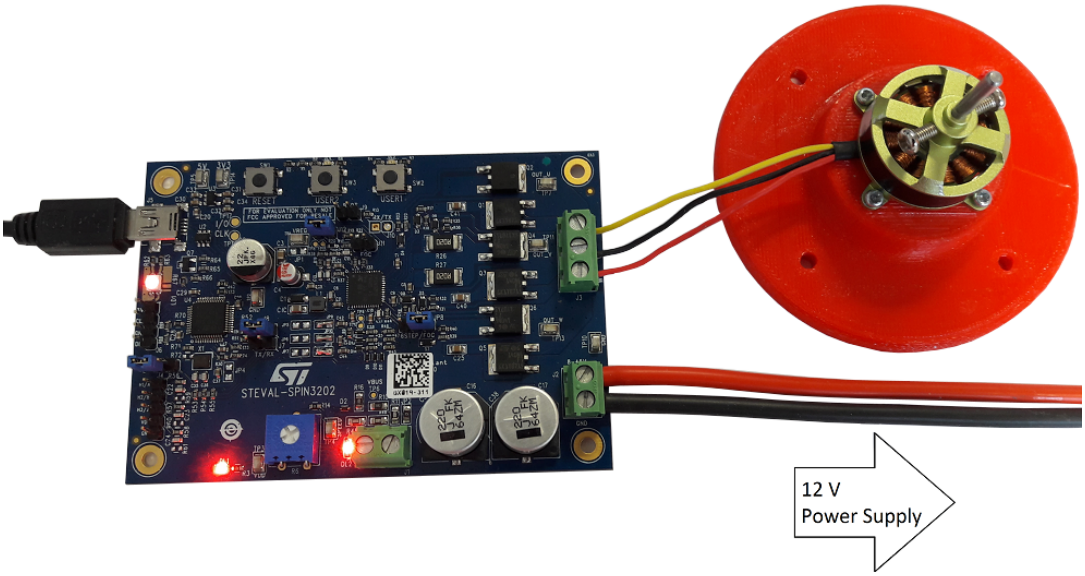
To use the firmware example with one of the predefined motor configuration, it is required to have:

- One STEVAL-SPIN3202 or STEVAL-SPIN3204 board, according to the firmware package
- One of the following motors:
  - Bull-Running BR2804-1700 kV
  - RS Pro 57BL54
  - Faulhaber Minimotor 2036U024B

  (1)
- A DC power supply between 8 V and 45 V (e.g., 12 V for Bull-Running BR2804).
- A USB cable with a mini-B connector.

1. *A different motor can be driven, using one of the existing MC_SixStep_param_*.h file as a template to enter the new configuration parameters.*

Figure 7. **STEVAL-SPIN3202 board with a Bull-Running motor**



2.5.2 Build and load a customized firmware

You can customize the firmware with one of the following IDEs:

- µVision of the ARM Keil Microcontroller Development Kit (MDK-ARM) toolchain (V5.17 or above)
- IAR embedded workbench for the ARM (EWARM) toolchain (V7.50 or above), provided by IAR Systems®
- System workbench for the STM32, a GCC toolchain based on Eclipse and provided by AC6.

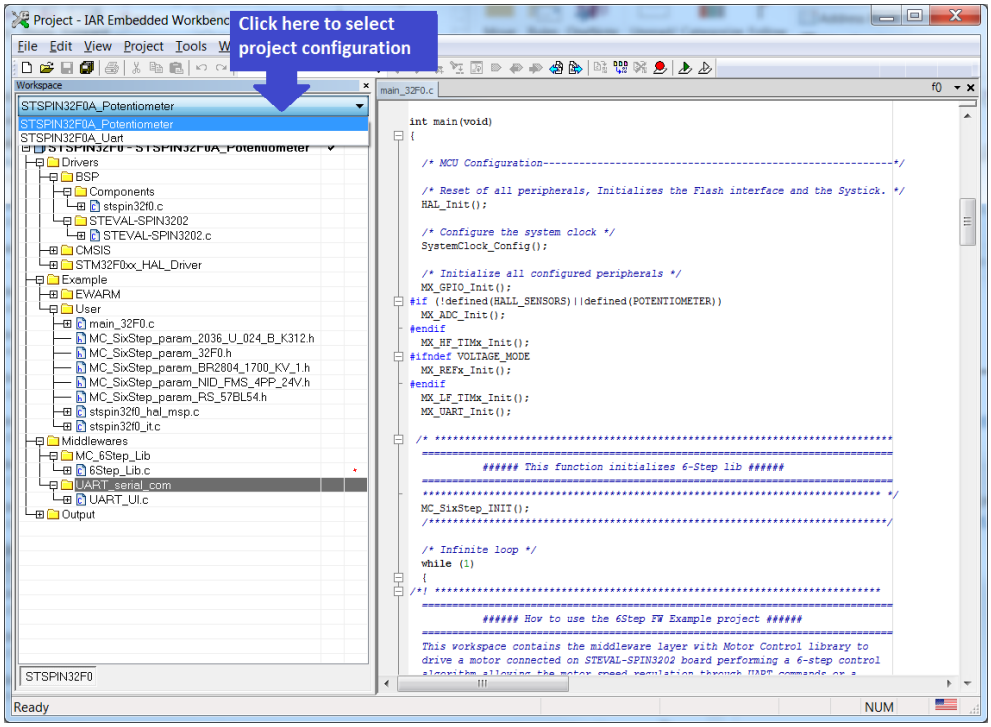
2.5.2.1 *Create a build with IAR IDE*

Step 1. Select one of the following configurations by clicking on the scrolling menu just below the workspace panel title:

a. Potentiometer (for speed setting)

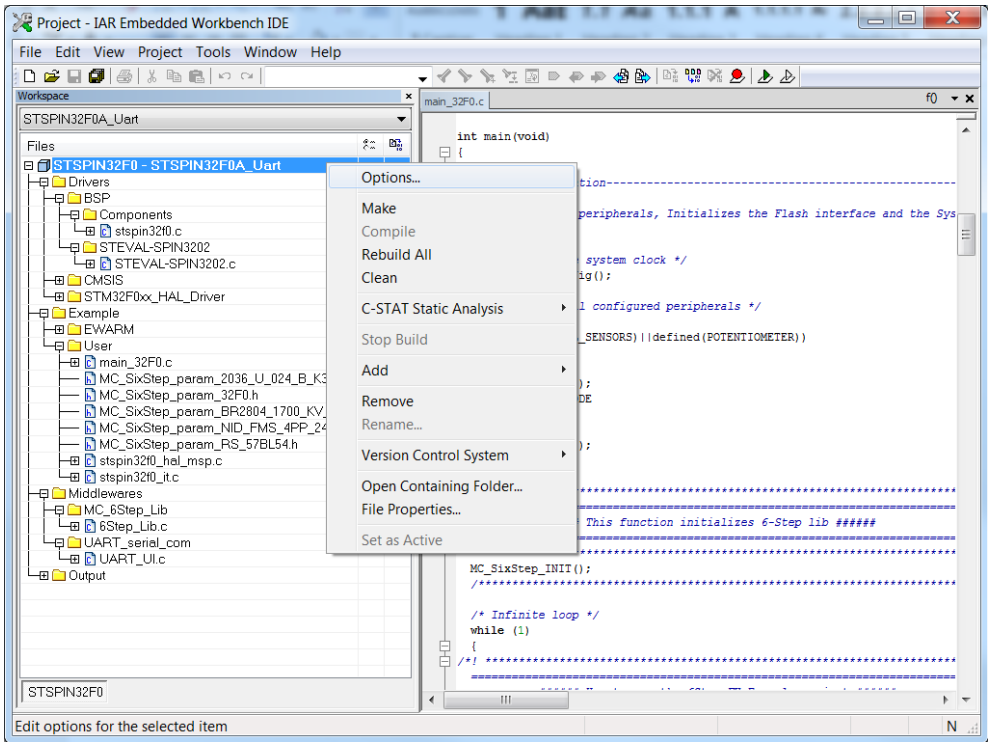
b. Uart (for UART communication)

Figure 8. IAR workspace with Potentiometer configuration



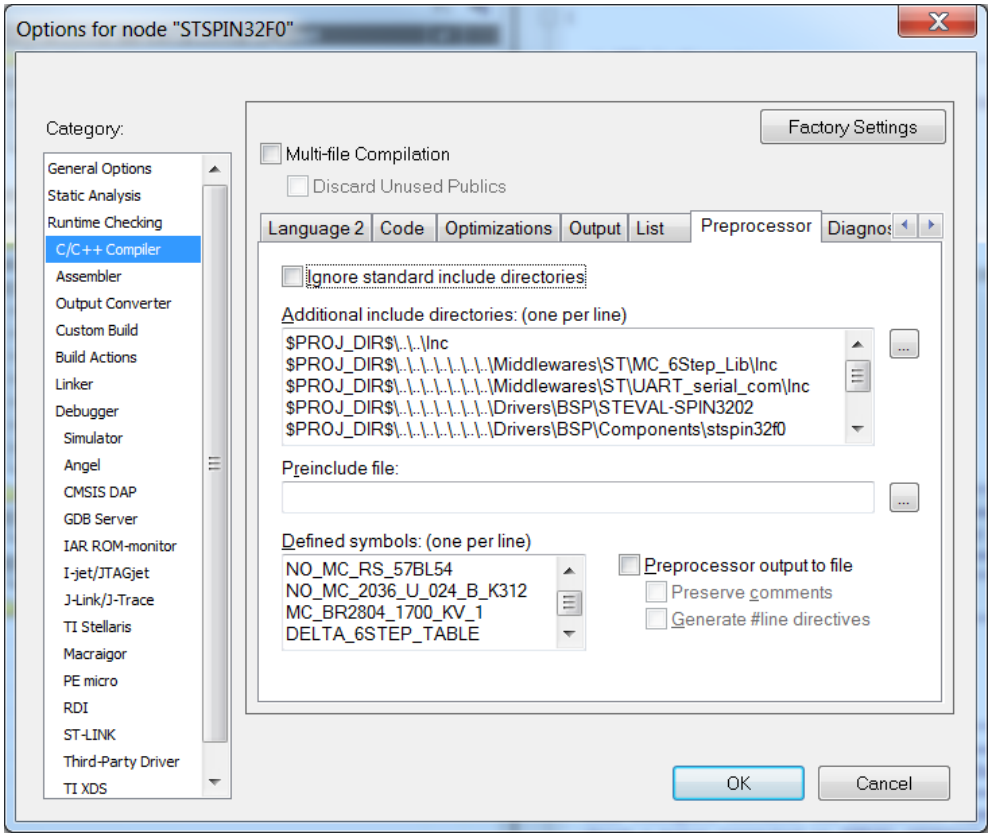
Step 2. Edit the project configuration: right-click on the project name and left-click “Options” in the context menu

Figure 9. **IAR workspace with Options menu to edit project configuration**



Step 3. In the “Options” window, enable or disable features of the firmware by going into the “C/C++ Compiler” Category “Preprocessor” panel and listing the relevant symbols in the “Defined symbols” list.

Figure 10. **IAR project options window – C/C++ Preprocessor defined symbols**



Step 4. Build your project: go to the menu “Project” and left-click “Rebuild All”.

Step 5. Ensure the board is powered and connected to the PC via USB cable.

Step 6. From the menu, select Project→Download and Debug to download the binary and start debugging.

2.5.2.2 *Create a build with Keil µVision IDE*

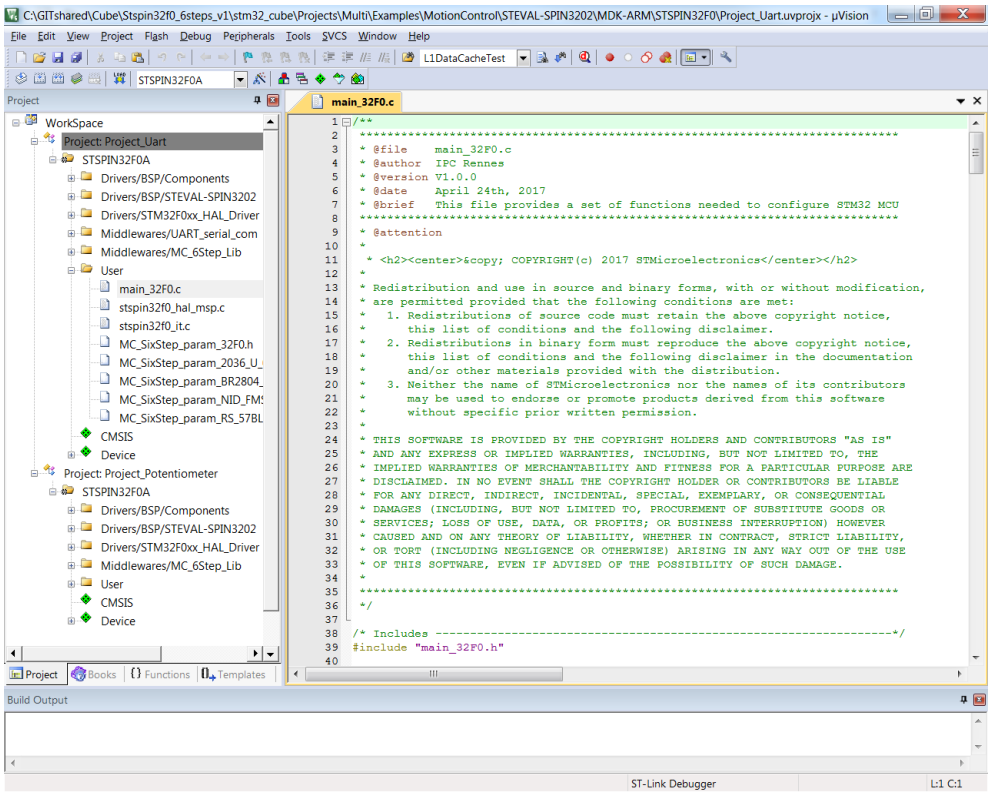
Step 1. Open the multi-projects workspace

The workspace includes two projects:

a. one for UART communication

b. one for speed setting via a potentiometer.

Figure 11. KEIL projects workspace



Step 2. From the menu, select Project→Batch build, then select the two project targets and click Rebuild.

Step 3. To edit the project options:

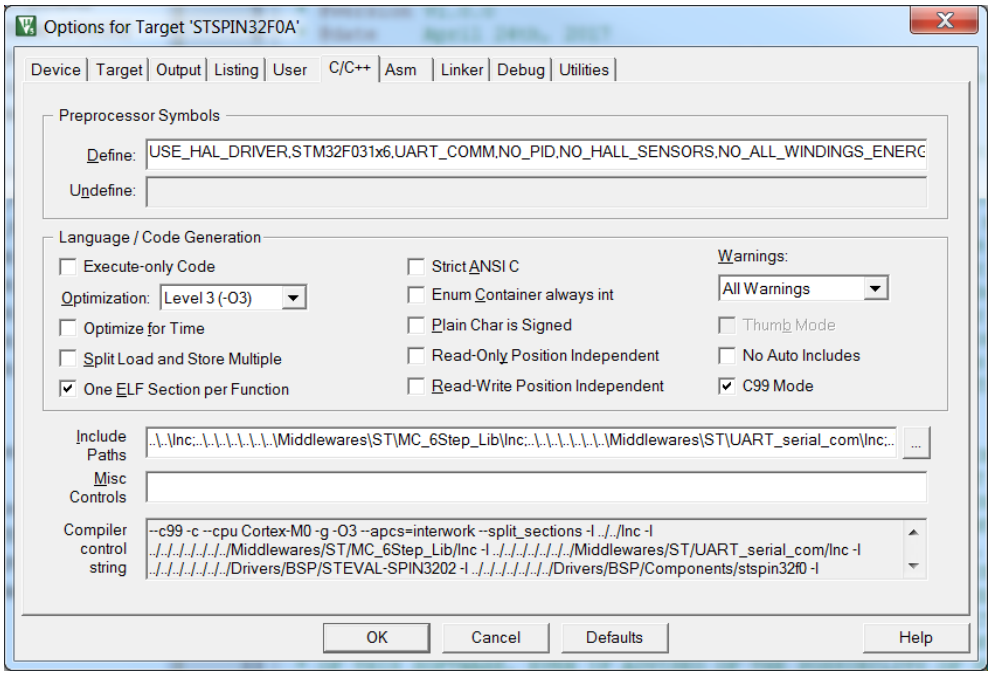
– right-click on the Project in workspace explorer to set it as Active

– right-click on target name just below the project name and select “Options for Project …”

Step 4. Use the “Options for Target” window to enable or disable features of the firmware:

– Open the “C/C++” tab and list the relevant symbols in the “Define” list of the “Preprocessor Symbols” group.

Figure 12. **KEIL project options for target STSPIN32F0**



Step 5. Ensure the board is powered and connected to the PC via USB cable.

Step 6. From the menu, select Flash→Download to load the binary.

Step 7. From the menu, select Debug→Start/Stop debug Session to debug your project.

2.5.2.3 *Create a build with system workbench for STM32*

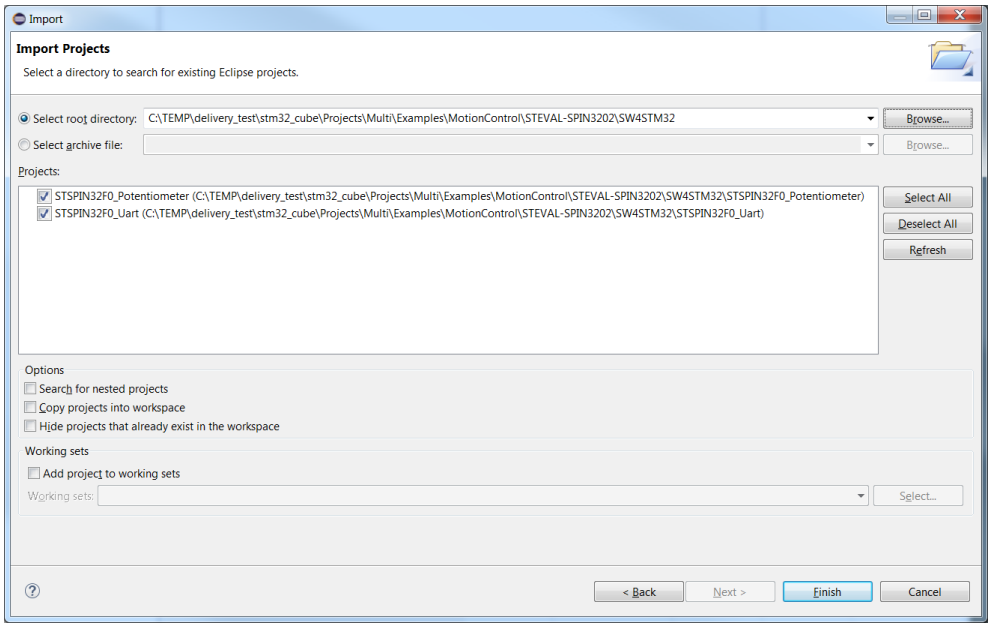
Step 1. Open the system workbench for the STM32.

Step 2. From the menu, select File→Import→General→Existing Projects into Workspace.

Step 3. Set {yourPath}\Projects\Multi\Examples\MotionControl/target evaluation board\SW4STM32 as the root directory

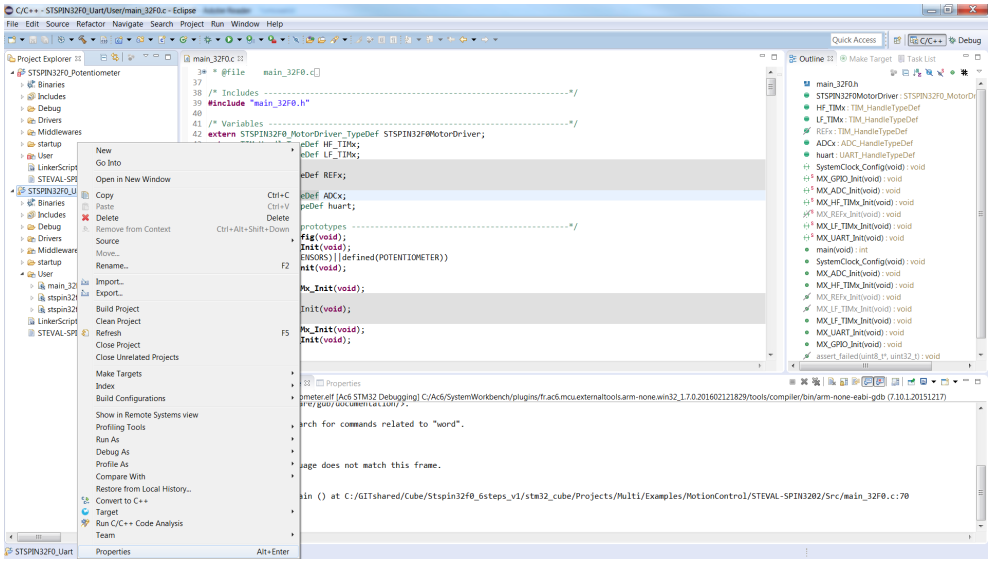
Step 4. Select the two projects STSPIN32F0_Potentiometer and STSPIN32F0_Uart and click the Finish button to import them.

Figure 13. **SW4STM32 projects import**



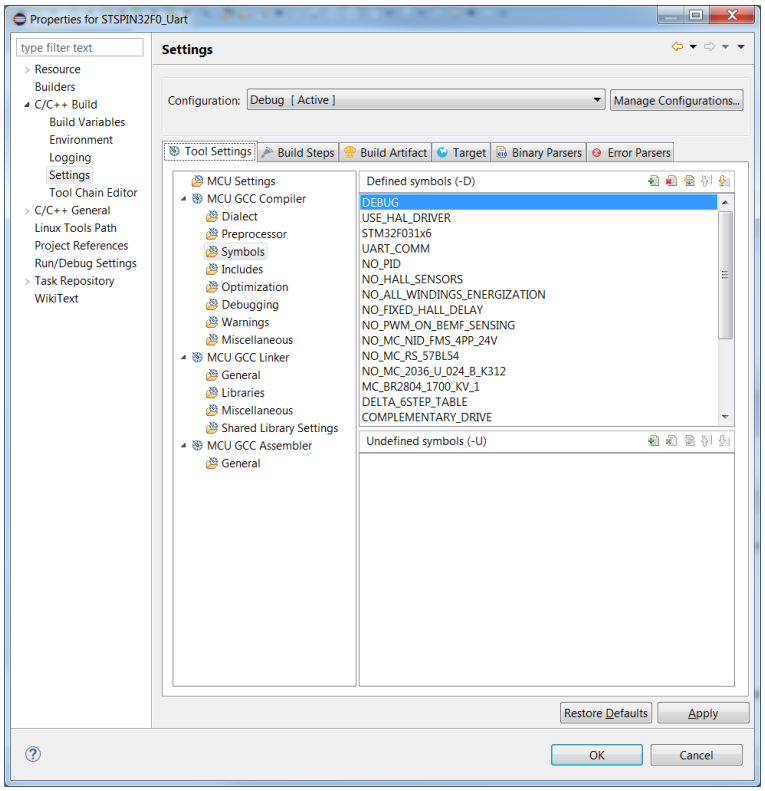
Step 5. Right-click on the project name in the Project Explorer and select Properties

Figure 14. **SW4STM32 project properties selection**



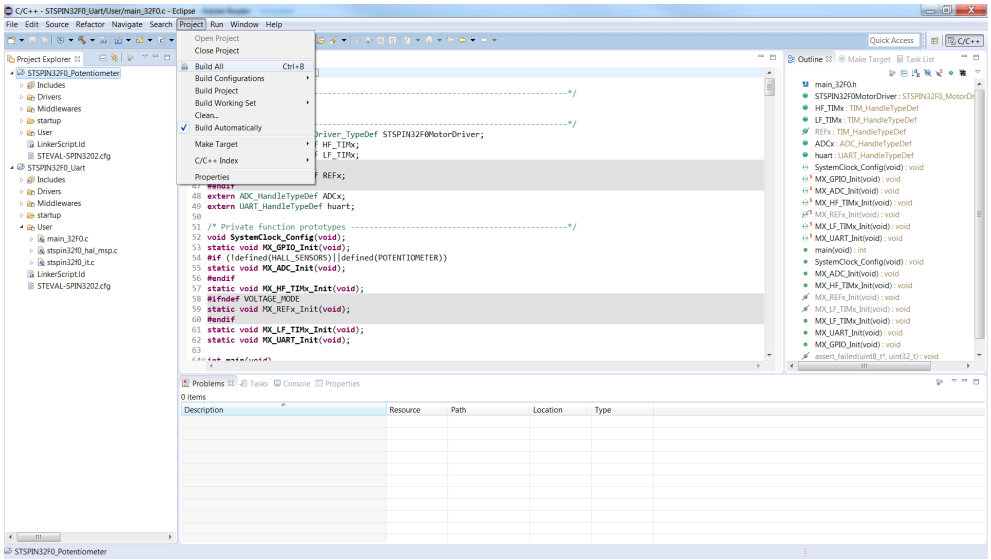
Step 6. In the Properties window, select the C/C++ Build→Symbols menu in the ‘Tool Settings” tab
You can now change the list of “Defined symbols”.

Figure 15. **SW4STM32 project defined symbols**



Step 7. From the menu, select Project→Build All to build your project.

Figure 16. **SW4STM32 workspace and projects compilation**



Step 8. Select one of the projects in the “Project Explorer” window.

A Binaries folder should have appeared with an “.elf” file inside. Otherwise, press F5 to refresh the window.

Step 9. Right-click on the .elf file and select “debug as AC6 STM32 C/C++ Application” tol flash the binary and start the debug window.

2.5.3 Defined symbols for firmware customization

Table 3. Symbols list

Symbol	Description	Default UART [1]	Default Potentiometer[1]
USE_HAL_DRIVER	Enable the HAL drivers (mandatory)	Enabled	Enabled
STM32F031x6	Enable the specific configuration for the STM32F031x6 microcontroller (mandatory)	Enabled	Enabled
UART_COMM	Enable the UART communication	Enabled	Disabled
PID	Enable the PID regulator instead of the standard PI regulator	Disabled	Disabled
HALL_SENSORS	Enable the Hall effect sensor feedback disabling the sensorless driving	Disabled	Disabled
ALL_WINDINGS_ENERGIZATION	Enable current to circulate in all motor phases during the alignment procedure.	Disabled	Disabled
FIXED_HALL_DELAY	Set the delay method when the Hall effect sensor feedback is enabled	Disabled	Disabled
PWM_ON_BEMF_SENSING	Enabled the BEMF sensing during the ON time of the PWM when sensorless driving is used	Disabled	Disabled
MC_FAN_4PP_24V	Select a generic fan motor with hall sensors configuration file.	Disabled	Disabled
MC_RS_57BL54	Select the RS Pro 57BL54 configuration file	Disabled	Disabled
MC_2036_U_024_B_K312	Select the Faulhaber Minimotor 2036U024B configuration file	Disabled	Disabled
MC_BR2804_1700_KV_1	Select the Bull-Running BR2804-1700 kV configuration file	Enabled	Enabled
DELTA_6STEP_TABLE	Enabled differential strategy in six-step sequence generation	Enabled	Enabled
COMPLEMENTARY_DRIVE	Complementary driving strategy	Enabled	Enabled
VOLTAGE_MODE	Enable the voltage mode driving instead of the current mode	Enabled	Enabled
POTENTIOMETER	Enable the control of the target speed through the potentiometer	Disabled	Enabled
CURRENT_SENSE_ADC	Enable the current monitoring through the ADC	Disabled	Disabled
VBUS_SENSE_ADC	Enable the supply voltage monitoring through the ADC	Disabled	Disabled
TEMP_SENSE_ADC	Enable the MCU temperature monitoring through the ADC	Disabled	Disabled
BEMF_RECORDING	Enable the recording of the BEMF	Disabled	Disabled
SPEED_SENDING	Enable the monitoring of the speed through UART	Disabled	Disabled
SPEED_RAMP	Set the target speed of the motor limiting the acceleration	Enabled	Enabled

1. *Symbols are disabled adding the “NO_” prefix.*

2.5.3.1 *USE_HAL_DRIVER (mandatory)*

This symbol must be defined to base the application, middleware and BSP code on the HAL driver API. Otherwise, only direct access to the embedded STM32F031 peripheral registers is possible.

2.5.3.2 *STM32F031x6 (mandatory)*

This symbol must be defined to use the data structures, address mapping, bit definitions and the macros specific to the STM32F031 embedded in the STSPIN32F0A chip.

2.5.3.3 *UART_COMM or POTENTIOMETER (at least one is mandatory)*

The UART_COMM symbol enables the usage of the serial port as a user interface. The service is based on the UART_serial_com middleware.

If the UART_COMM symbol is not defined, the POTENTIOMETER has to be defined to allow speed control through the on-board potentiometer.

The POTENTIOMETER symbol can also be defined in addition to the UART_COMM symbol. In this configuration, the speed can be controller either by the on-board potentiometer or by the SETSPD command using a serial port terminal.

2.5.3.4 *PID (optional)*

This symbol enables a PID regulator for speed and torque control instead of a PI regulator.

2.5.3.5 *HALL_SENSORS (optional)*

This symbol enables the support of Hall effect sensors (Hall sensing drive) and disables the sensorless drive.

2.5.3.6 *FIXED_HALL_DELAY (optional, valid only in HALL SENSING drive)*

This symbol controls the way the six-step commutation delay is calculated after a hall status change.

When this symbol is defined and the motor is in the RUN state, the delay is a constant number of clock cycles of the low frequency timer (LF_TIMx); otherwise, it is half the step time.

2.5.3.7 *PWM_ON_BEMF_SENSING (optional, valid only in VOLTAGE SENSORLESS drive)*

This symbol enables BEMF voltage sensing during the high frequency timer (HF_TIMx) PWM ON time:

For low duty cycle, the BEMF voltage is sensed during the PWM OFF time

For high duty cycle, the BEMF voltage is sensed during the PWM ON time

This symbol is only valid in voltage mode sensorless drive.

2.5.3.8 *MC_FAN_4PP_24V, MC_RS_57BL54, MC_2036_U_024_B_K312, MC_BR2804_1700_KV_1 (mandatory and mutually exclusive)*

You must list one these symbols or define your own to indicate which motor control parameters the firmware can use.

Each of the four existing symbols has a motor control parameter file associated with it. The mapping of the symbol to the file is done in the MC_SixStep_param_32F0.h file.

2.5.3.9 *DELTA_6STEP_TABLE (optional, default)*

This symbol enables six-step high frequency timer PWM channel configuration differential change: only the differences between previous step and new step are programmed. This allows less microcontroller computation and consequent higher motor speed.

2.5.3.10 *COMPLEMENTARY_DRIVE (optional, default)*

This symbol enables the complementary output of HF_TIMx PWM. This is also called synchronous rectification. It reduces conduction losses by letting the free-wheeling current flow into the lower MOSFET transistor instead of its body diode.

2.5.3.11 *VOLTAGE_MODE (optional, default)*

This symbol mode enables the voltage mode control: the HF_TIMx PWM duty cycle is directly set by the PI regulator output.

When this symbol is not defined, current mode control is enabled: The HF_TIMx duty cycle is programmed at a constant value and the HF_TIMx PWM is gated in the SIP by the Gate Driver Control Logic when the OC_Comp input voltage is above the OC threshold value. The PI speed regulator output is used to set the OC_Comp input voltage. This means that the PI speed regulator is the input of the current loop that actually generates the PWM duty cycle on the power MOSFET inputs.

2.5.3.12 *CURRENT_SENSE_ADC, VBUS_SENSE_ADC, TEMP_SENSE_ADC (optional)*

These symbols enable the use of the ADC to sense the current in the power MOSFET, the VBUS voltage on the high side of the power MOSFET bridges or the circuit temperature.

2.5.3.13 *BEMF_RECORDING (optional)*

This symbol enables firmware BEMF recording and sending: up to BEMF_ARRAY_SIZE consecutive BEMF samples and corresponding step positions are stored and then sent to the serial port using a UART DMA.

The frame sent to the serial port is also “time” stamped in order to an external trace tool (BemfRecording.py python script) to construct a BEMF voltage versus time and step position versus time graph.

2.5.3.14 *SPEED_SENDING (optional)*

This symbol enables firmware speed sending: each time the speed loop is run, the filtered speed feedback is sent to the serial port. An external trace tool (SpeedRecording.py python script) can then construct a Speed versus time graph.

2.5.3.15 *SPEED_RAMP (optional)*

This symbol enables firmware control of motor acceleration and deceleration with a speed ramp. The speed target is ramped linearly at the mechanical rate ACC defined in the motor control parameters file. The speed target is the set point input of the PI regulator loop.

2.5.4 Loading the firmware

The easiest way to load new firmware onto the device is to copy the binary file into the mass storage interface provided by the ST-LINK; a simple drag and drop operation to the disk is sufficient. The procedure to do so is explained below:

Step 1. **First time only**: install the STLINK V2-1 drivers that can be downloaded from the ST website (STSW-LINK009).

Step 2. Connect the evaluation board to a PC via USB cable.

Step 3. Supply the evaluation board through the connector (J2) with a DC voltage in the8 V – 45 V operating range.

Step 4. A new drive named SPIN32F0 should appear in the list of removable storages.

Step 5. Delete all the files on this drive.

This step is recommended when the binary size is close to the 32 Kbyte limit of the embedded MCU.

Step 6. Copy the binary file to the drive root.

The red/green LED on the ST-LINK should start blinking.

Step 7. Refresh file explorer.

If the binary file has disappeared and no error log file has been generated, the binary file has been successfully loaded.

Step 8. Reset the board to run the newly loaded firmware.

2.6 Sample application

2.6.1 Potentiometer example

The hardware setup for the example is:

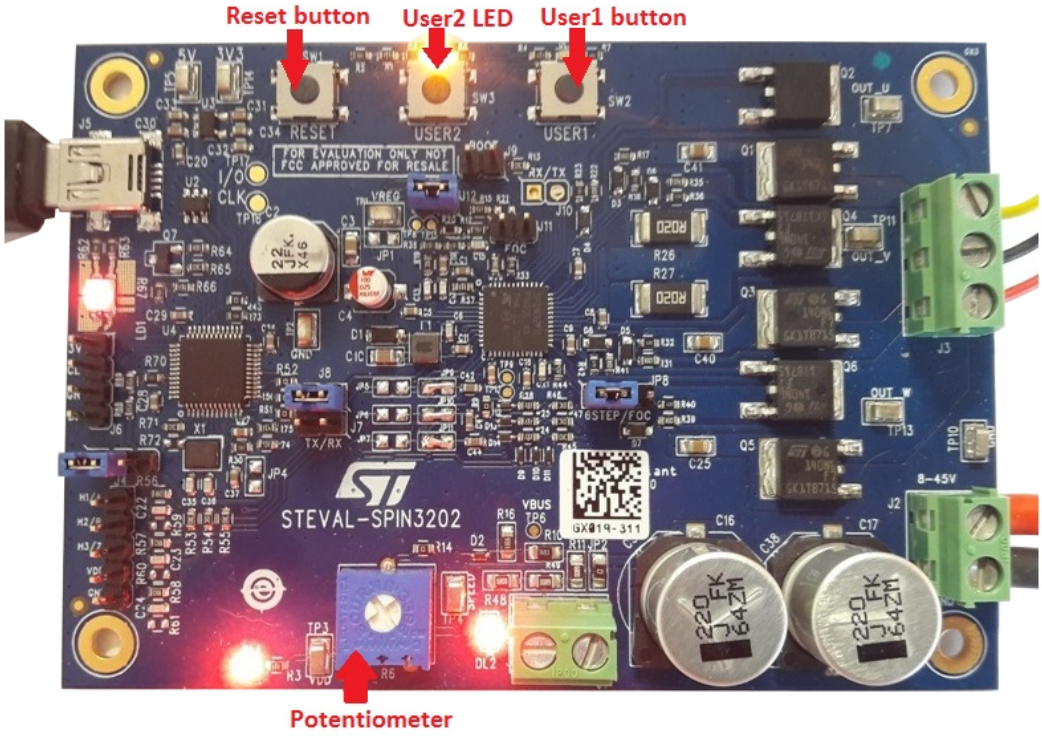
- The board correctly powered (USB powered and VBUS connected to a power supply)
- The three phases of the motor are correctly connected to the board (by default the target motor is the Bull-Running BR2804-1700 kV)

Push the USER1 button to start motor movement. The potentiometer (R6) can be used to dynamically adjust motor speed between 1200 RPM and 12000 RPM.

Push the USER1 button again to stop the motor.

If an error occurs while the motor is running, the LED of the “USER2” key lights up and the motor stops. Push the USER1 button to clear the failure.

Figure 17. **STEVAL-SPIN3202 HW user interface**



2.6.2 Uart example

The hardware setup for the example is:

- The board is correctly power supplied (USB powered and VBUS connected to a power supply)
- The three phases of the motor are correctly connected to the board (by default, the target motor is the Bull-Running BR2804-1700 kV)

The serial port interface can be used to control the motor. Open a serial port terminal on the USB connected to the PC and configure it as per the table below.

Then select the COM port corresponding to the evaluation board; it is labeled STMicroelectronics STLink Virtual COM Port.

Table 4. **Serial port configuration**

Parameter	Setting
Rx	CR
Tx	CR+LF
Baud rate	230400
Data	8 bit
Parity	none
Stop	1 bit
Transmit delay	10 ms/char

The table below lists the available commands and their actions.

Table 5. **Serial port commands**

Command	Action
STARTM	Start Motor
STOPMT	Stop Motor
DIRECT	Set Motor direction <0> CW or <1> CCW. The firmware prompt the direction after the command is sent.
SETSPD	Set Motor Speed (RPM). The firmware prompts the target speed after the command is sent.
GETSPD	Get Motor Speed (RPM)
STATUS	Get Status
MEASTA	Measurements transmission start
MEASTO	Measurements transmission stop
MEASEL	Measurement type <0> Speed <1> Bemf. The firmware prompt the type after the command is sent.
HELP	Show help menu
INIREF	Startup reference (0-4095). The firmware prompt the new value after the command is sent.
ACCELE	Set Motor acceleration (RPM/s). The firmware prompt the new value after the command is sent.
POLESP	Set the Motor pole pairs. The firmware prompt the new value after the command is sent.
KP-PRM	Set PI proportional term. The firmware prompt the new value after the command is sent.
KI-PRM	Set PI integral term. The firmware prompt the new value after the command is sent.

By default in the Uart example, the potentiometer feature is disabled. However, it is still possible to push the “USER1” button to start and stop the motor movement.

3 Implementation of six-step motor control algorithm with the STSPIN32F0x system-in-packages

3.1 Introduction to BLDC theory

A brushless three-phase motor consists of a fixed stator with three windings and a mobile rotor with an internal permanent magnet.

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. The third winding remains open. The successive steps are executed in the same way, except that the motor phase winding changes to generate a rotating stator field.

In Figure 18. BLDC motor control sequence, the red arrow indicates the stator flux vector rotation and the blue arrow indicates the direction.

A BLDC motor experiences a trapezoidal back electromagnetic force (BEMF) induced in the motor phase windings. The six-step driving is also named trapezoidal control because of the shape of the phase current. This control method provides maximum efficiency and minimum torque ripple as the motor is intrinsically built to be driven this way.

Figure 18. BLDC motor control sequence

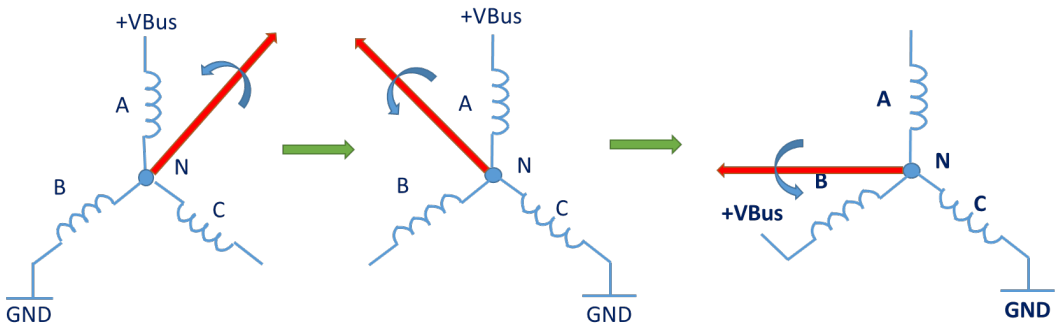
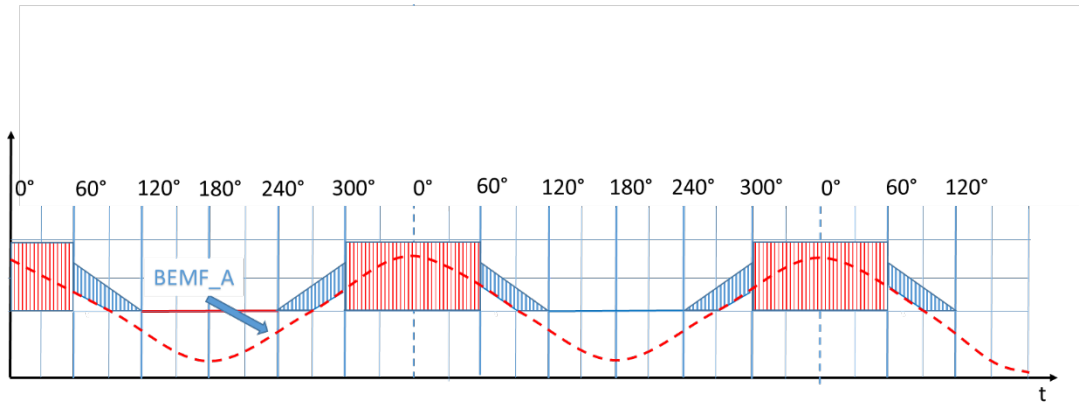


Figure 19. Six-step phase voltage with BEMF sequence and PWM modulation



3.2 Rotor synchronization

Six-step driving is a synchronous control drive, so maximum of the efficiency is achieved if the commutation between two consecutive steps is performed when the rotor is in the right position, where the BEMF signal and the phase current are synchronized and in phase.

Two methods can be implemented to perform rotor synchronization:

- **Sensored drive**: uses position sensors to determine rotor position.
- **Sensorless drive**: is based on the BEMF detection. It analyzes the zero crossing of the floating phase BEMF signal to establish the commutation point. The match between the BEMF signal of the floating phase with respect to the ground point is used to generate the commutations between two consecutive steps in order to achieve the rotor synchronization.

The 6Step Lib can use sensored (Hall effect sensors) or sensorless drive, depending on the symbols defined in the user project options.

3.3 Motor driver control

Six-step driving is performed by modulating the phase voltage through a PWM sequence generated by two different control methods:

Current mode (peak control): in this case, two control loops are needed, one inner and one outer. The inner loop generates, by hardware through an analog comparator, the PWM sequence for motor current regulation starting from an analog reference. It provides the torque control and inherent current limitation in the windings.The inner loop reference is generated by the outer loop that contains a digital PI regulator comparing the speed reference with the feedback of the motor speed.

Voltage mode: in this case, only one loop is needed. The PI regulator generates directly the PWM regulating the phase voltage comparing the speed reference with the feedback of the motor speed.

3.3.1 Current mode control

In Figure 20. Current mode control (STEVAL-SPIN3202), the inner loop contains all the components to acquire the motor current, amplify and condition the signal to be compared with a reference value. The output of the comparator is connected to the ETR timer function of the embedded MCU and possibly to the control logic of the gate drivers.

When the output of the comparator goes high and is connected to the gate driver logic, all high side gate driver outputs are switched off and, consequently, so are the external high side power switches. The high side switches are kept off until the next rising edge of the respective driving input. A similar mode of operation can be obtained using the MCU ETR timer function, but this is not implemented in the 6Step Lib as the mechanism already exist in the embedded gate driver logic. This mechanism regulates the duration of the PWMs on time.

Figure 20. Current mode control (STEVAL-SPIN3202)

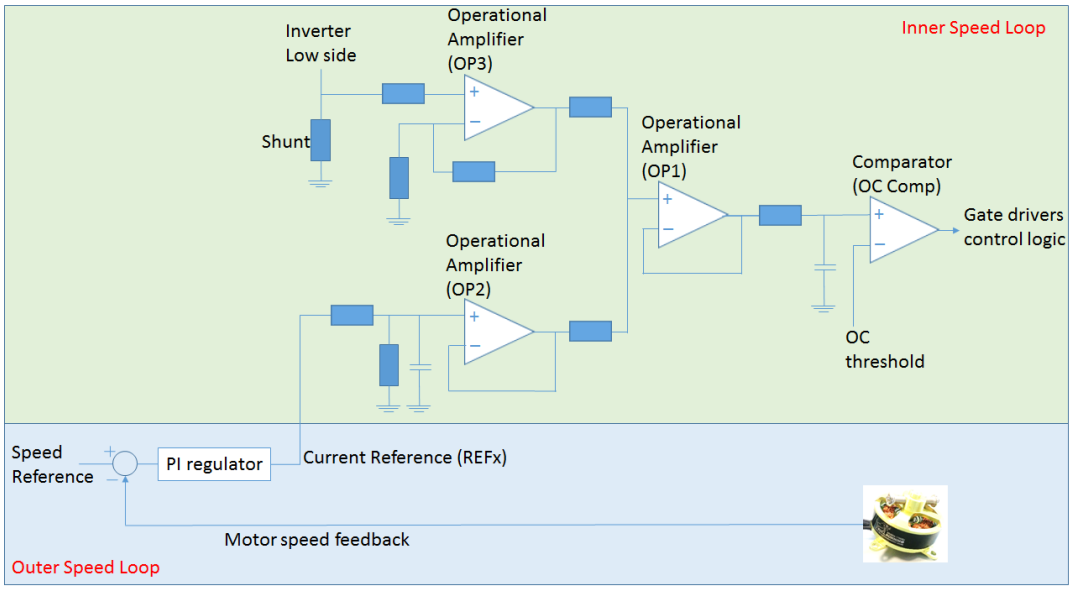


Figure 21. **Current mode control (STEVAL-SPIN3204)**

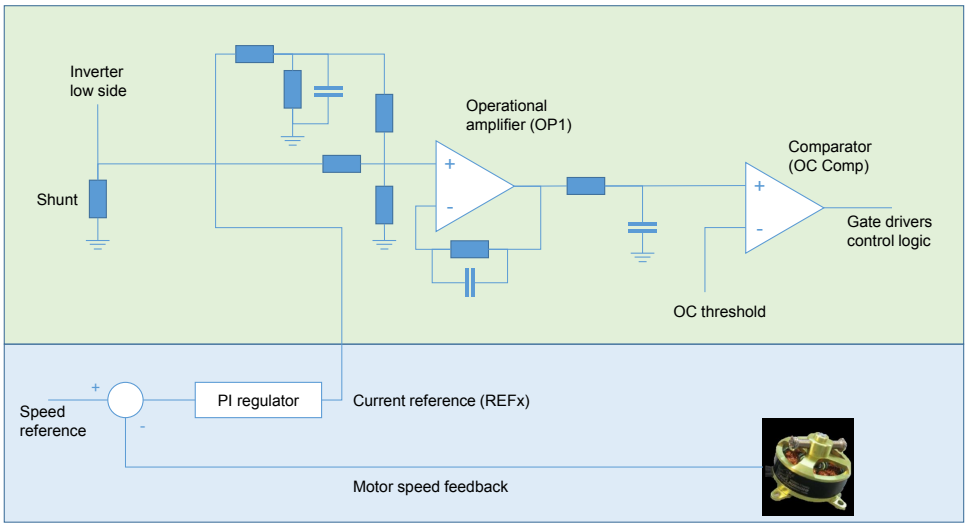
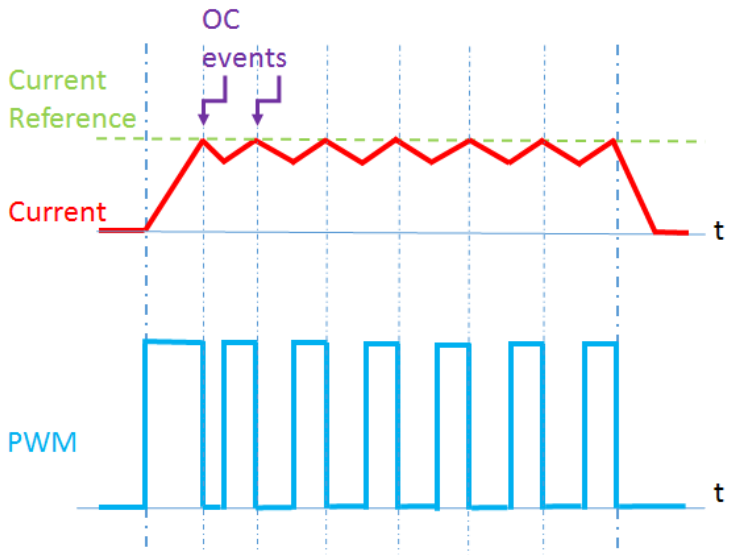


Figure 22. Gate Driver Control Logic function for Current mode control shows the Gate Driver Control Logic during normal operation. The red line is the motor current regulated at a fixed current reference (green). The blue wave is the PWM signal managed by the Gate Driver Control Logic. Whenever the conditioned motor current reaches the OC reference threshold, the Gate Driver Control Logic switches off of the internal high side switches and consequently the turns off the PWM signals at the external input of the high side power switches.

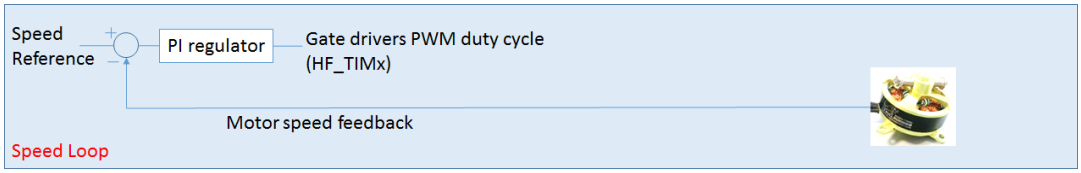
Figure 22. **Gate Driver Control Logic function for Current mode control**



3.3.2 Voltage mode control

Figure 23. Voltage mode control shows the Voltage mode driving method. The loop generates the output through a PI regulator that compares the Speed reference with the actual speed motor feedback. No Gate Driver Control Logic or ETR function is needed because the output calculates the new duty cycle value of PWM signals.

Figure 23. **Voltage mode control**



3.4 Sensorless drive methods

3.4.1 Rotor speed measurement

Each motor is built with a specific winding construction; we shall look at the sinusoidal and the trapezoidal structures. These can be distinguished by rotating the motor by hand and connecting a probe on two motor phases. The figures below shows the signal acquired on an oscilloscope.

Figure 24. **BEMF with Sinusoidal construction motor**

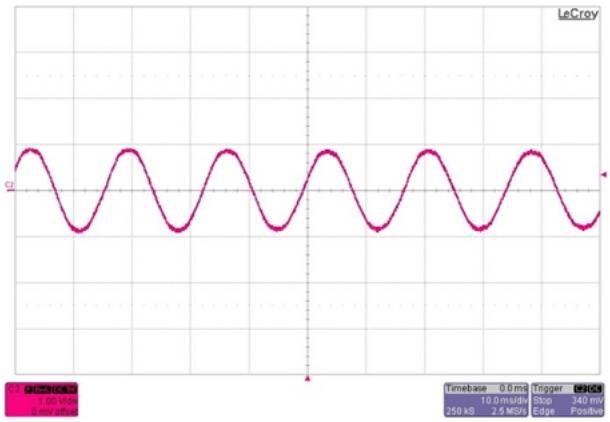
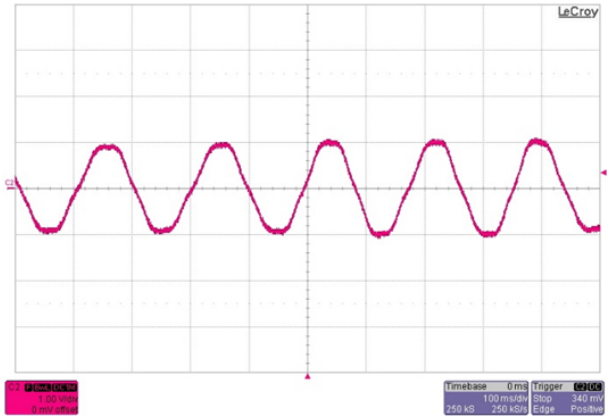


Figure 25. **BEMF with Trapezoidal construction motor**



The rotating magnetic field of the rotor induces a BEMF signal in the stator windings. If the motor is intrinsically sinusoidal or trapezoidal, the induced BEMF signals are periodic, and their frequency is proportional to the frequency of the motor turns. The proportional coefficient is the number of motor pole pairs: this number is used to calculate the mechanical speed of the motor. For instance, for 1000 Hz (electrical frequency) and 10 pole pairs (or 20 poles) the mechanical frequency is 100 Hz.

The motor speed in RPM can consequently be calculated by measuring the electrical frequency of the motor:

Equation 1: Motor speed (RPM)

$$Motor\ speed\ (RPM) = \frac{60}{Poles\ pairs\ \times\ step_{time}(s) \times 6} \quad (1)$$

Where $step_{time}$ is the duration (s) of a low frequency timer (LF_TIMx) that regulates the length of each step. It is multiplied by the number of steps and converted in RPM. The new value of $step_{time}$ is managed by the zero crossing calculation function and it is updated at the frequency of step generation (Step N to Step N+1).

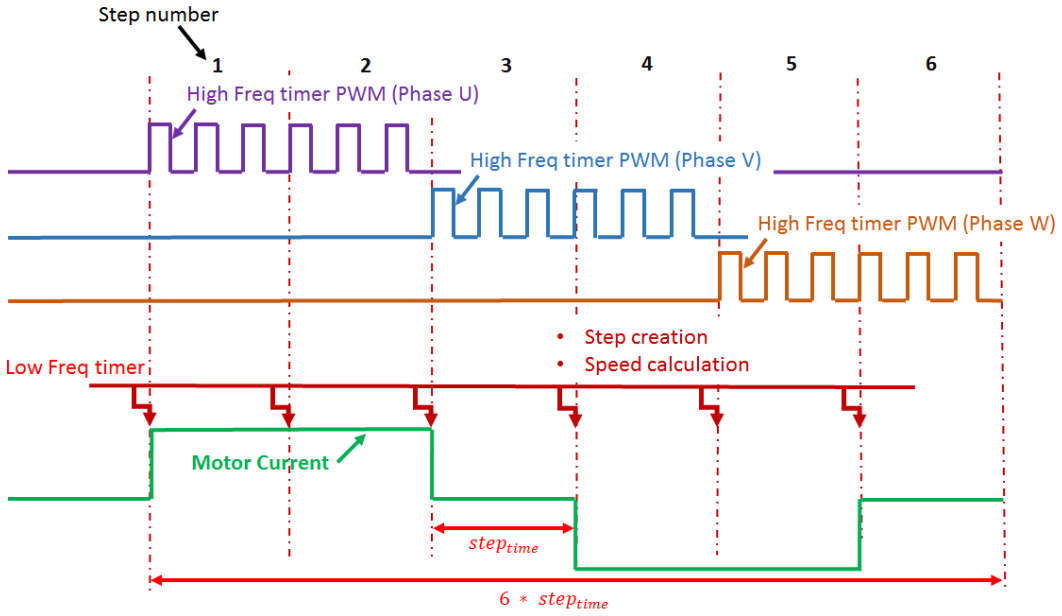
Equation 2 : Step time (s)

$$step_{time}(s) = \frac{LF\_TIMx\ duration \times\ LF\_TIMx\ prescaler\ on\ system\ clock}{system\ clock\ frequency\ \ (Hz)} \quad (2)$$

Where, LF_TIMx duration is in timer counter clock units and LF_TIMx prescaler is the division ratio between the system clock and the clock of the timer counter.

As shown in Figure 26. High and low frequency timer signals, each time the low frequency timer period elapses, a new step is created with the relevant high frequency timer PWM configuration and the mechanical speed is computed.

Figure 26. High and low frequency timer signals

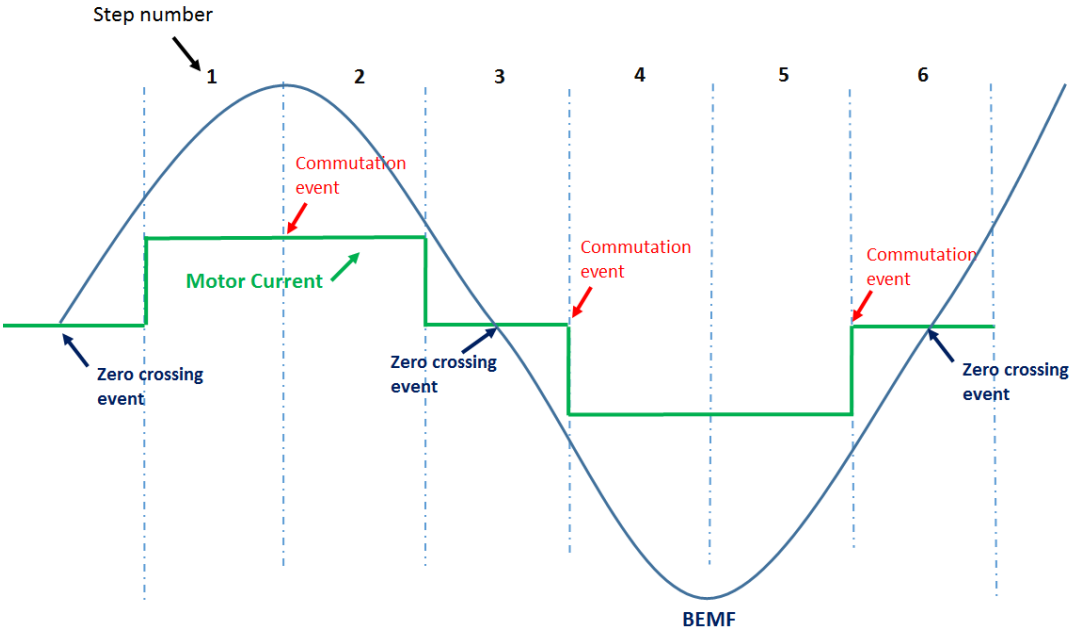


3.4.2 Commutation, demagnetization delay, zero crossing event

As the BLDC is a synchronous motor, the main objective of the drive is to ensure that the rotor position remains synchronized with the stator magnetic field. It is possible to demonstrate that the maximum efficiency of the drive is obtained by keeping the magnetic field of the stator with a 90-degree spatial advance with respect to the rotor magnetic field. This can be achieved by keeping the zero crossing in the middle of each step (see Figure 27. BEMF signal and Zero crossing detection).

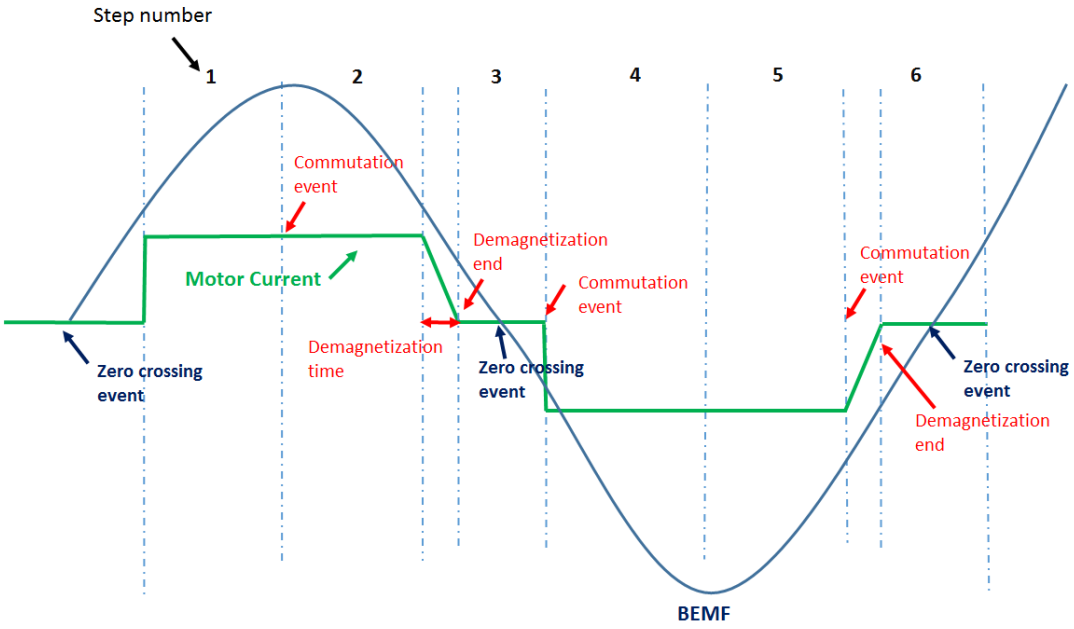
Since each motor has different inductance values, the current could reach zero with a delay at each step commutation. This “demagnetization” event must be handled and it determines the starting point for zero crossing detection.

Figure 27. **BEMF signal and Zero crossing detection**



This delay is a percentage of the step time (see Figure 28. Demagnetization time). The BEMF signal can be measured when the current in the floating phase reaches the zero value after the demagnetization delay time.

Figure 28. **Demagnetization time**



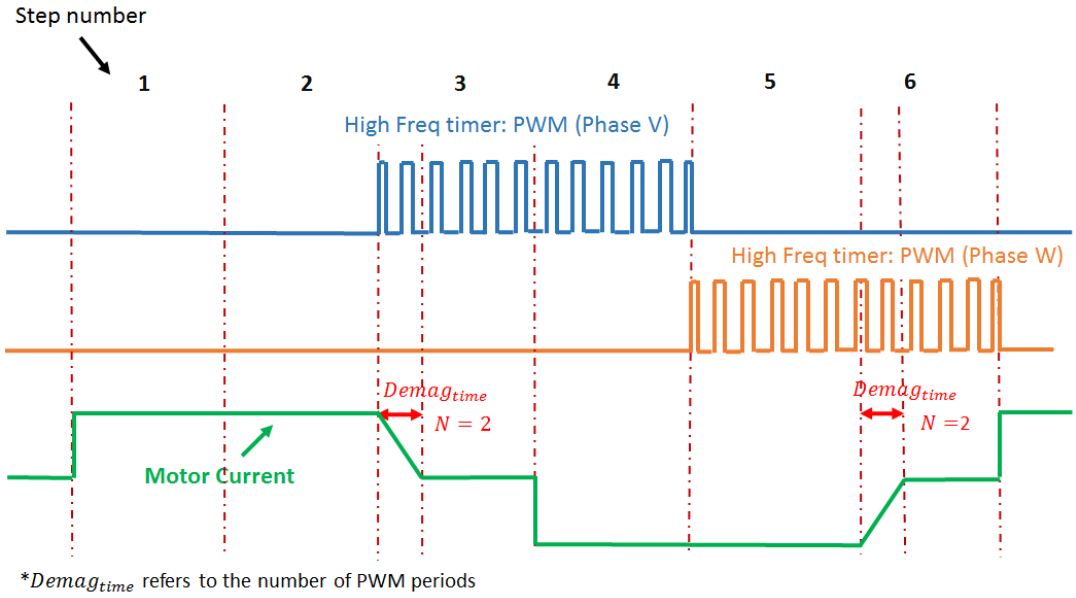
This method is managed by firmware and is based on the filtered speed feedback (inversely proportional to the step time). Moreover above a predefined speed threshold, a minimum predefined demagnetization delay is applied, allowing the reduction of microcontroller computation at high speeds.

The demagnetization delay time is calculated as a number of high frequency PWM periods.

Note: For a center aligned timer counter, the PWM period is twice the timer period.

In the STSW-SPIN3202 firmware package, the high frequency timer is center aligned in sensorless drive. The following figure shows the demagnetization delay for N = 2 in this configuration.

Figure 29. Demagnetization delay time with N = 2



3.4.3 Zero crossing detection

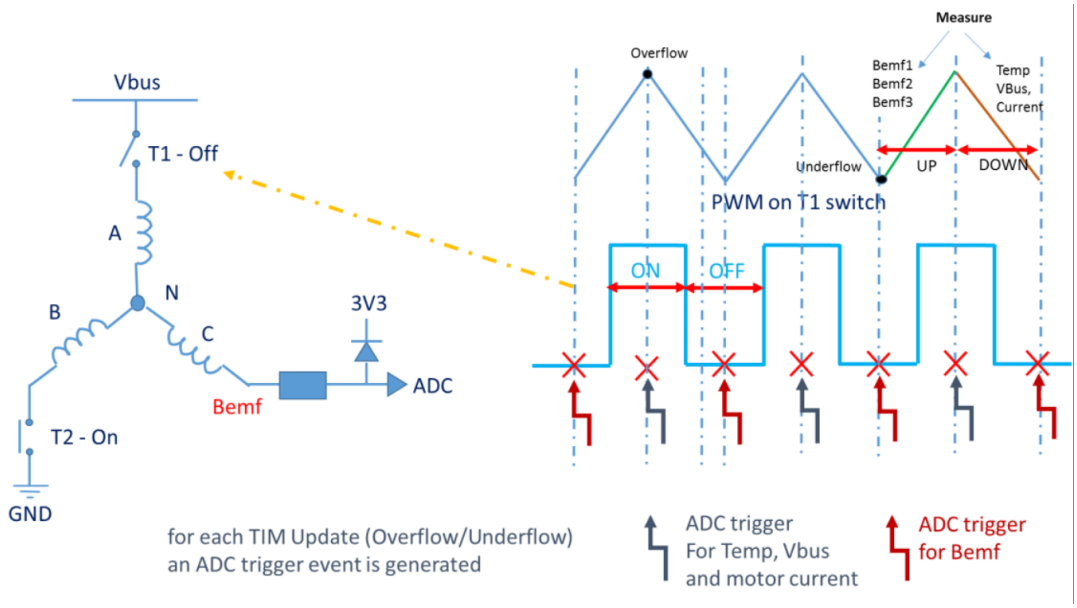
In order to synchronize the BLDC motor with the driver, the instant when the BEMF signal equals the motor neutral voltage is identified as the zero crossing point. There are two different scenarios to measure the BEMF value, depending on the high frequency PWM signals generated to modulate the high side devices:

The zero crossing can be detected both during the on and the off time of the PWM signal.

The 6Step Lib supports both BEMF measurement modes. However, by default, only the BEMF measurement based on T-off mode is enabled in the firmware package. To enable measurement during both T-on and T-off time, the PWM_ON_BEMF_SENSING symbol must be defined in the project options.

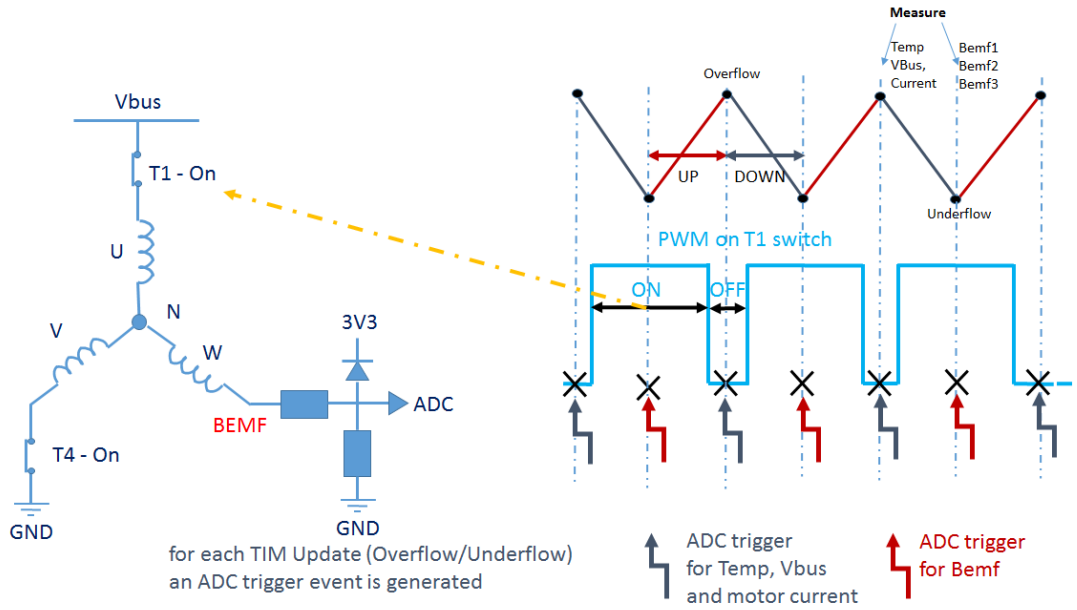
In both modes, the detection of the zero crossing point is performed by comparing the BEMF converted value to a fixed threshold. Figure 30. BEMF measurement during PWM Off time and ADC sampling shows the BEMF measurement during the T-OFF of T1 switch, when phase W is floating and the ADC sampling method is used to measure the BEMF signals, VBUS, temperature, motor current etc.

Figure 30. **BEMF measurement during PWM Off time and ADC sampling**



The following figure shows the BEMF measurement during the T-ON of T1 switch, which can be used for large PWM duty cycles.

Figure 31. **BEMF measurement during PWM On time and ADC sampling**



3.4.4 Startup procedure

To perform sensorless control on a BLDC motor, the BEMF must reach an appreciable value to be readable. Since the BEMF amplitude increases with rotor speed, it is readable at a speed above a specific threshold. Thus the motor has to be forced to rotate first without BEMF feedback. For this reason, it is important to implement an asynchronous startup strategy that brings the motor up to the required speed.

Two different steps are needed to start the motor and begin closed loop control with BEMF detection:

1. Alignment phase
2. Ramp generation phase

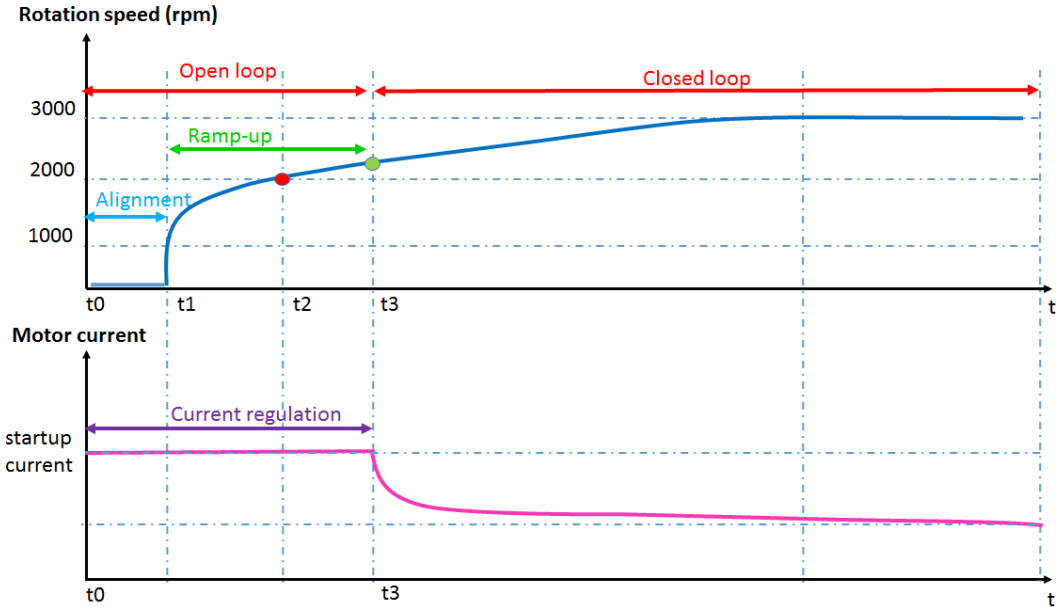
3.4.4.1 *Alignment phase*

During the alignment phase, the rotor is forced to remain at a specified position with a specific switch configuration. A fixed current (managed by firmware) is maintained in order to move the rotor to the closer pole pair position. The alignment is performed with the choice of a specific step number.

3.4.4.2 *Ramp generation phase*

To perform good BEMF control, the motor is started with a speed ramp profile. During this time, a fixed and controlled current is maintained to run and accelerate the motor. The control system monitors the motor speed and checks whether this values reaches a specified threshold (managed by firmware, red circle in Figure 32. Alignment and Ramp-up generation). In case of a positive check result, the zero crossing events counter starts and, at a specific value (managed by firmware, green circle in Figure 32. Alignment and Ramp-up generation), determines the starting point for closed loop operation. This condition is reached if the number of zero crossing events reaches a threshold (settable in the firmware).

Figure 32. Alignment and Ramp-up generation



4 Getting started with the six-step firmware library

4.1 Hardware resource mapping

The stspin32f0.h and STEVAL-SPIN3202.h/STEVAL-SPIN3204.h files interface between the MCU and the 6-Step library.

If the high frequency timer (PWM) is TIM1, the stpin32f0.h file maps the TIM1 MCU data structure with a generic name BSP_SIP_HF_TIMx used by the demonstration code in main_32F0.c HF timer init function to attach a generic timer handle HF_TIMx used by the 6Step Lib to the TIM1 MCU data structure:

HF_TIMx.Instance = BSP_SIP_HF_TIMx;

If you change the timer, you must update this file with the right peripheral. The following table lists the main peripherals mapped for sensorless voltage mode driving.

Table 6. Main peripherals mapping for sensorless coltage mode driving

6Step Lib generic handle	Drivers peripheral definition	MCU date structure
HF_TIMx	BSP_SIP_HF_TIMx	TIM1
LF_TIMx	BSP_BOARD_LF_TIMx	TIM3
ADCx	BSP_SIP_ADCx	ADC1
huart	BSP_SIP_UART	USART1

Another example of mapping is the ADC. The GPIOs used by the ADC are board design dependent. Thus, the GPIO mapping for the ADC is in the STEVAL-SPIN3202.h/STEVAL-SPIN3204.h file, whereas the ADC peripheral mapping is in the stspin32f0.h file as it is only integrated circuit dependent.

With the STEVAL-SPIN3202 board in sensorless voltage mode driving, the ADC is used to sense on:

- PA0, PA1 and PA2 the BEMF
- PA3 the potentiometer voltage regulating the speed
- PA4 the voltage in the current sense shunt resistor
- PB1 the VBUS supply voltage for the power MOSFETS

Hence in STEVAL-SPIN3202.h, the definitions are:

```
#define BSP_BOARD_ADCx_GPIOA              (GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4)
```

```
#define BSP_BOARD_ADCx_GPIOB              (GPIO_PIN_1)
```

If, perhaps for board design reasons, it is preferable to use PA5 for the speed sensing, the above definitions would be modified as shown below:

```
#define BSP_BOARD_ADCx_GPIOA              (GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_5|GPIO_PIN_4)
```

```
#define BSP_BOARD_ADCx_GPIOB            (GPIO_PIN_1)
```

4.2 Motor control parameters

The following tables explain all the motor control parameters with explanations. Some constants are calculated by the compiler preprocessor and cannot be modified.

Table 7. Motor control basic parameters

Constant name	Description	Type, Unit	Modifiable
NUM_POLE_PAIRS	Number of pole pairs of BLDC motor	16bit unsigned	Yes
DIRECTION	Set the motor direction: (0) for CW or (1) for CCW.	0 or 1	Yes
TARGET_SPEED_OPEN_LOOP	Target speed in open loop phase (ramp up)	32bit unsigned, RPM	Yes
TARGET_SPEED	Target speed on closed loop phase when the potentiometer is disabled	32bit unsigned, RPM	Yes

Table 8. Motor control advanced voltage mode parameters

Constant name	Description	Type, Unit	Modifiable
STARTUP_DUTY_CYCLE	Tenths of percentage of high frequency gate driving PWM on time. In example, for a percentage of 20%, the STARTUP_DUTY_CYCLE is 200.	16bit unsigned, between 1 and 1000	Yes
KP_GAIN	Kp parameter for PI(D) regulator.	16bit unsigned	Yes
KI_GAIN	Ki parameter for PI(D) regulator.	16bit unsigned	Yes
KD_GAIN	Kd parameter for PID regulator.	16bit unsigned	Yes
K_GAIN_SCALING	Kp, Ki, (Kd) scaling for PI(D) regulator.	8bit unsigned, typically 14	Yes
LOWER_OUT_LIMIT	Low Out value of PI regulator.	16bit signed	Yes
UPPER_OUT_LIMIT	High Out value of PI regulator.	16bit signed	No
MAX_POT_SPEED	Maximum Speed regulated by potentiometer.	32bit unsigned, RPM	Yes
MIN_POT_SPEED	Minimum Speed regulated by potentiometer.	32bit unsigned, RPM	Yes

Table 9. Motor control advanced current mode parameters

Constant name	Description	Type, Unit	Modifiable
STARTUP_PEAK_CURRENT	Startup Peak Current.	16bit unsigned, mA	Yes
SENSE_RESISTOR	Shunt resistor value.	16bit unsigned, mOhms	Yes
SENSE_RESISTOR_VOLTAGE	Shunt resistor voltage.	16bit unsigned, mV	No

Constant name	Description	Type, Unit	Modifiable
SENSE_GAIN	Gain of the operational amplifier having for input the shunt resistor voltage. In example for a gain of 5.6, the SENSE_GAIN is 5600.	16bit unsigned, Thousandths	Yes
SENSE_AMPLIFIED_VOLTAGE	Amplified shunt resistor voltage	16bit unsigned, mV	No
REFERENCE_PWM_HIGH_VOLTAGE	Reference PWM logic high voltage (VDD).	16bit unsigned, mV	Yes
REFERENCE_PWM_DIVIDER_RATIO	Divider ratio of resistors bridge on the reference PWM. In example, for a division of input voltage by 3.2, the REFERENCE_PWM_DIVIDER_RATIO is 3200.	16bit unsigned, Thousandths	Yes
OC_THRESHOLD	Internal OC threshold of the 3-phase controller with embedded MCU circuit.	16bit unsigned, mV, 100, 250 or 500	Yes
STARTUP_CURRENT_REF_THDS	Intermediate constant for STARTUP_CURRENT_REFERENCE preprocessor computation.	16bit unsigned	No
STARTUP_CURRENT_REFERENCE	Internal startup current reference corresponding to the startup peak current and used to compute the reference PWM duty cycle.	16bit unsigned	No
KP_GAIN	Kp parameter for PI(D) regulator	16bit unsigned	Yes
KI_GAIN	Ki parameter for PI(D) regulator	16bit unsigned	Yes
KD_GAIN	Kd parameter for PID regulator	16bit unsigned	Yes
K_GAIN_SCALING	Kp, Ki, (Kd) scaling for PI(D) regulator	8bit unsigned, typically 14	Yes
LOWER_OUT_LIMIT	Low Out value of PI regulator	16bit signed	Yes
UPPER_OUT_LIMIT	High Out value of PI regulator	16bit signed	No
MAX_POT_SPEED	Maximum Speed regulated by potentiometer	32bit unsigned, RPM	Yes
MIN_POT_SPEED	Minimum Speed regulated by potentiometer	32bit unsigned, RPM	Yes

Table 10. Motor control advanced common parameters – Gate driving

Constant name	Description	Type, Unit	Modifiable
GATE_DRIVING_PWM_FREQUENCY	PWM frequency for the gate drivers.	32bit unsigned, Hz	Yes
SYSCLOCK_FREQUENCY	System clock frequency of the embedded MCU.	32bit unsigned, Hz	Yes
LF_TIMX_PSC	Low frequency timer prescaler.	32bit unsigned	Yes
LF_TIMX_ARR	Low frequency timer period.	32bit unsigned	Yes
HF_TIMX_PSC	High frequency timer prescaler.	32bit unsigned	Yes
HF_TIMX_ARR	High frequency timer period.	32bit unsigned	No
DEAD_TIME	Time during both complementary channels of the high frequency PWM are off in clock pulses. Please refer to STM32F0x1 reference manual for more details.	32bit unsigned, between 0x00 and 0xFF	Yes

Constant name	Description	Type, Unit	Modifiable
PULSE	Duration of the PWM on time at the initialization of the high frequency timer. Defines the maximum duty cycle achievable in Current Mode control.	16bit unsigned	Yes

Table 11. Motor control advanced common parameters – Hall effect sensors

Constant name	Description	Type, Unit	Modifiable
START_COUNTER_STEPS_DECREMENTATION	Number decrementing a start counter during the STARTUP state. See NUMBER_OF_STEPS for more details.	32bit unsigned	Yes
NUMBER_OF_STARTS	Number of start attempts to be failed before calling entering the STARTUP_FAILURE state.	8bit unsigned	Yes
HALL_KO_SUCCESSIVE_MAX	Maximum number of low frequency timer delay consecutive elapsing without a hall status change in-between. When this number is overflowed, the MC_SixStep_Hall_Startup_Failure_Handler is called.	8bit unsigned	Yes
MAX_SPEED	Motor rated max speed. This number is used to compute the STEP_DURATION_MINIMUM.	32bit unsigned, RPM	Yes
STEP_DURATION_MINIMUM	Minimum step time duration above which the speed measurement is updated (number of low frequency timer counter clocks).	16bit unsigned	No
COMMUTATION_DELAY	Delay between hall capture and step commutation (number of low frequency timer counter clocks) when FIXED_HALL_DELAY option is enabled	16bit unsigned	Yes

Table 12. Motor control advanced common parameters – Open loop control

Constant name	Description	Type, Unit	Modifiable
ACC	Mechanical acceleration rate used during ramp-up and speed ramp generation.	32bit unsigned, RPM/s	Yes
MINIMUM_ACC	Minimum mechanical acceleration rate (for high inertia motor).	32bit unsigned, RPM/s	Yes
NUMBER_OF_STEPS	Sensorless drive: Maximum number of steps for acceleration during the open loop phase. If the closed loop validation is not completed at the end of this time a speed fault error will be generated. Hall sensing drive: Number initializing a start counter for the STARTUP state. Each hall status change decreases the start counter by START_COUNTER_STEPS_DECREMENTATION. When the start counter is lower or equal to 0, the motor enters the RUN state.	32bit unsigned	Yes
TIME_FOR_ALIGN	Time for alignment	16bit unsigned, ms	Yes

Constant name	Description	Type, Unit	Modifiable
BUTTON_DELAY	Delay time to enable push button for new command	32bit unsigned, ms	Yes
NUMBER_ZCR	Number of zero crossing events in sequence for closed loop validation	32bit unsigned	Yes

Table 13. Motor control advanced common parameters – Closed loop control

Constant name	Description	Type, Unit	Modifiable
INITIAL_DEMAGN_DELAY	Initial value for delay time during startup for BEMF detection.	16bit unsigned, number of gate driving PWM periods	Yes
DEMAG_TIME_STEP_RATIO	Tenths of percentage of step time allowed for demagnetization. In example, for a demagnetization delay equals to 27% of a step time, the DEMAG_TIME_STEP_RATIO is 270.	16bit unsigned, Tenths of percentage	Yes
K_DEMAG	Proportional parameter to compute the number of gate driving PWM periods before checking BEMF threshold.	16bit unsigned	No
DEMAG_SPEED_THRESHOLD	Mechanical speed threshold above which the MINIMUM_DEMAGN_DELAY is used.	32bit unsigned, RPM	Yes
MINIMUM_DEMAGN_DELAY	Demagnetization delay applied above DEMAG_SPEED_THRESHOLD.	32bit unsigned, number of gate driving PWM periods	Yes
BEMF_THRSLD_DOWN_OFF	Zero Crossing threshold during OFF time.	16bit unsigned, between 0 and ADC FS.	Yes
BEMF_THRSLD_UP_OFF	Zero Crossing threshold during OFF time.	16bit unsigned, between 0 and ADC FS.	Yes
BEMF_THRSLD_DOWN_ON	Zero Crossing threshold during ON time.	16bit unsigned, between 0 and ADC FS.	Yes
BEMF_THRSLD_UP_ON	Zero Crossing threshold during ON time.	16bit unsigned, between 0 and ADC FS.	Yes
DUTY_CYCLE_50	Duty cycle threshold above which the BEMF is sensed during ON time when the PWM_ON_BEMF_SENSING symbol is defined.	16bit unsigned, number of high frequency timer counter clocks	Yes
SPEED_LOOP_TIME	Time for new execution of the speed loop.	16bit unsigned, ms	Yes
FILTER_DEEP_SHIFT	Number used to compute the speed filter length which is 2^FILTER_DEEP_SHIFT.	8bit unsigned	Yes
FILTER_DEEP	Number of bits for the speed digital filter.	8bit unsigned	No
POT_BUFFER_SIZE_SHIFT	Number used to compute the potentiometer filter length which is 2 + 2^POT_BUFFER_SIZE_SHIFT.	8bit unsigned	Yes
POT_BUFFER_SIZE	Number of bits for the potentiometer digital filter.	8bit unsigned	No
ADC_SPEED_TH	Fixed threshold to change the target speed.	16bit unsigned	Yes
BEMF_CONSEC_DOWN_MAX	Maximum value of BEMF Consecutive Threshold Falling Crossings Counter in closed loop.	8bit unsigned	Yes

Constant name	Description	Type, Unit	Modifiable
BEMF_CNT_EVENT_MAX	Maximum number of BEMF Counter in open loop (Motor stall detection).	16bit unsigned	Yes

Table 14. Motor control advanced common parameters –Debug

Constant name	Description	Type, Unit	Modifiable
GPIO_ZERO_CROSS	Enable (1) the GPIO toggling output for zero crossing detection.	0 or 1	Yes
GPIO_COMM	Enable (1) the GPIO toggling output for commutation detection.	0 or 1	Yes
GPIO_ZCR_MODE	Enable (1) the GPIO echo of TON/TOFF BEMF sensing method.	0 or 1	Yes

4.3 6Step Lib main functions

4.3.1 6Step Lib algorithm core functions

4.3.1.1 Sensorless drive

- `MC_SixStep_ARR_step()`: generate the ARR value for the low frequency timer during start-up
- `MC_SixStep_Ramp_Motor_calc()`: calculate the acceleration profile step by step for motor during start-up
- `MC_SixStep_NEXT_step()`: generate the next step number according to the direction (CW or CCW)
- `MC_Task_Speed()`: Speed Loop with PI regulator
- `MC_ADCx_SixStep_Bemf()`: manage the ADC measurements and compute the zero crossing detection
- `MC_SixStep_TABLE()`: set the peripherals (TIMx, GPIO etc.) for each step

4.3.1.2 Hall effect sensors feedback drive

- `MC_TIMx_SixStep_CommutationEvent()`: get the timing of the Hall effect sensors status change and program the step commutation
- `MC_SixStep_NEXT_step()`: get the Hall effect sensors status, generate the next step number according to the direction and set the peripherals for each step
- `MC_Task_Speed()`: Speed Loop with PI regulator

4.3.2 6Step Lib Motor Control API

The exported API functions are inside the 6Step_Lib.h header file, which also contains all the variables and structures for the six-step algorithm.

`MC_SixStep_INIT()`: init the main variables for motor driving from MC_SixStep_param.h

`MC_SixStep_RESET()`: reset all variables used for 6Step control algorithm

`MC_StartMotor()`: start the motor

`MC_StopMotor()`: stop the motor

`MC_Set_Speed()`: set the new motor speed value

`MC_EXT_button_SixStep()`: handling of push button event

4.3.3 6Step Lib tasks

Three underlying tasks run at different frequencies and with different priority levels, according to the specific functions they cover.

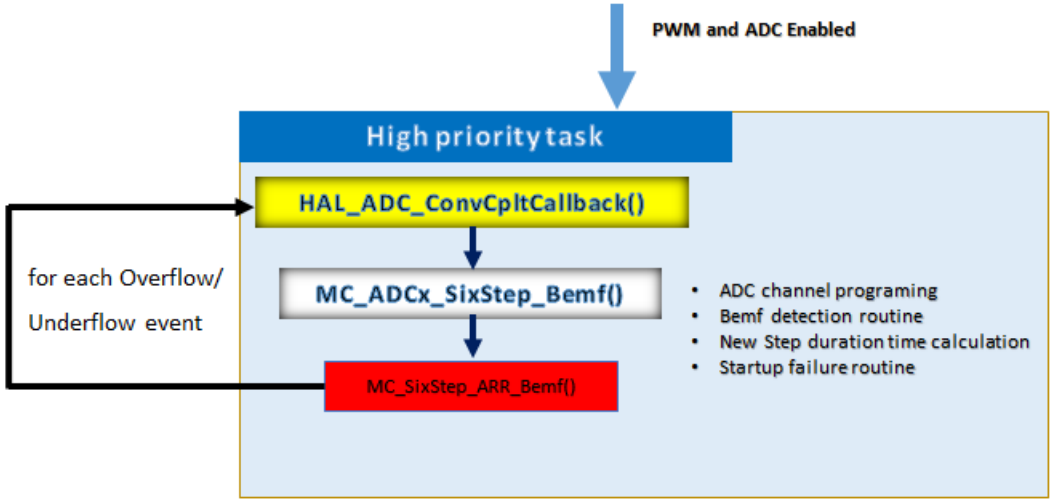
4.3.3.1 Sensorless drive high priority task

This task relates to high frequency functions (gate driving PWM generation, ADC conversion and reading) and it is managed at the highest priority. The frequency of this task is twice the gate driving PWM frequency (GATE_DRIVING_PWM_FREQUENCY) and can be changed in the motor control parameter file.

The ADC callback is called two times in a gate driving PWM period. In particular, at the beginning of timer counter countdown, a Bemf signal from the three motor phases is converted by the ADC.

As a user option, when a timer counter underflow occurs, one of the following signals is acquired in sequence: potentiometer (speed), temperature, vbus and motor current.

Figure 33. Sensorless high priority task

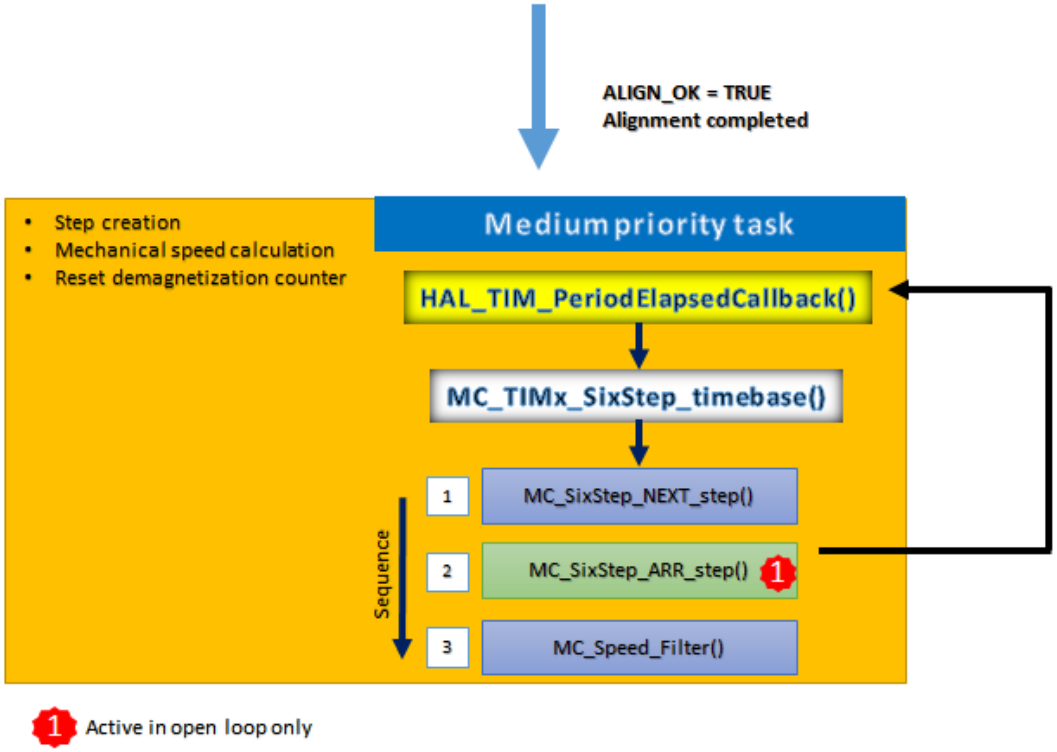


4.3.3.2 *Sensorless drive medium priority task*

This task relates to the low frequency timer for step generation and is managed at a medium priority. The frequency of this task depends on the current speed of the motor.

The timing of this callback is based on the Low Frequency timer. It changes dynamically and is inversely proportional to the electrical frequency of the motor; i.e., the time decreases when the motor speed increases.

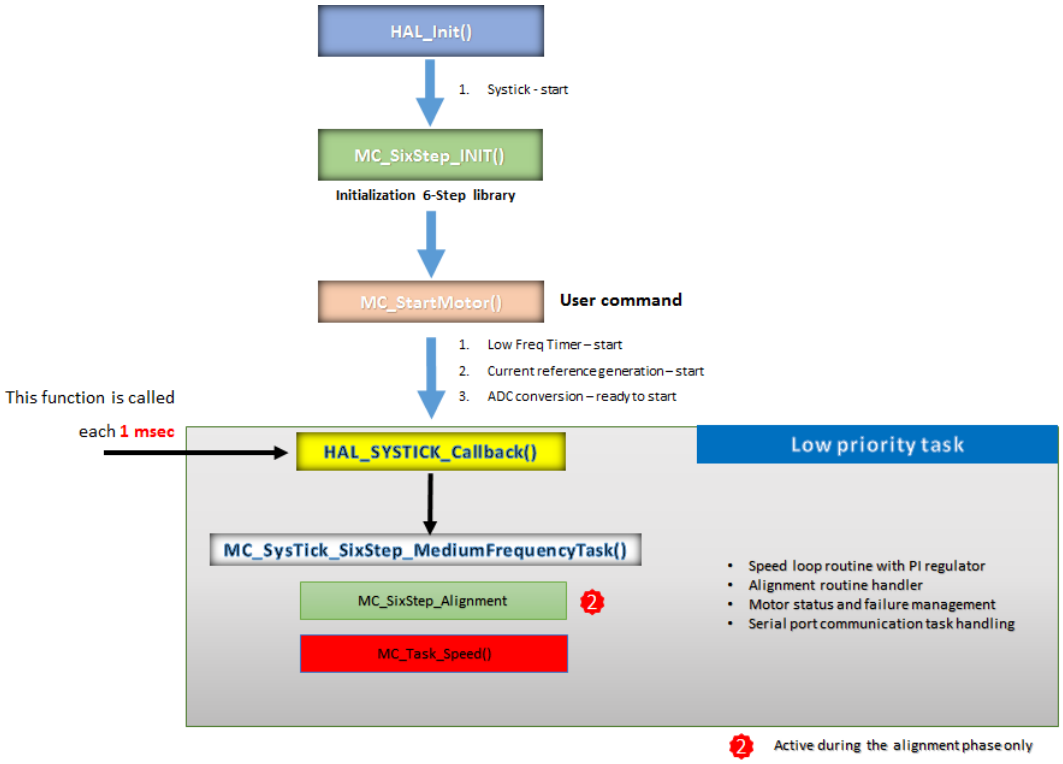
Figure 34. **Medium priority task**



4.3.3.3 *Sensorless drive low priority task*

This task relates to the loop timing speed, motor state management and serial port communication. It is managed at the lowest priority. The `MC_SysTick_SixStep_MediumFrequencyTask()` is called at SysTick frequency (1 ms), while the speed loop function is managed by SPEED_LOOP_TIME (ms) defined in motor control parameter file.

Figure 35. **Sensorless drive low priority task**



5 Designing a user application using the firmware library

5.1 Running a different sensorless BLDC motor

The firmware package provides some configuration examples for the sensorless driving operation with different motors. Since each motor has its own specifications, the package is designed to simplify adaptation to different motors.

You can use one of the supplied motor control parameter header files MC_SixStep_param_<motor model identifying string>.h as your template, with several parameters grouped into the following categories:

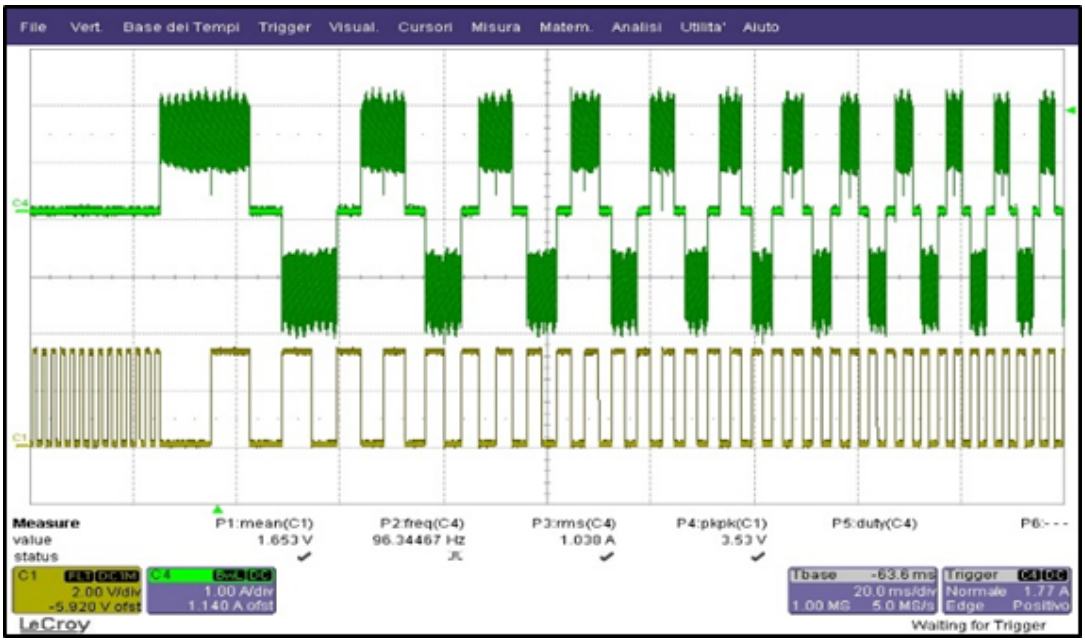
- basic: to manage the main parameters like motor pole pairs, motor direction and motor target speed in open and closed loop.
- advanced: to set the PI parameters, define the alignment time or the acceleration rate during startup, change the zero crossing threshold and other fine tuning parameters.

To spin a generic sensorless BLDC motor, we suggest the following procedure:

1. open the package
2. open the MC_SixStep_param_BR2804_1700_KV_1.h file and set the main parameters, in particular:
   - MOTOR POLE PAIRS
   - DIRECTION
   - TARGET_SPEED_OPEN_LOOP
   - TARGET_SPEED
   - STARTUP_DUTY_CYCLE in voltage mode or STARTUP_PEAK_CURRENT in current mode
   - ACC (for high inertia motor reduce a lot the default value)
   - TIME_FOR_ALIGN
3. set GPIO_ZERO_CROSS to 1, set GPIO_COMM to 1, and connect two voltage passive probes on default MCU pins (PA5, PA15) for zero crossing and commutation event monitoring
4. compile the code and upload it onto the board
5. power on the board and press the USER1 button to start the board

If the above values are valid and the default parameters are appropriate for the connected motor, the system will start as shown below.

Figure 36. **New motor running**



In the above figure, the green signal shows the motor phase current and the yellow channel the commutation event. It is also possible to determine the zero crossing event with a probe: the zero crossing count starts when the speed threshold TARGET_SPEED_OPEN_LOOP is reached during startup; after this event, a toggle signal is generated as output.

If the motor does not start or enters a fault state, you should check the parameters again:

- Check that NUM_POLE_PAIRS is correct.
- Check that TIME_FOR_ALIGN is long enough for motor alignment.
- Check STARTUP_DUTY_CYCLE in voltage mode or STARTUP_PEAK_CURRENT in current mode during the open loop stage. If the motor does not move or it stalls, increase this value. If this value is too big an OVERCURRENT error occurs, disabling the PWMs.
- Check that the ACC (acceleration time) is not too large. For motors with large inertia, reduce the value for smoother starts over longer periods. If the ACC is too small, the motor may never reach the speed threshold for closed loop validation and trigger a STARTUP_FAILURE error.
- Check the number NUMBER_ZCR (Zero Crossing event) for open loop end and closed loop validation phase. This number may need to be reduced. Plot the Zero Crossing and Step commutation event through the default GPIO pin. When the ZC signal appears, the speed is validated and the count of ZC consecutive events (n_zcr_startup ++) is started. If this number reaches NUMBER_ZCR on consecutive occasions, the closed loop control starts. In there are no consecutive ZC detections, the algorithm resets the counter (n_zcr_startup = 0) and increases another counter (cnt_bemf_event ++). This last counter generates a STARTUP_BEMF_FAILURE if it reaches the BEMF_CNT_EVENT_MAX.
- Decrease the speed threshold for closed loop validation (TARGET_SPEED_OPEN_LOOP).
- For closed loop control, tune the KP_GAIN and KI_GAIN by using, for instance, the speed recording script to check that the speed response to a start command or to a speed change behaves as expected. KP_GAIN and KI_GAIN are not calculated by default and it is better to adjust them during testing. To start the tuning, you should set KI_GAIN 0 and increase KP_GAIN until the motor speed is clearly oscillating; then divide the KP_GAIN by 2 and start increasing the KI_GAIN, taking care to always have KI_GAIN < KP_GAIN. To increase both KP_GAIN and KI_GAIN by the same factor, K_GAIN_SCALING can be decreased. Reducing K_GAIN_SCALING can also be used to increase the resolution for KP_GAIN and KI_GAIN. In case of an OVERCURRENT error, reduce the reaction of the PI regulator with lower values of KP and KI.

5.2 Use of scripts to tune the motor control

Python 2.7.13 or above must be installed on your computer (https ://www.python.org/downloads/release/python-2713 /) to use the scripts in the “Utilities” folder.

Moreover, the following modules must be added by entering the corresponding commands in a command line window from the Python folder:

- Pyserial: `python -m pip install pyserial`
- Numpy: `python -m pip install numpy`
- Matplotlib: `python -m pip install matplotlib`

The STSPIN32F0A embedded firmware must output data expected by the python scripts to make use of them. For this purpose, some symbols must be listed in the “Defined symbols” list in the “Preprocessor” panel in the “C/C++ Compiler” category in the IAR project “Options”:

- SpeedRecording.py: UART_COMM and SPEED_SENDING
- BemfRecording.py: UART_COMM and BEMF_RECORDING
- StartStopTest.py: UART_COMM

5.2.1 SpeedRecording script usage

With the SpeedRecording.py script, you can control the motor speed (SETSPD command), change the regulator parameters (KP-PRM, KI-PRM commands) and see how it modifies the speed response.

The graph pops up when you press enter. The average speed and the standard deviation of the last 1000 speed samples are also displayed.

The syntax is:

```
python SpeedRecording.py <COM port number> <Speed loop time (ms)> [<xmax (ms)> [<ymax (RPM)>]
]
```

Figure 37. Command window example for SpeedRecording script

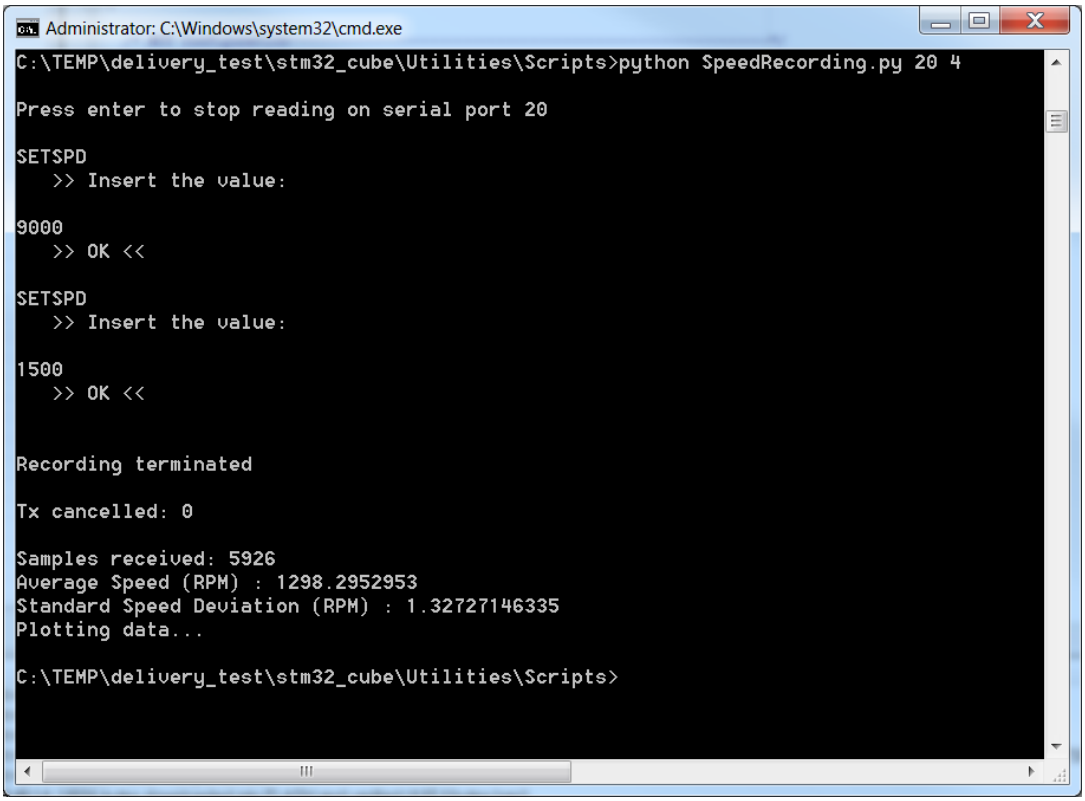
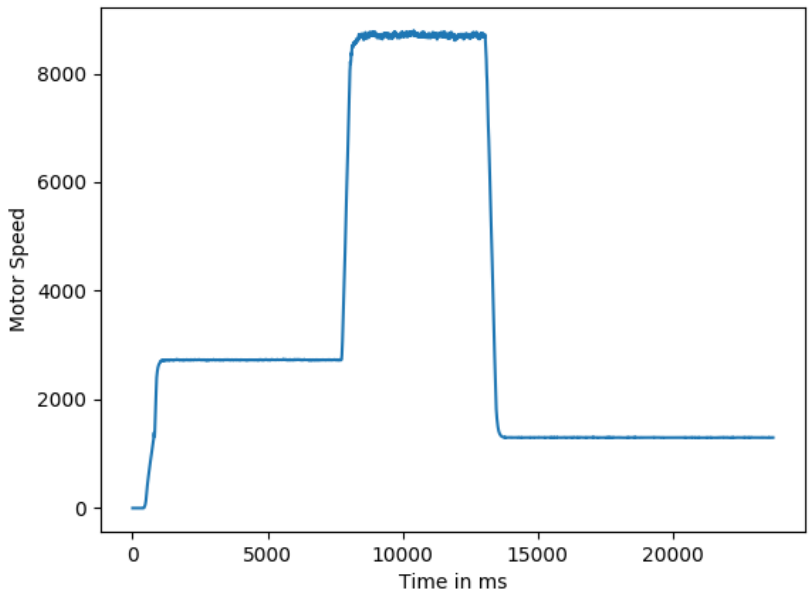


Figure 38. **Plot output example for SpeedRecording script**



5.2.2 BemfRecording script usage

With the BemfRecording script, you can view the step position and BEMF voltage on the same graph. You can also enter motor control commands during the recording.

The recording stops and the graph is displayed when you press enter.

The syntax is:

```
python BemfRecording.py <COM port number>
```

5.2.3 StartStopRecording script usage

With the StartStop script, you can launch a series of start and stop commands and check that the motor starts and reaches the target speed, and then stops when commanded.

The syntax is:

```
python StartStopTest.py <COM port number> <start stop iterations> <target speed> <speed excursion> <stop time>
```

5.3 How to use the API function for application development

User code development is simplified with a several APIs provided by the 6Step Lib.

The API functions are available at the application layer (main_32F0.c), and you can start or stop the motor and change the speed during the normal operation.

The `MC_SixStep_INIT()` function initializes the 6Step Lib; it is normally added in the first lines of the main file, after the MCU peripherals configuration. At this point the motor control algorithm is configured and ready to start. No input parameters are required.

`MC_SixStep_RESET` resets all the main variables with the default values used for the 6-step algorithm. It is called inside the `MC_StopMotor()` command if, for instance, a fault occurs or in case of manual stop of the control. No input parameters are required.

`MC_StartMotor()` is the main command used to start the 6-step algorithm: e.g., the PWM timer, the ADC conversions, the low frequency timer etc. No input parameters are required.

`MC_StopMotor()` is the main command used to stop the 6-step algorithm: e.g., for instance, the PWM timer, the ADC conversions, the low frequency timer etc. When the control has been disabled, the `MC_SixStep_RESET` is called. No input parameters are required.

`MC_Set_Speed(new_speed)` is the main command used to set a new speed value for the motor during the closed loop stage. The input variable is defined as 16-bit unsigned and is expressed in RPM units.

`MC_EXT_button_SixStep()` is called by `HAL_GPIO_EXTI_Callback()`, associated with an external push button. It sends a start or a stop motor command depending on the previous status of the control.

Revision history

Table 15. **Document revision history**

Date	Revision	Changes
20-Oct-2017	1	Initial release.
18-Jun-2019	2	Throughout document: added information regarding the STSW-SPIN3204 firmware package.

Contents

List of tables

List of figures