UM0708
User manual

STM8Sxxx
three-phase BLDC motor control software library V1.0

Introduction

This user manual describes the brushless direct current motor (BLDC) scalar software library, a scalar control firmware library for 3-phase permanent-magnet (PM) motors developed for the STM8Sxxx microcontrollers.

ST STM8Sxxx 8-bit microcontrollers come with a set of peripherals that makes them suitable for performing both PM and AC induction motor scalar control.

The present document describes the STM8Sxxx software library developed to control trapezoidal driven PM motors in both open loop and speed control mode. These motors may be equipped with three Hall sensors or may be sensorless. The control of an AC induction motor equipped with tachogenerator is described in the UM0712 user manual.

The BLDC motor software library is made of several C modules, and is fitted out with STVD workspace. It is used to quickly evaluate both the MCU and the available tools. In addition, when used together with the STM8/128-MCKIT motor control starter kit and a PM motor, a motor can be made to run in a very short time. The BLDC library also eliminates the need for time-consuming development of low level drive and speed regulation algorithms by providing ready-to-use functions that let the user concentrate on the application layer. Moreover, it is possible to get rid of any speed sensor thanks to the sensorless algorithm for rotor position reconstruction based on BEMF detection.

A prerequisite for using this library is basic knowledge of C programming, PM motor drives and power inverter hardware. In-depth know-how of STM8Sxxx functions is only required for customizing existing modules and for adding new ones for a complete application development.

Contents

List of tables

List of figures

1 Features

1.1 Performance line STM8S features

- Core
  - Advanced STM8 core with Harvard architecture and 3-stage pipeline
  - f_{CPU} up to 24 MHz setting, 0 wait state at $f_{CPU} \leq 16$ MHz.
  - Extended instruction set
  - Maximum 20 MIPS performance at f_{CPU} = 24 MHz
- Memories
  - Program memory: up to 128 Kbytes Flash; with 20 year data retention at 55 °C after 10 kcycles
  - Data memory: up to 2 Kbytes true data EEPROM; with 300 kcycle endurance
  - RAM: up to 6 Kbytes
- Clock, reset and supply management
  - 2.95 to 5.5 V operating voltage
  - Flexible clock control, 4 master clock sources:
    Low power crystal resonator oscillator
    External clock input
    Internal user-trimmable 16 MHz RC
    Internal low power 128 kHz RC
  - Clock security system with clock monitor
  - Power management:
    Low power modes (Wait, Active-halt, Halt)
    Individual peripheral clock switch-off
  - Permanently active, low consumption power-on and power-down reset
- Interrupt management
  - Nested interrupt controller with 32 interrupts
  - Up to 37 external interrupts on 6 vectors
- Timers
  - 2x 16-bit general purpose timers, with 2+3 capture/compare channels (input capture, output compare, or PWM)
  - Advanced 16-bit control timer
    4 capture/compare channels (input capture, output compare, PWM (edge or center-aligned mode)), single pulse output mode
    3 complementary outputs with adjustable dead-time insertion
    Hardware fault protection (Break input)
    Flexible synchronization
  - 8-bit basic timer with 8-bit prescaler
  - Auto wake-up timer
  - 2 watchdog timers: Window watchdog and independent watchdog

- **Communications interfaces**
  - High-speed 1 Mbit/s active CAN 2.0B interface
  - UART with clock output for synchronous operation, LIN master mode
  - LIN 2.1 compliant UART, master/slave mode, automatic resynchronization
  - SPI interface up to 10 Mbit/s
  - I^2C interface up to 400 Kbit/s
- 10-bit analog to digital converter (ADC) with up to 16 multiplexed channels
- I/Os
  - Up to 68 I/Os on a 80-pin package including 18 high sink outputs
  - Highly robust I/O design, immune against current injection
- Development support
  - Embedded Single Wire Interface Module (SWIM) and Debug module (DM) for fast on-chip programming
  - Non intrusive debugging

1.2 Access line STM8S features

- Core
  - 16 MHz advanced STM8 core with Harvard architecture and 3-stage pipeline
  - Extended instruction set
- Memories
  - Program memory: up to 32 Kbytes Flash; with 20 year data retention at 55 °C after 10 kcycles
  - Data memory: up to 1 Kbytes true data EEPROM; with 300 kcycle endurance
  - RAM: up to 2 Kbytes
- Clock, reset and supply management
  - 3.0 to 5.5 V operating voltage
  - Flexible clock control, 4 master clock sources:
    Low power crystal resonator oscillator
    External clock input
    Internal user-trimmable 16 MHz RC
    Internal low power 128 kHz RC
  - Clock security system with clock monitor
  - Power management:
    Low power modes (Wait, Active-halt, Halt)
    Individual peripheral clock switch-off
  - Permanently active, low consumption power-on and power-down reset
- Interrupt management
  - Nested interrupt controller with 32 interrupts
  - Up to 37 external interrupts on 6 vectors
- Timers
  - 2x 16-bit general purpose timers, with 2+3 capture/compare channels (input capture, output compare, or PWM)
  - Advanced 16-bit control timer
    4 capture/compare channels (input capture, output compare, PWM (edge or center-aligned mode)), single pulse output mode
    3 complementary outputs with adjustable dead-time insertion
    Hardware fault protection (Break input)
    Flexible synchronization
  - 8-bit basic timer with 8-bit prescaler
  - Auto wake-up timer
  - 2 watchdog timers: Window watchdog and independent watchdog
- Communications interfaces
  - UART with clock output for synchronous operation, Smartcard, IrDA and LIN mode
  - LIN 2.1 compliant UART, master/slave mode, automatic resynchronization
  - SPI interface up to 8 Mbit/s
  - I^2C interface up to 400 Kbit/s
- Analog to digital converter (ADC)

  - 10-bit, ±1 LSB ADC with up to 10 multiplexed channels, scan mode and analog watchdog
- I/Os
  - Up to 38 I/Os on a 48-pin package including 9 high sink outputs
  - Highly robust I/O design, immune against current injection
- Development support
  - Embedded Single Wire Interface Module (SWIM) for fast on-chip programming
  - Non intrusive debugging

1.3 BLDC software library V1.0 features (CPU running at 16 MHz)

The BLDC software library source code is available free of charge if it is used in an end-application based on ST products.

The main library features are the following:

- Supported BLDC modes (trapezoidal 6 step method):
  - Sensorless mode: back EMF voltage on the non-energized phase is monitored and used to trigger the commutation events
  - Sensor mode: Hall sensors trigger the commutation events.
  - Voltage mode: PWM duty cycle is set directly via 16-bit PWM generator.
  - Current mode: internal current loop and external voltage reference are used jointly to maximize the current in motor windings. The PWM duty cycle is automatically managed by the current feedback loop output.
  - Open loop operation
  - Closed loop operation: Proportional integral (PI) regulator, 1 to 255 ms sampling time.
  - DC bus voltage measurement
  - DC bus brake resistor management
  - Heatsink temperature measurement
  - Fault handling including overcurrent (shunt resistor network required), DC bus overvoltage/undervoltage, heatsink overtemperature.
  - User interface with LCD and joystick
  - Two-channel virtual DAC functionality for real-time tracing of software variables
  - Firmware compatibility with STM8S motor control builder GUI (see *Section A.3*)

- Required ROM/RAM: *Table 1* gives the ROM and RAM requirements. These values include non-motor control related code implemented for demonstration purposes, such as ADC management, or software time bases. They serve as a rough guide since the code size produced can be smaller or larger depending on the chosen memory model.

Table 1. ROM and RAM requirements

Configuration	ROM (Kbytes)	RAM (bytes)
Sensorless open loop	7.08	310
Sensorless closed loop	7.62	324
Sensor open loop	6.92	314
Sensor closed loop	7.46	328

1.4 Development tools

The BLDC software library has been fully validated using the STM8/128-MCKIT motor control starter kit. This kit also includes a Raisonance R-Link hardware debugger which makes it an ideal solution to start a project and evaluate with the BLDC software library.

As a consequence, it is recommended to acquire the STM8/128-MCKIT to quickly implement and evaluate the BLDC library.

1.4.1 Toolchains

The BLDC library has been compiled using COSMIC C-toolchains, running under ST Visual development release 4.1.2 (STVD). Free IDE and demonstration versions of third party toolchains can be found at http://www.st.com/mcu under Support/Downloads.

A complete software package consists of:

- An integrated development environment (IDE) interface: STVD (free download available on internet).
- A third party C-compiler: Cosmic (a free 16K size-limited evaluation versions can be obtained upon request. This version is sufficient to compile all standalone firmware configurations.

  The choice of the C toolchain is left to the appreciation of the user. Only COSMIC compilers are fully supported, and the dedicated workspace compatible with STVD can be directly opened in the root of the library installation folder (**STM8_STVD_COSMIC.stw**, **STM8_STVD_COSMIC_BLDC.stp**).

In addition, the STM8S motor control builder GUI allows to customize these libraries according to your application (see *Section A.3*). This makes the first implementation of this library significantly easier (see *Section 5: Designing an application using the BLDC software library*).

1.4.2 Programming tools

In order to program an MCU with the generated .S19 file, you should also install the ST Visual Programmer software (STVP), and use a SWIM programming interface (Raisonance RLink). The STVP tool provides an easy way to erase, program and verify the code programmed in the MCU. Go to http://www.st.com for information on STVP and RLink.

1.5 Reference documents

- Application note AN1129 – PWM management for BLDC motor drives using the ST72141
- Reference manual RM0016 – STM8S microcontroller family
- Datasheet – STM8S103xxx Access line, 16 MHz STM8S 8-bit MCU, up to 8 Kbytes Flash, integrated EEPROM,10-bit ADC, timers, UART, SPI, I²C
- Datasheet – STM8S105xx Access line, 16 MHz STM8S 8-bit MCU, up to 32 Kbytes Flash, integrated EEPROM,10-bit ADC, timers, UART, SPI, I²C
- Datasheet – STM8S20xxx - Performance line, 24 MHz STM8S 8-bit MCU, up to 128 Kbytes Flash, integrated EEPROM,10-bit ADC, timers, 2 UARTs, SPI, I²C, CAN
- Datasheet – STM8S903K3 - 16 MHz STM8S 8-bit MCU, up to 8 Kbytes Flash, 1 Kbyte RAM, 640 bytes EEPROM,10-bit ADC, 2 timers, UART, SPI, I²C
- User manual UM0379 – MB459 motor control evaluation board
- User manual UM0712 – STM8S three-phase ACIM motor software library V1.0
- PM0044 – STM8 CPU programming manual
- User manual UM0144 – ST Assembler-Linker
- User manual UM0036 – ST Visual Develop (STVD)
- User manual UM0482 – STM8/128-EVAL evaluation board
- UM0709 – STM8S-MCKIT motor control starter kit

2 Introduction to STM8S BLDC control

2.1 Introduction to BLDC theory

A brushless three phase motor is composed by a fixed part made of a set of three windings called stator, and a mobile part containing an internal permanent magnet called rotor.

In BLDC motor control, the electrical cycle is subdivided 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 (see *Figure 1*).

A BLDC motor has a trapezoidal BEMF (Back electromagnetic force) induced into the motor phase windings. The BLDC drive is also called trapezoidal control because of the shape of the phase current. The maximum performance in terms of efficiency and minimum torque ripple is achieved if the motor is intrinsically built as BLDC. *Figure 2* shows the BEMF signal for the BLDC Shinano motor included in the STM8/128-MCKIT.

Figure 1. BLDC motor control sequence (first three steps)

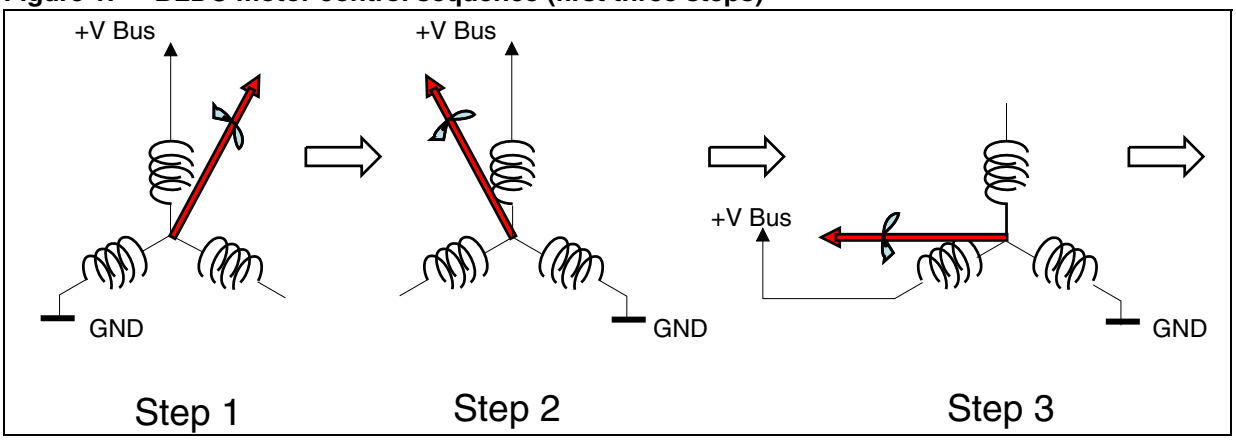
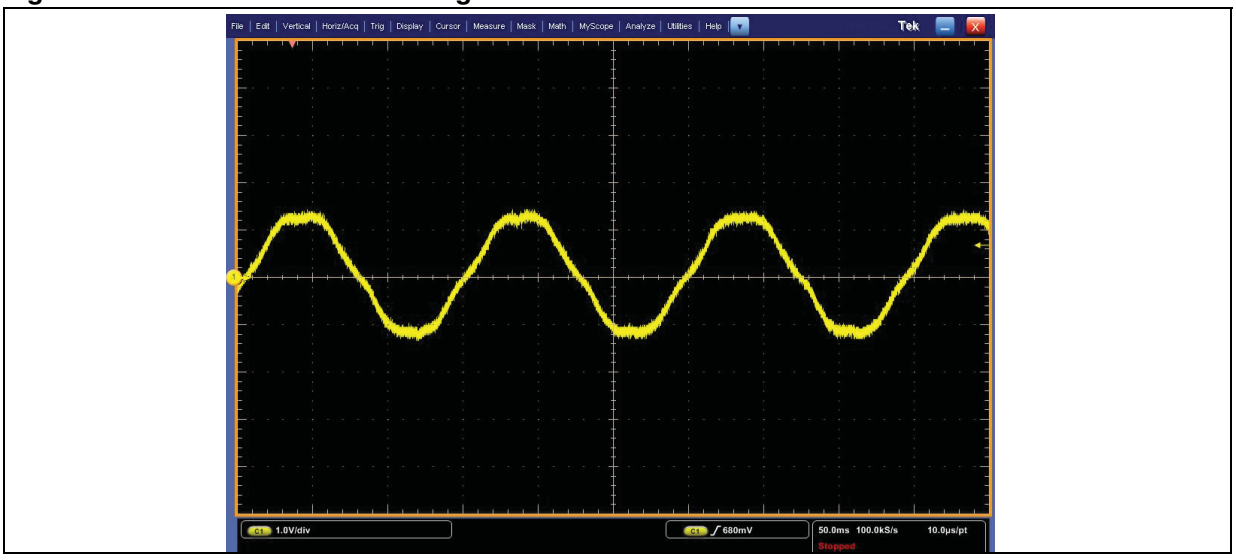


Figure 2. Shinano motor BEMF signal



2.1.1 Rotor synchronization

The BLDC drive is a synchronous control drive. This means that the maximum of the efficiency is achieved if the commutation between two consecutive steps is performed only when the rotor is in the right spatial position. This corresponds to the position where the BEMF signal and the phase current are synchronized.

Two methods can be implemented to perform the rotor synchronization:

- **Sensored drive**

  The first method uses position sensors, usually Hall sensors, to measure the rotor position.

- **Sensorless drive**

  The other method is based on the BEMF. 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 motor neutral point (or star point) is used to generate the commutations between two consecutive steps in order to achieve the rotor synchronization.

2.1.2 Rotor speed regulation

In addition to the synchronization with the rotor, the BLDC drive can control the rotor speed to compensate for eventual load variations. In this case the drive it is called “**closed loop drive**”.

If the BLDC synchronous drive does not perform rotor speed control, the drive is named “**open loop drive**”. In this case the final rotor speed is affected by load variations.

2.1.3 Bus voltage modulation

Normally the BLDC drive is performed by applying a PWM signal to modulate the bus voltage. This is implemented by using one of the following control method:

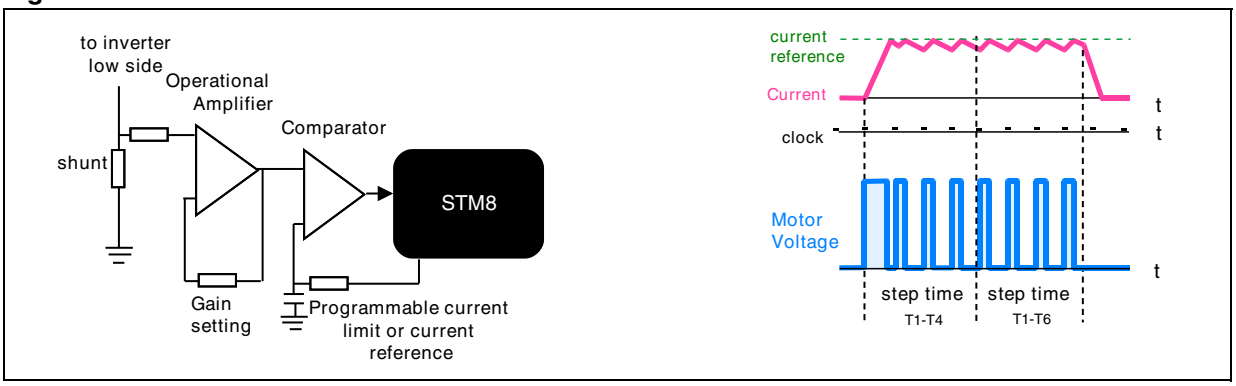
- **Current mode**

  The control variable is the phase current. Current regulation is achieved by setting the current level through a PWM signal and by using an external RC filter to set the comparator threshold. A dedicated microcontroller input driven by the output of the comparator is used to switch off the PWM signal used for current regulation. The turn-on is synchronized with then PWM period (see *Figure 3*).

- **Voltage mode**

  The control variable is the phase voltage. Voltage regulation is achieved by fixing the duty cycle of the applied PWM signal. Current limitation level is set either by an external resistor voltage divider or by a PWM signal filtered by the RC circuit.

Figure 3. Current mode



2.2 Rotor speed measurement

When the motor is running, 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.

The motor speed can consequently be calculated by measuring the frequency of the BEMF signals as shown in Equation 1:

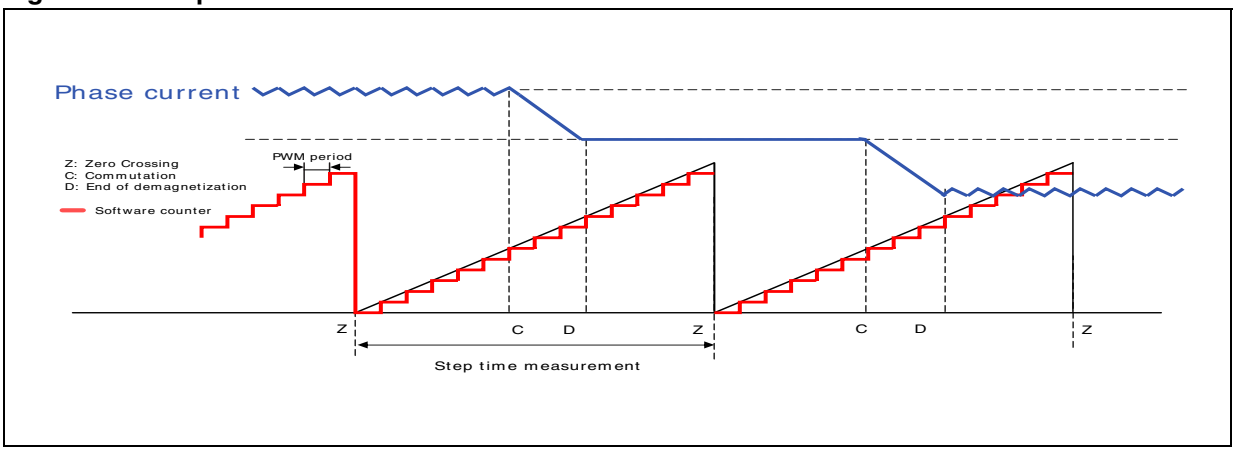
Equation 1

$$Moto(rpm) = \frac{BEMFfreq \times 60}{PolePairs}$$

The measurement of the rotor speed is performed by counting the time delay between two consecutive zero crossings of the BEMF signal. This computation is also called “step time measurement”. It is implemented in the BLDC firmware by means of a software counter managed by the timer update interrupt service routine (TIM1 UPD ISR). The TIM1 UPD ISR routine is executed during each PWM period.

The precision of the step time measurement is always proportional to the PWM period since zero crossings are sampled only once during each PWM cycle, with the resolution of the closest cycle.

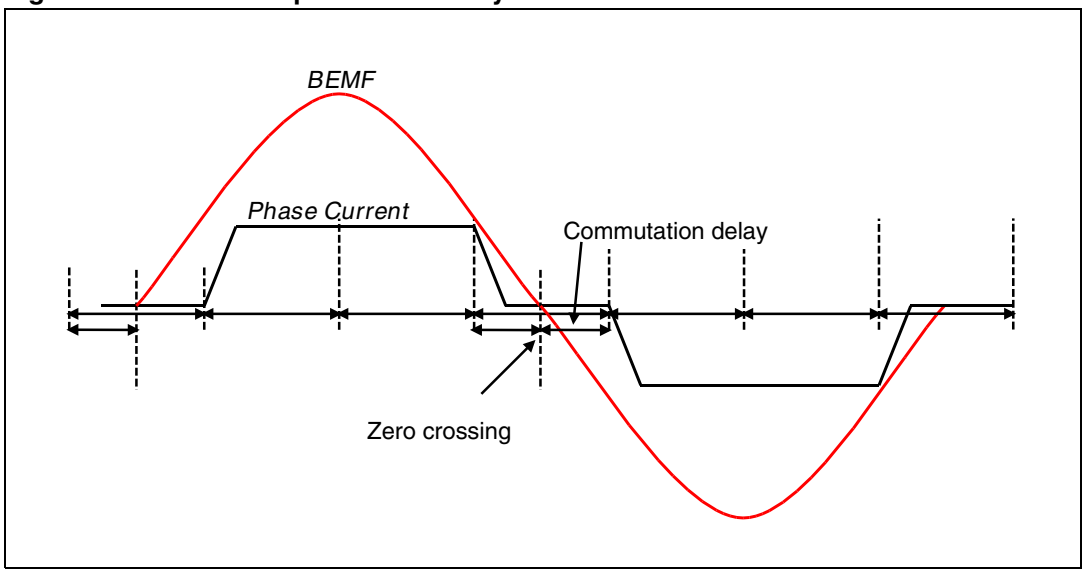
Figure 4. Step time measurement sensorless drive



2.3 Commutation delay and demagnetization time

Since the BLDC drive is a synchronous drive, 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 5*).

Figure 5. BEMF and phase current synchronization



The commutation delay is the delay equal to a half step time, between the zero crossing and the step commutation.

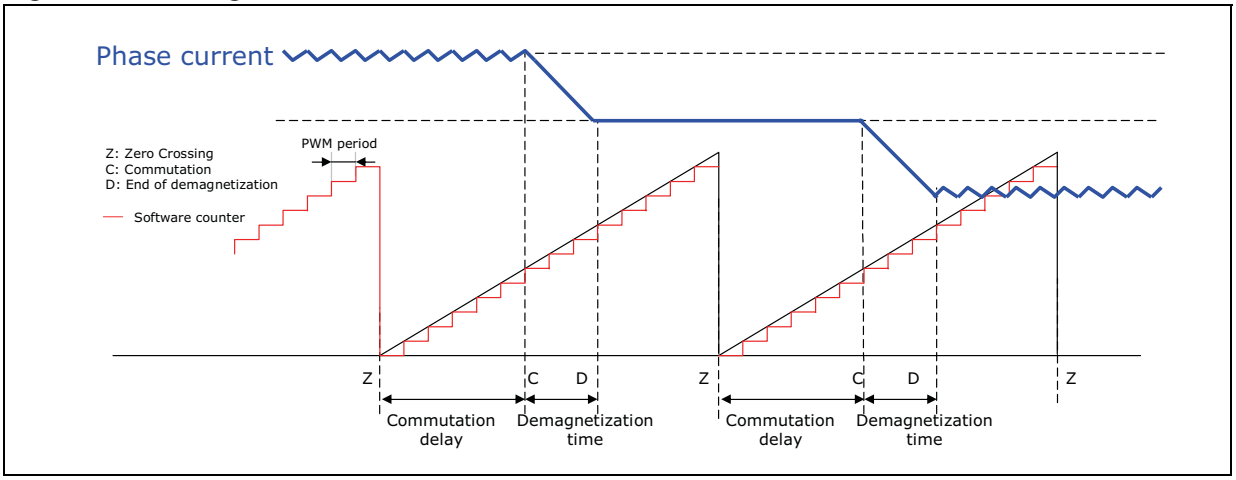
The BEMF signal can be measured only after the current of the floating phase has reached zero. Otherwise the free wheeling diode would clamp the floating terminal voltage or the bus voltage to ground depending on the current direction.

As a consequence, a delay time, called demagnetization time, must be respected after the commutation, to wait for the floating phase current to reach zero. The method used to manage the demagnetization time is usually called “software demagnetization”. It consists

in adding a delay between the commutation instant and the end of the demagnetization (start of next BEMF zero crossing detection).

The demagnetization delay is expressed as a percentage of the step time (see *Section 3.6.2*). It is computed using the mean value of two successive step time measurements.

Figure 6. Demagnetization time



2.4 Detection of BEMF zero crossings

Zero crossing points must be identified to achieve sensorless BLDC drive. Zero crossing points indicate the instant when the BEMF signal becomes equal to the motor neutral voltage.

During the on-time (T-on) of the PWM signal applied to the high side switch, the motor neutral voltage becomes equal to the bus voltage/2 (see *Section 2.4.1*).

During the off-time (T-off) of the PWM signal applied to the high side switch, the motor neutral voltage becomes 0 V (see *Section 2.4.2*).

The zero crossing can consequently be detected both during the on- and the off-time of the applied PWM signal. Refer to *Section 2.4.1* and *Section 2.4.2* for a detailed description of sampling methods during PWM signal on- and off-time.

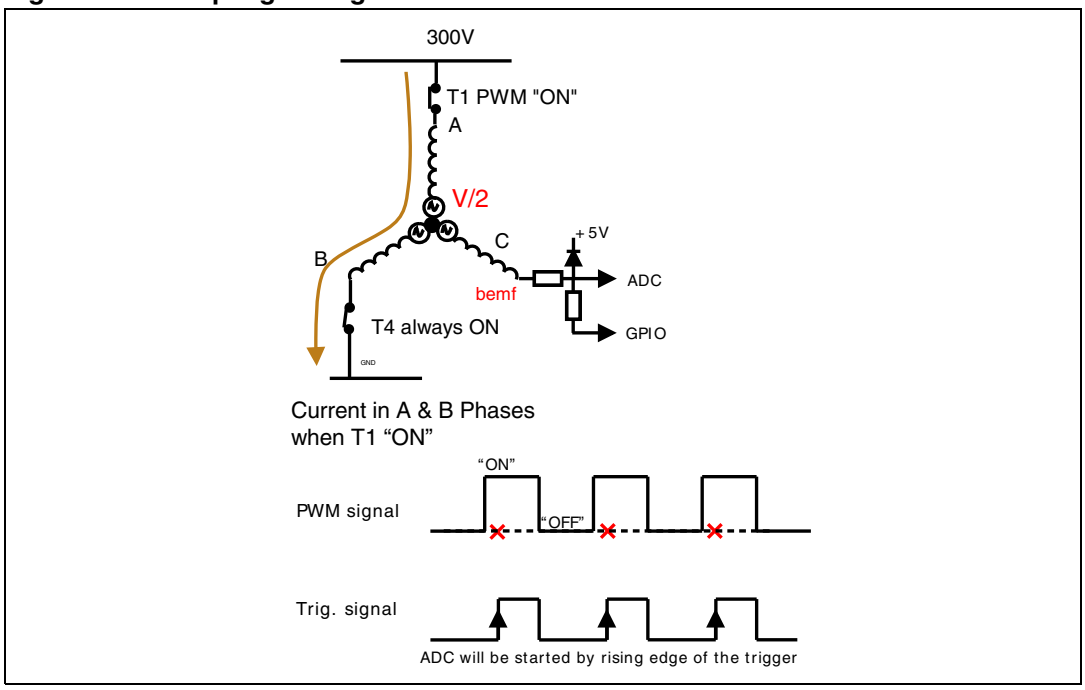
2.4.1 Sampling during PWM signal on-time

It is recommended to perform the sampling during the T-on when 100% of the duty cycle applied is required.

During T-on of the PWM signal applied to the high side switch, the motor neutral point is equal to the bus voltage/2. If this value is above the maximum voltage allowed by the STM8S ADC converter, it is necessary to divide the phase voltage before feeding the signal into the ADC, and to convert the bus voltage to reconstruct the neutral point. This value is used as a threshold for the zero crossing (see *Figure 7*).

Detection of the zero crossing is performed by comparing the BEMF converted scaled value with the converted scaled value of the bus voltage (neutral point reconstruction).

Figure 7. Sampling during the on-time



2.4.2 Sampling during PWM signal off-time

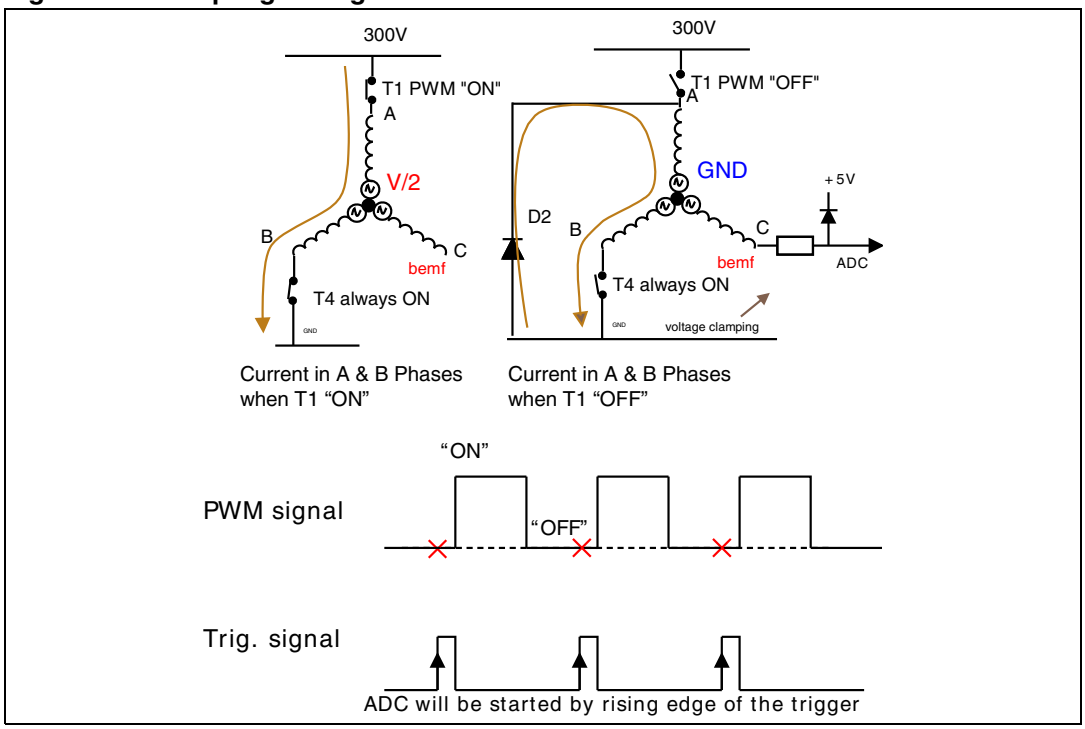
During T-off of the PWM signal applied to the high side switch, the motor neutral point becomes equal to ground so that the BEMF value can be directly converted by using the ADC (ST patented method).

The sampling point inside the PWM period is set by using TIM1 channel 4 to trigger the ADC conversion.

The detection of the zero crossing point is performed by comparing the converted value of BEMF to a fixed threshold. The default value of this threshold is 0.2 V.

Figure 8 gives a description of the sampling during T-off sequence.

Figure 8. Sampling during off-time

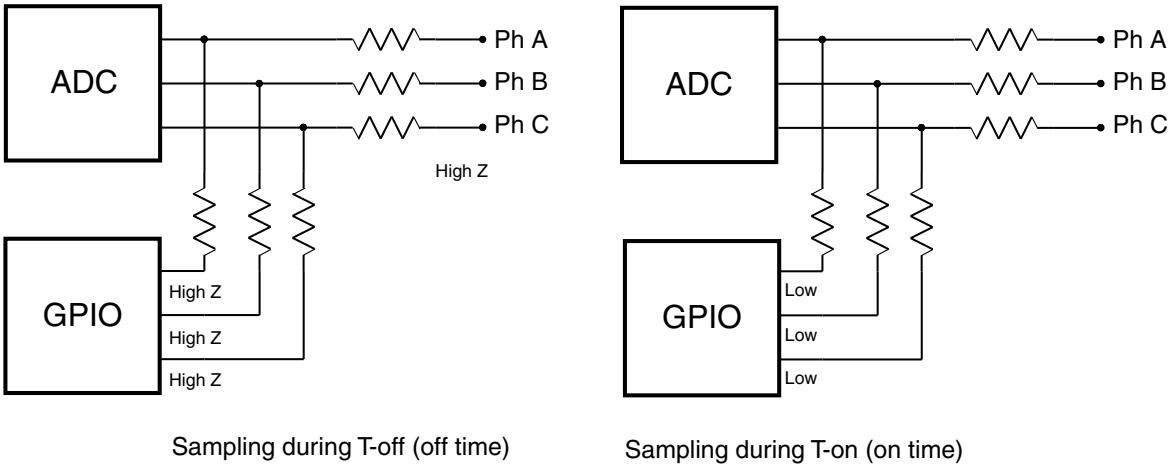


2.4.3 Dynamic BEMF sampling method

If the value of the duty cycle to be applied is known in advance (for instance when no current regulation or limitation is required), the sampling method to be applied (sampling during T-off or T-on) can be set on the fly according to the duty cycle value. This is achieved by using three voltage dividers driven by GPIO pins as shown in *Figure 9*.

This method allows both to minimize BEMF partitioning (sampling during off-time) for low duty cycle applied (low speed), and to get 100% of duty cycle (high speed).

Figure 9. Dynamic BEMF sampling method implementation



2.5 Fast demagnetization

The reverse voltage on the winding can be increased to speed up the floating phase demagnetization. This is achieved by applying the PWM signal to the appropriate high or low side switch transistor. If the PWM signal is not applied to the right transistor, the free wheeling diode applies a voltage in the same direction as the current to be demagnetized, thus slowing down the demagnetization process. Refer to application note AN1129 referenced in *Section 1.5: Reference documents*.

The following description assumes that the three phase bridge is similar to the one shown in *Figure 10*, and that the difference between the forward voltage drop of the free wheeling diode and the conduction drain source voltage drop of the power switches is negligible.

Figure 10. Three phases bridge and stator windings

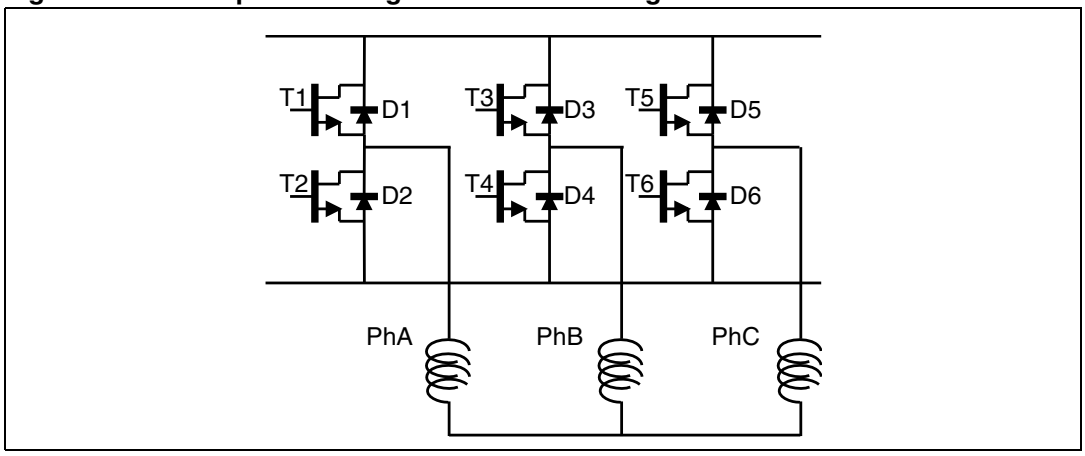


Figure 11 shows the transition between step 1 (T1-T4 closed) and step 2 (T1-T6) closed. In this case the current that flows out of phase B has to be demagnetized.

Phase B terminal will be clamped to +HV by the free wheeling diode D3. The neutral point is HV/2 during T-on.

If the PWM signal is applied to the high side switch T1 during the T-off, the neutral point will be clamped to GND by the D2 free wheeling diode. As a consequence, the phase B winding will be demagnetized with an equivalent PWM voltage between HV and HV/2.

If the PWM is applied to the low side switch T6 during T-off, the neutral point will be clamped to HV by the D5 free wheeling diode. The phase B windings will consequently be demagnetized with an equivalent PWM voltage between HV/2 and GND. The reverse voltage applied to the windings is lower than in the first case.

To speed up the demagnetization of B windings between step 1 and step 2, the PWM control signal should consequently be applied to the high side switch T1.

Figure 11. Demagnetization between step 1 and step 2

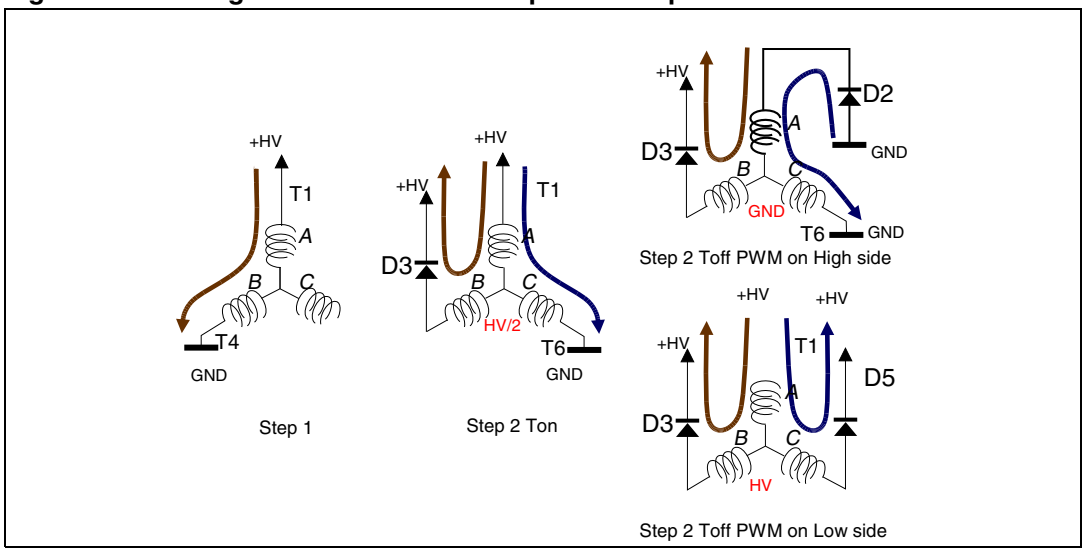


Figure 12 shows the transition between step 2 (T1-T6 closed) and step 3 (T3-T6) closed. In this case, the current that flows through phase A has to be demagnetized.

Phase A terminal will be clamped to GND by the free wheeling diode D2. The neutral point is HV/2 during T-on.

If the PWM is applied to the high side switch T3 during T-off, the neutral point will be clamped to GND by the D4 free wheeling diode. As a consequence, phase A winding will be demagnetized with an equivalent PWM voltage between HV/2 and GND.

If the PWM is applied to the low side switch T6 during T-off, the neutral point will be clamped to HV by the D5 free wheeling diode. Phase A windings will consequently be demagnetized with an equivalent PWM voltage between HV and HV/2. The reverse voltage applied to the windings is higher than in the first case.

The PWM control signal should consequently be applied to the low side switch T6 to speed up the demagnetization of A windings between step 2 and step 3.

In conclusion, to accelerate the demagnetization the PWM signal should be applied to the low side switch during the step when the demagnetizing current is flowing from the bridge to the motor phases. Refer to *Table 2* for a description of the firmware implementation.

Figure 12. Demagnetization between step 2 and step 3

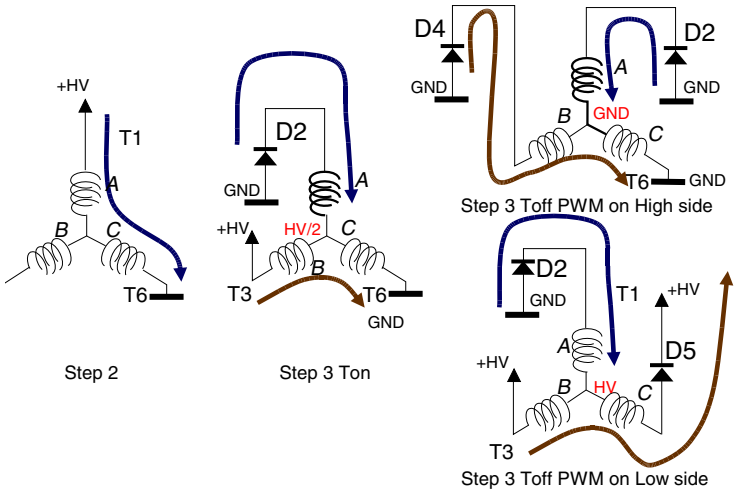


Table 2. Fast demagnetization table

Step	Switches (HS+LS)	Demagnetizing current	PWM applied for fast demagnetization
Step 1	T1+T4	Current C (from bridge to motor)	Low side (T4)
Step 2	T1+T6	Current B (from motor to bridge)	High side (T1)
Step 3	T3+T6	Current A (from bridge to motor)	Low side (T6)
Step 4	T3+T2	Current C (from motor to bridge)	High side (T3)
Step 5	T5+T2	Current B (from bridge to motor)	Low side (T2)
Step 6	T5+T4	Current A (from motor to bridge)	High side (T5)

2.6 Setting the delay coefficient accordingly a specified curve

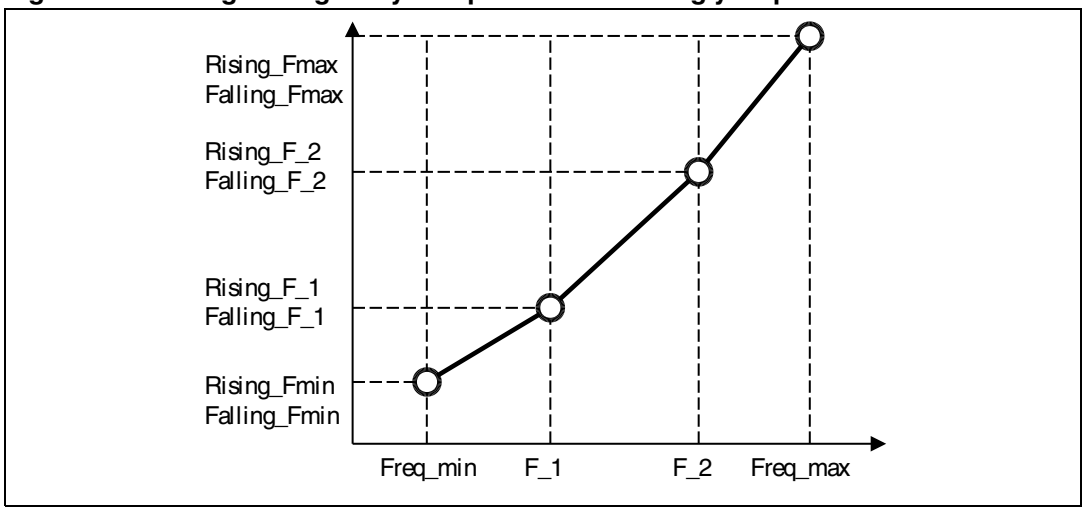
It is important to select the most adequate delay coefficients to run a BLDC motor in standalone closed loop mode and achieve a given target speed.

For each target speed, these values should be recorded in a table which will be used by the STM8S firmware. It is recommended to collect data for four speeds: the minimum and maximum speeds specified, plus two intermediate speeds. The STM8S firmware then makes a linear extrapolation of delay coefficients between the four specified speeds to ensure smooth operation.

Once the data are collected, edit the `MC_BLDC_drive.h` file and fill in the fields dedicated to the rising/falling delay coefficients calculation (see *Section 5.1.3: BLDC drive control parameters (MC_BLDC_Drive_Param.h)*).

Once the motor runs, rising/falling delay coefficients are computed following a linear curve between Freq_min and F_1, F_1 and F_2, F_2 and Freq_max (see *Figure 13*).

Figure 13. Rising/Falling delay computation accordingly a specified curve



1. Freq_min, F_1, F_2, and Freq_max are electrical frequencies, with 0.1 Hz resolution. For example F_1 = 1234 means F_1 = 123.4 Hz).
2. Rising and Falling curve can be different.

2.7 Startup strategy in sensorless mode

To perform sensorless control, the BEMF signal inducted by the rotor field in the stator windings must reach a detectable amplitude. Since the amplitude of the BEMF signal is proportional to the rotor speed, the motor cannot be synchronized using the BEMF detection strategy if the rotor speed is lower than a specific threshold.

It is important to underline that the detection is performed on the BEMF zero crossing and not on the BEMF peak value. However, to achieve a good zero crossing detection precision, the BEMF signal slope must be sufficiently steep during the zero crossing.

The BEMF level increases proportionally with the speed. As a result, the BEMF signal slope becomes more and more steep in the proximity of the zero crossing, and the synchronous driving (using the BEMF detection) can be applicable. It is consequently important to implement an asynchronous startup strategy that brings the motor up to the required speed.

A classical BLDC motor startup sequence consists of 2 phases:

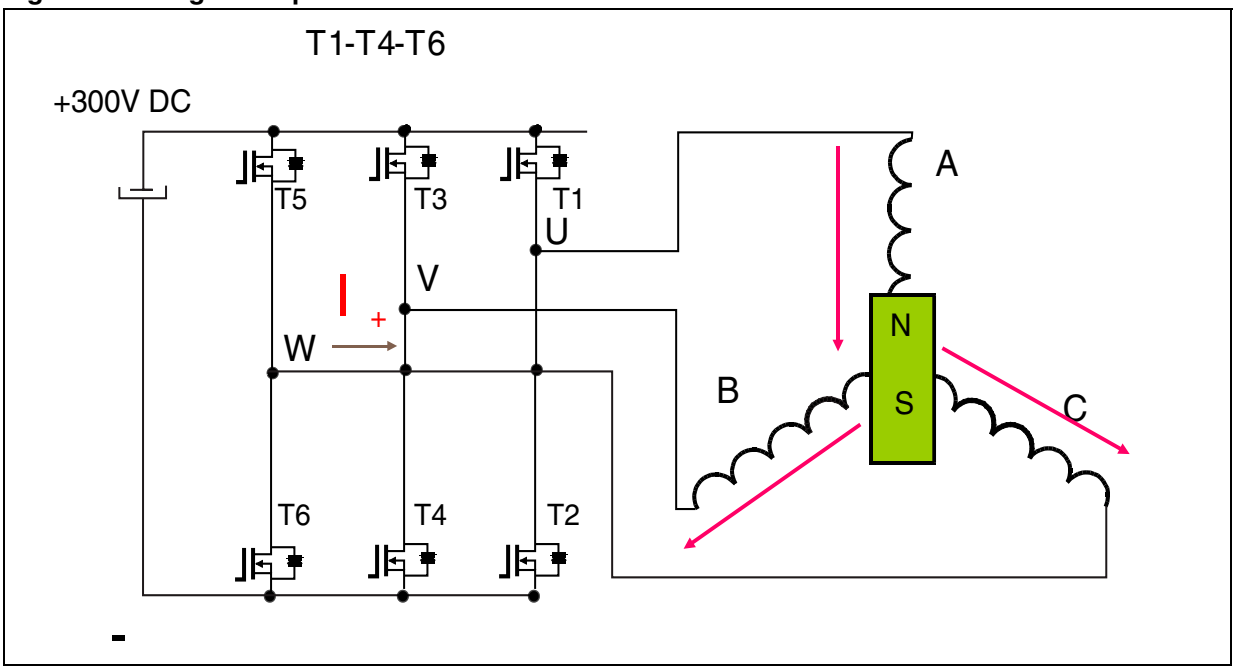
1. The Alignment phase
2. The ramp-up phase

2.7.1 Alignment phase

During the alignment phase, the rotor is put in a specified position before running the motor. To do this, the power switches must be set to a specific configuration corresponding to a predetermined position of the stator vector. This configuration is not an ordinary position but it is the intermediate position between two steps (for example between step 6 and step 1 if you want to start the next phase with step 1). Moreover this configuration has the particularity to energize the three windings to generate a more important flux (see *Figure 14*).

Usually the high side switch (T1) is controlled by a PWM signal with a duty cycle that increases following a ramp. The current limitation sets the maximum current that flows into the motor during the alignment phase.

Figure 14. Alignment phase



2.7.2 Ramp-up phase

The second phase is the ramp-up phase. It is performed to accelerate the motor up to the required speed that allows the BEMF zero crossing detection. To do this, an asynchronous sequence of steps must be applied to the motor. This sequence of steps generates a rotating stator field that will accelerate the rotor up to the target speed. The strength of the applied torque is usually controlled applying a fixed duty cycle or controlling the direct current that flows into the motor.

When the rotor speed is sufficient to detect the BEMF zero crossing, the synchronization mechanism is enabled. This mechanism it is also called “auto-commutation mode”.

Then according to the configuration of the speed loop regulation, one of the following strategy is applied:

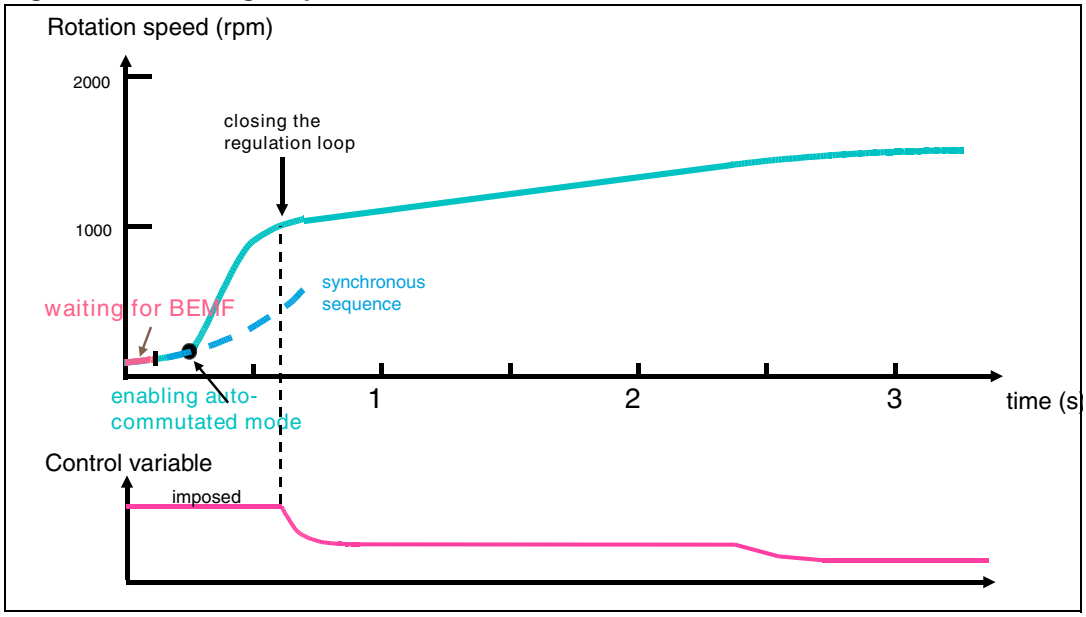
- In speed closed loop configuration, the speed regulation is not applied until the measured rotor speed reaches the required threshold. The control variable remains constant until this condition is met (see *Figure 15*). The startup output condition is validated only when the minimum rotor speed is measured.
- In speed open loop configuration, the startup output condition is validated when the drive is switched to auto-commutation mode.

If no BEMF zero crossing is detected until the end of the synchronous sequence the **Startup failed** fault message is displayed on the LCD (see *Section 3.6.8*).

Note: Before each motor startup, a bootstrap sequence ensures that the bootstrap capacitor used by the high side driver will be recharged. This operation is performed by closing the three

low side switches for the time required to charge the capacitors. In the BLDC firmware, a duration of 5 ms is implemented.

Figure 15. Starting sequence



2.8 Active brake

The BLDC firmware features the Active brake option. This function switches on the active brake of the motor by injecting a DC current into the motor phases.

This is performed by applying the same configuration as during the alignment phase (see *Figure 14*). The high side switch (T1) is controlled by a PWM signal with a fixed duty cycle. Current limitation is always performed to prevent the current flowing through the motor to exceed its maximum value.

The active brake is kept for a fixed time that can be configured through the firmware.

Caution: If the active brake is switched on while the motor is running, the extra energy generated by the motor is flowed into the bus voltage and may cause overvoltage. In this case it is recommended to use the active brake in conjunction with the dissipative brake function (see *Section 4.3.4*).

3 Running the demonstration program

The BLDC motor control software library includes a demonstration program which allows driving a Shinano motor in sensorless mode.

A user interface has been developed to display drive variables, and customize the application by changing parameters, and disabling/enabling options in real time.

The interface is composed of:

- A 2x15 character LCD screen
- A joystick (see *Table 3* for the list of joystick actions and conventions)
- A push button (Key button)

3.1 User interface structure

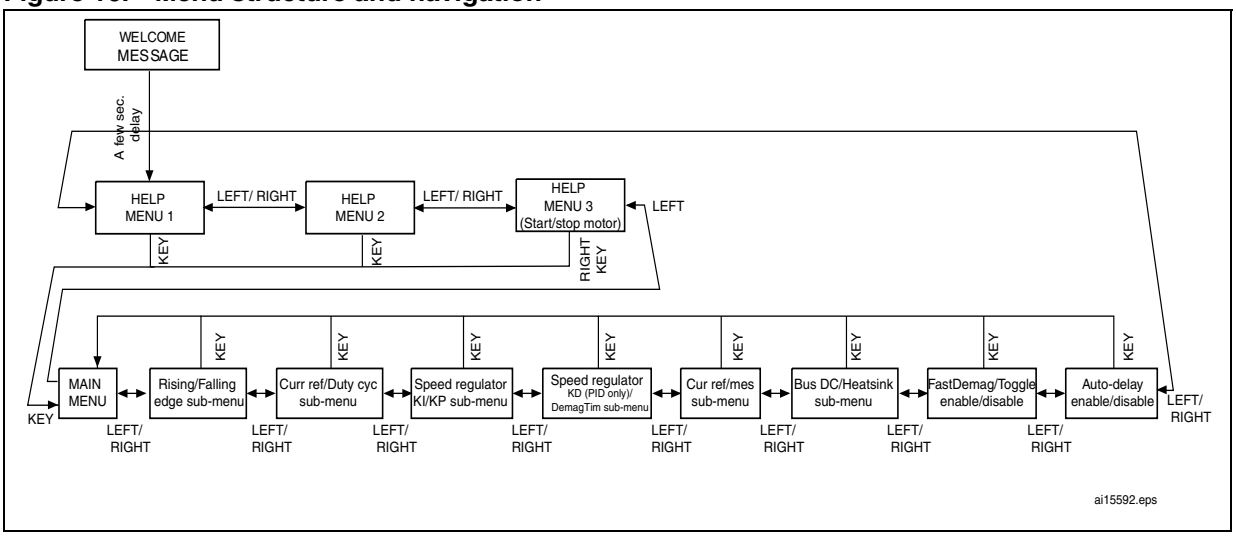
The demonstration program user interface is based on a circular navigation menu, with submenus, item selection and back capability.

Figure 16 shows the menu structure.

To navigate the help menus and sub-menus, perform the following actions as required:

- **RIGHT**: navigates to the next menu or sub-menu on the right.
- **LEFT**: navigates to the next menu or sub-menu on the left.

Figure 16. Menu structure and navigation



3.2 Document conventions

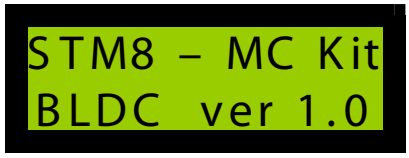
Table 3. Document conventions

Keyword	User action
UP	Joystick pressed up
DOWN	Joystick pressed down
LEFT	Joystick pressed to the left
RIGHT	Joystick pressed to the right
JOYSEL	Joystick pushed
KEY	Press the KEY push button

3.3 Welcome message

After the MB459B is powered on or reset, a welcome message is displayed on the LCD screen to inform the user about the release of the firmware code loaded on board. Refer to *Figure 17* for the BLDC drive welcome message.

Figure 17. BLDC drive welcome message

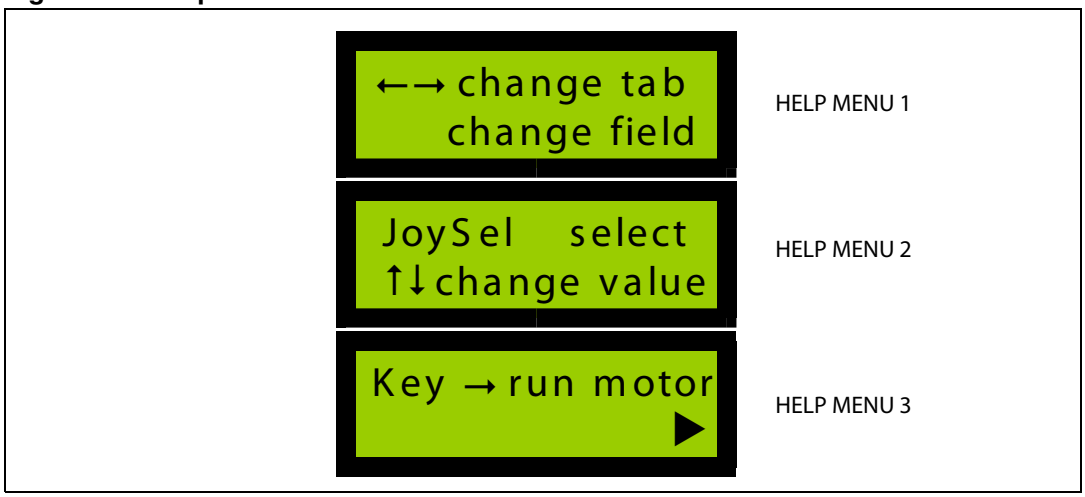


3.4 Help menus

After a few seconds, the LCD screen displays the first help menu (see *Figure 18*). The user can then navigate to the next help menu by pressing the joystick RIGHT, or go back to the previous help menu by pressing the joystick LEFT.

The KEY button can be pressed anytime to start and stop the motor. When the KEY button is pressed, the user interface is automatically switched to the main menu, regardless of which menu is selected.

Figure 18. Help menu

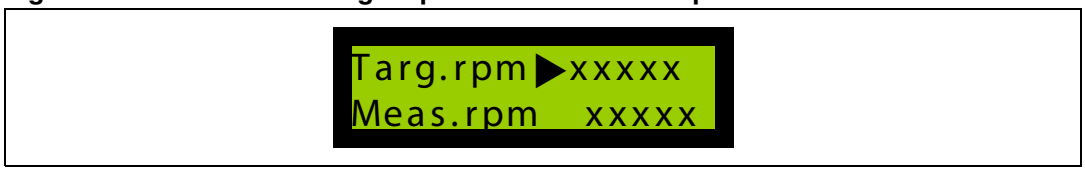


3.5 Main menu - target and measured rotor speed

To enter the main menu, press the joystick RIGHT one more time from the Help menu 3.

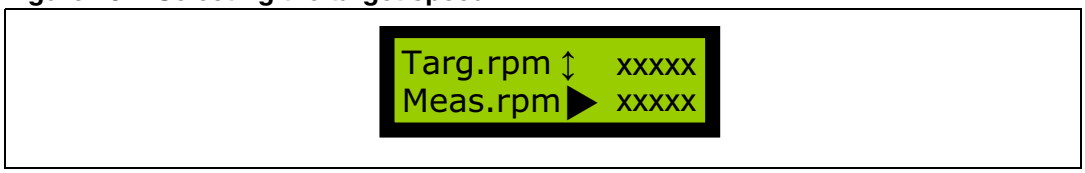
The main menu allows to set the target rotor speed and display the measured speed (see *Figure 19*). The main menu function is the same as the other sub-menus except that the system is automatically switched to the main menu when the motor is started or stopped.

Figure 19. Main menu - Target speed and measured speed



1. To set the target rotor speed, press JOYSEL when the **Targ.rpm** function is active (▶ displayed and blinks).
2. After pressing JOYSEL, the arrow changes in to a double arrow ↕ to indicate that the value can be changed (see *Figure 20*).
3. Press the joystick UP and DOWN to increase and decrease the value.

Figure 20. Selecting the target speed



When the motor is still, enter a negative target speed to run the motor in the opposite direction.

3.6 User interface sub-menus

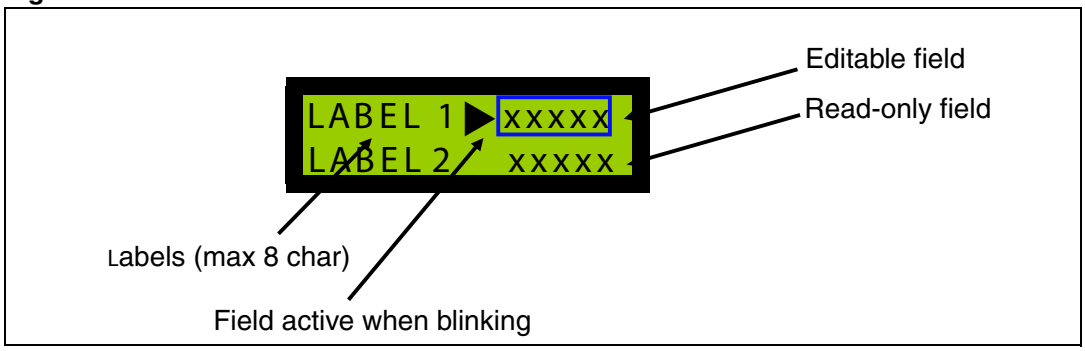
3.6.1 Sub-menu overview

Each sub-menu of the user interface is composed of two fields, which are in turn composed of one label, one value and the corresponding unit (see *Figure 21*).

The field can either be editable or read-only:

- An editable field can be selected and modified by the user when the cursor ▶ is displayed near the field.
- A read-only field is used to display a value. The user can neither select it nor modify it.

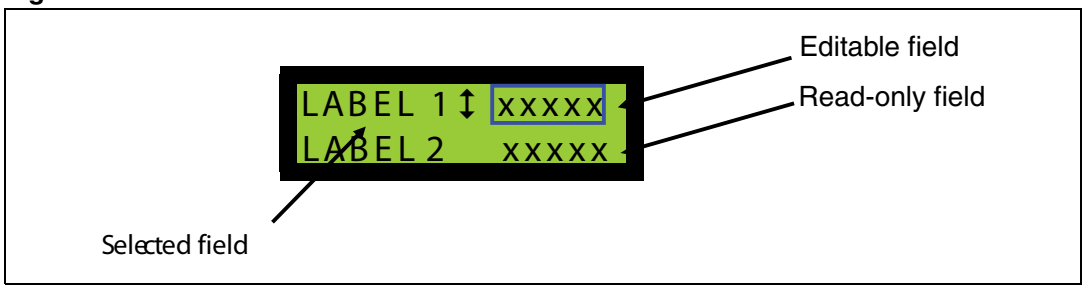
Figure 21. User interface sub-menu



To select and modify and editable field, the following steps are required:

1. Press the joystick UP or DOWN to navigate between the editable fields of the sub-menu. The cursor blinks when the field is active.
2. Then press JOYSEL to select the active field. The cursor changes to ↕.
3. Change the value by pressing the joystick UP or DOWN. (see *Figure 22*).
4. To exit from edit mode, press JOYSEL again or change sub-menu.

Figure 22. Field selected for edition



Press the joystick LEFT or RIGHT to navigate between sub-menus.

Each sub-menu is related to a specific issue of BLDC motor control. The following sections provide a detailed description of the sub-menus. Users with less experience in BLDC motor control are advised to jump to *Section 4: Getting started with the STM8S BLDC firmware*.

3.6.2 Changing rising and falling delays

To enter this sub-menu, press the joystick RIGHT from the main menu.

This sub-menu allows to set the falling and rising delay coefficients expressed in 1/255 of percentage. 255 correspond to 100% of the step time. If the auto-delay function is enabled, this sub-menu displays the current values of the delay coefficients applied.

Contrarily to the main menu in which only the **Targ.rpm** field is editable, both fields can be selected and modified.

Figure 23. Changing rising and falling delays

3.6.3 Setting the current reference and duty cycle

The current reference and the duty cycle are the control variables related to the current mode and voltage mode, respectively. Refer to *Section 2.1* for a detailed description of the control variables.

This sub-menu features two fields:

- The current reference

  The current reference is expressed in Ampere using a 2.1 fixed point format. This field is used:

  – To set the current reference in open loop current mode configuration

  – To display the applied current reference in closed loop current mode.

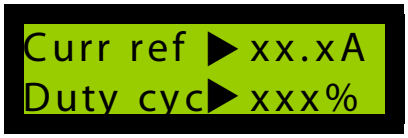
- The duty cycle

  This field is used:

  – To set the applied duty cycle in open loop voltage mode

  – To visualize the applied duty cycle in closed loop voltage mode.

Figure 24. Control variables



Note: These fields can be modified only if the related control method (current mode for current reference and voltage mode for duty cycle) is enabled in the firmware and the closed loop is not set. Modifications in current reference or duty cycle will be taken into account if the corresponding mode is not selected.

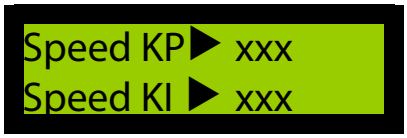
3.6.4 Configuring the speed regulator

When the firmware is configured in speed closed loop mode, the user can adjust the speed regulator parameters while the motor is running. The speed regulator implemented can be a proportional-integral (PI), or a proportional-integral-derivative (PID) regulator. The regulator

acts on the control variable minimizing the error between the target speed and the measured speed.

Figure 25 shows the speed regulator sub-menu. It allows to change the proportional term (Speed KP) and the integral term (Speed KI) of the speed controller.

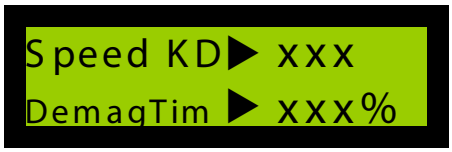
Figure 25. Configuring the speed regulator proportional and integral terms



The speed regulator type (PI or PID) can be selected using the following define statement included in the `MC_BLDC_Drive_Param.h` header file:

- `#define SPEED_PID_TYPE PI // (unit none)`
  Select a PI speed regulator
- `#define SPEED_PID_TYPE PID // (unit none)`
  Select a PID speed regulator. In this case, the derivative term is set through the Speed KD field of the sub-menu shown in *Figure 26*.

Figure 26. Configuring the speed regulator tuning derivative term: demagnetization time



Note: 1 Changing the Speed KD field has no effect if the speed controller is of PI type.

2 Changing the Speed KP, Speed KI, Speed KD has no effect if the firmware is configured in speed open loop.

3 The controller coefficients actually applied are the ratio between the values (ranging from 0 to 255) configured by the user through the sub-menus, and the division factors defined in the `MC_BLDC_Drive_Param.h` header file:

```
#define SPEED_KP_DIVISOR 128 // (unit none)
#define SPEED_KI_DIVISOR 512 // (unit none)
#define SPEED_KD_DIVISOR 16 // (unit none)
```

3.6.5 Changing the demagnetization time

To adjust the demagnetization time, use the sub-menu shown in *Figure 26*. It is recommended to set a demagnetization time high enough to allow the discharge of the current inside the floating phase. This parameter is a percentage of the step time (see *Section 2.3: Commutation delay and demagnetization time*).

3.6.6 Displaying the measured current, DC bus voltage and heatsink temperature

The firmware measures several power stage output characteristics (see *Section 4.3.2: Using the ADC*):

- The direct current that flows into the motor.
- The DC bus voltage level
- The heatsink temperature

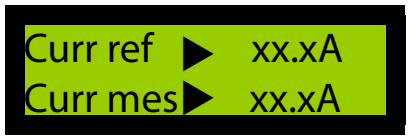
Direct current

The sub-menu shown in *Figure 27* displays the measured value of the direct current together with the current reference.

The **Curr ref** field assumes different meanings according to the configuration of the firmware:

- In closed loop current mode, it corresponds to the current reference computed by the speed regulator.
- In open loop current mode, this is the current reference set by the user in the sub-menu.
- In voltage mode, it is not used for the drive.

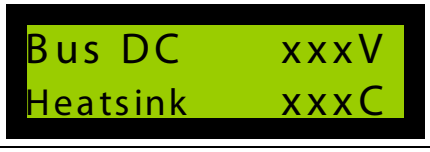
Figure 27. Current measurement



DC bus voltage level and heatsink temperature

The sub-menu shown in *Figure 28* displays the measured DC bus voltage level (**BUS DC)**, and the heatsink temperature (**Heatsink**). **BUS DC** is the rectified input voltage express in Volt, while **Heatsink** is the temperature expressed in °C and measured by the negative coefficient temperature (NTC) resistor placed on the power stage close to the heatsink of the power switches.

Figure 28. DC bus voltage and heat sink measurements



3.6.7 Additional options: fast demagnetization, toggle mode, and auto-delay

Additional options have been implemented in the firmware:

- Fast demagnetization
- Toggle mode
- Auto-delay

These options can be enabled or disabled in real-time through the user interface.

Fast demagnetization

The sub-menu shown in *Figure 29* can be used to enable/disable the fast demagnetization (**FastDemag**) option.

The following steps are required:

1. Press the joystick UP or DOWN to select the **FastDemag** field.
2. Then press JOYSEL to activate it
3. Push the joystick UP or DOWN to switch the option from enabled to disabled, and vice versa.

Refer to *Section 2.5* for more information concerning fast demagnetization.

Toggle mode

To enable/disable the toggle mode, use the same sub-menu and sequence as for fast demagnetization (see *Figure 29*).

When the toggle mode is enabled, the absolute value of the target speed is kept but with a complemented sign to allow the user to run the motor alternatively in clock wise and counter clock wise direction each time the motor is restarted.

Figure 29. Enabling/disabling fast demagnetization and toggle mode options

Auto-delay

The sub-menu shown in *Figure 30* can be used to enable or disable the auto-delay option. Refer to *Section 2.6* for more information regarding this option.

Figure 30. Auto-delay option

3.6.8 Fault messages

This section provides a description of the fault messages that can be displayed on the LCD screen when using the BLDC firmware together with the STM8/128-MCKIT.

There are seven different fault sources when using the BLDC firmware in conjunction with the STM8/128-MCKIT:

- Overcurrent

  A low level was detected on the PWM-peripheral-dedicated pin (BKIN). This means that either the hardware overtemperature protection or the hardware overcurrent protection has been triggered.

- Overtemperature

  An overtemperature was detected on the dedicated analog channel. The threshold (NTC_THRESHOLD_C) and the related hysteresis (NTC_HYSTERESIS_C) are specified in the `MC_PowerStage_Param.h` header file.

- Bus overvoltage

  An overvoltage was detected on the dedicated analog channel. The threshold (MAX_BUS_VOLTAGE) is specified in the `MC_PowerStage_Param.h` header file. This fault message is available only if the DISSIPATIVE_BRAKE define statement is commented (default status) in `MC_PowerStage_Param.h`.

  If the DISSIPATIVE_BRAKE define statement is uncommented, it is assumed that a resistor with a high power dissipation capability was connected in parallel to the bus capacitors through a switch. In this case the over voltage does not generate a fault event because the resistor is supposed to be able to dissipate the excess of voltage across the bus capacitors. For more detailed information on brake resistor management see also section *Section 4.3.4: Dissipative brake*.

- Bus undervoltage

  The DC bus voltage is below 18 V DC. This threshold is specified in the `MC_PowerStage_Param.h` header file by the MIN_BUS_VOLTAGE parameter.

- Startup failed

  This fault message displays only when SENSORLESS is uncommented. It signals that no startup output condition was detected during motor ramp-up. Refer also to *Section 2.7: Startup strategy in sensorless mode*.

- Speed feedback error

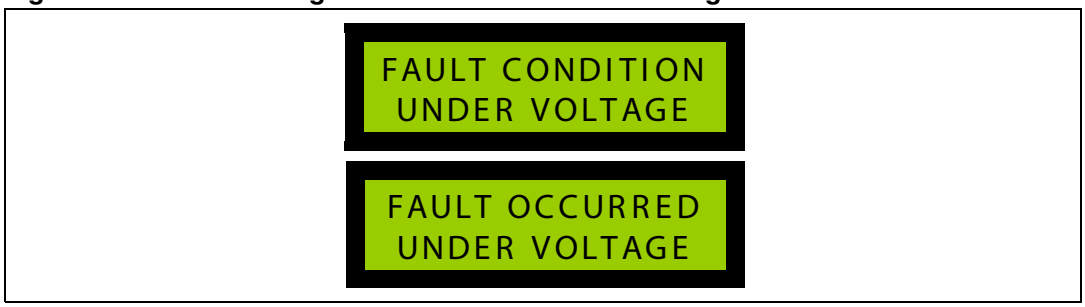
  An error on the speed/position feedback was detected.

- Motor running error

  This fault can occur if the firmware is configured in sensor mode. It will be reported if the user tries to start the motor when it is not still.

Figure 31 shows a typical error message.

Figure 31. Fault message shown in case of undervoltage



To indicate the fault which occurred to the user, the ‘FAULT OCCURRED’ message remains displayed even if the source of the fault has disappeared.

The ‘FAULT CONDITION’ message is displayed only when the fault condition is present.

If several fault events occur simultaneously, they are displayed in the LDC one after the other.

If all fault sources disappear, pressing the KEY button causes the main state machine to switch from the fault to the idle state.

4 Getting started with the STM8S BLDC firmware

4.1 Application state machine

The motor control firmware library was developed around the state machine showed in *Figure 32*. The state machine is implemented in `MC_StateMacchine.c` module. it is composed by the following states: Reset, Idle, Start init, Start, Run, Stop, Wait, Fault and Fault over.

4.1.1 Description of the states

Reset

The system is in Reset state once after the main reset. It is used to perform the main initialization.

Idle*

When the state machine goes into Idle state, the motor is stopped and the system waits for a startup to be executed.

Start init

The Start init state is executed at every restart of the motor. It is used for specific initialization.

Start*

In this state the motor ramps.

Run*

After the end of the startup phase, the motor is in normal Run state. The user can interact with the system, change parameters in real-time, or issue a stop request.

Stop

The motor is in Stop state each time the motor needs to be stopped.

Wait*

The system is in Wait state when the motor is stopped. It remains in this state till the required condition for a new restart is present.

Fault*

The system goes into Fault state when an error condition occurs. It remains in this state as long as the fault condition is present.

Fault over*

When the fault condition has disappeared, the system remains in Fault-over state to indicate which error occurred, and waits for a user action.

Note: The states marked with an asterisk are executed continuously until an event occurs (user action or fault condition).

4.1.2 Description of the state machine operation

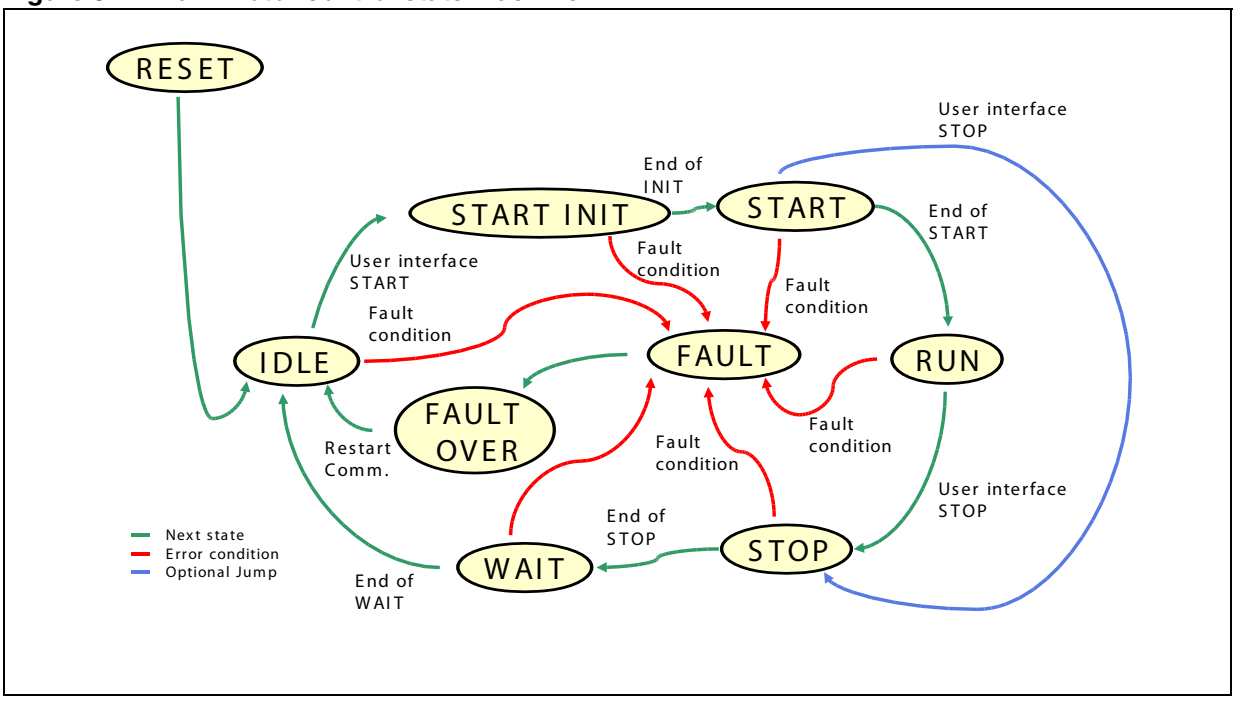
Each state corresponds to the execution of the related state machine function. The change of state is performed according to the value returned by that state machine function.

The returned value of a state machine functions can be one of the following:

- **State remain**: no change in the state is required by the state machine function.
- **Next state**: the natural flow of the state machine is followed (for example, Idle -> Init start -> Start -> Run). The natural flow is symbolized by green lines in *Figure 32*
- **Optional jump**: a path deviation caused by user action occurred (for example Start -> Stop). The optional jumps are symbolized by blue lines in *Figure 32*.
- **Error condition**: a fault condition occurred (for example Startup failed, overtemperature). The error conditions ar symbolized by red lines.

Each state machine function make calls to the related drive functions, to the user interface interaction functions, and to the error check functions. It executes the action on the basis of the outputs of these functions.

Figure 32. Main motor control state machine

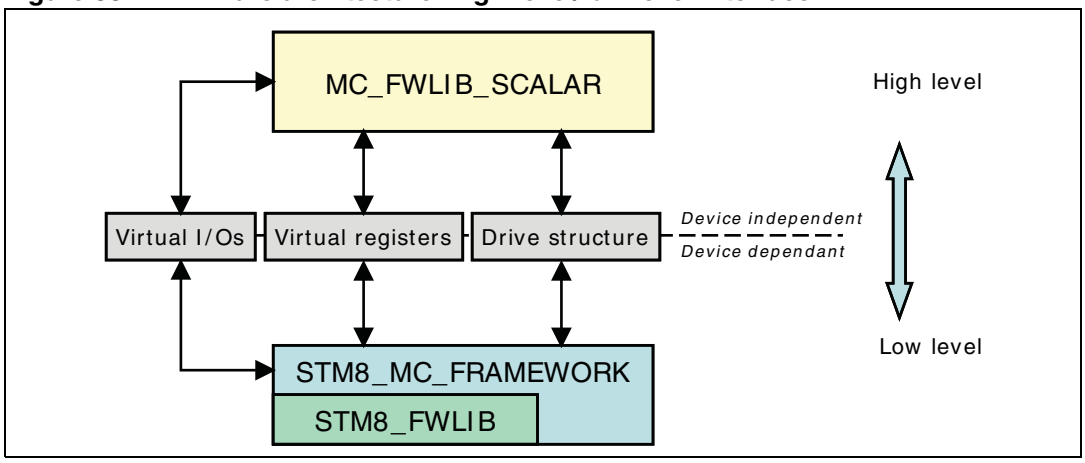


4.2 Library architecture

The STM8S BLDC library has been logically divided in three different parts:

- MC_FWLIB_SCALAR - containing the high-level motor control modules.
- STM8_FWLIB - containing the STM8 standard libraries.
- STM8_MC_FRAMEWORK - containing the low-level motor control routines.

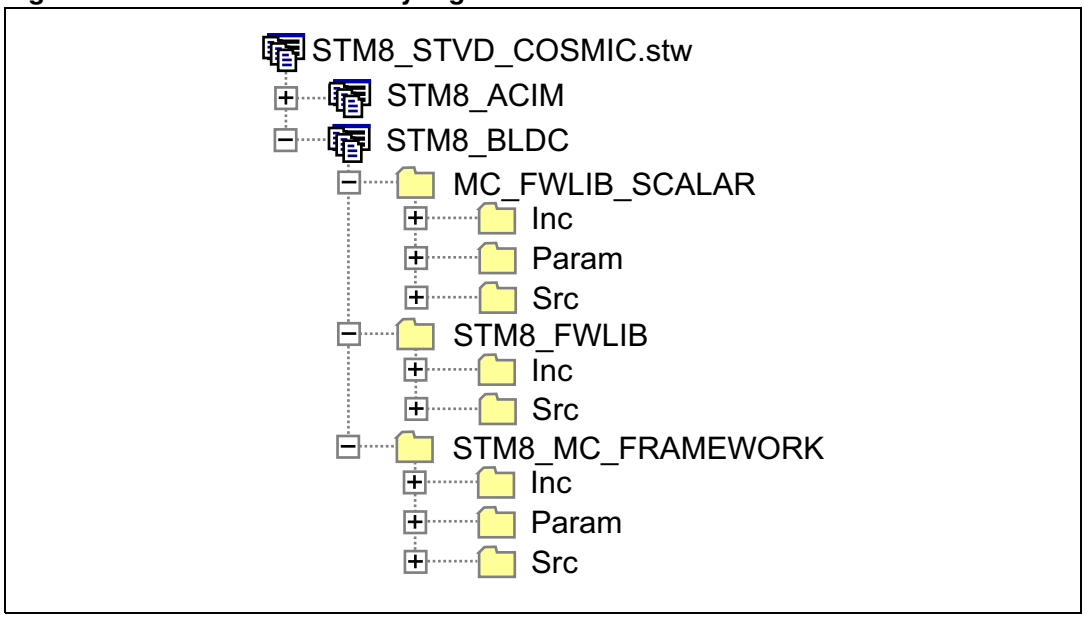
Figure 33. Firmware architecture: high-level/low-level interface



Each part is in turn divided into is divided in three sublevels like shown in *Figure 34*:

- **Inc** folder - containing the prototype definitions (.h files)
- **Src** folder - containing the implementation files (.c files)
- **Param** folder - containing the configuration files (.h file). The configuration files contains all the required define statements allowing to customize the motor control drive. **Param** folder is not present for the **STM8_FWLIB**.

Figure 34. STM8S BLDC library organization



The high-level modules contain the device-independent algorithms, while the low-level modules contain the hardware-dependent code. It means that only the low-level modules interact directly with the peripherals and the interrupt service routines of the microcontroller. The high-level modules interact with the low-level modules mainly through three interfaces (see *Figure 33*):

- The virtual registers
- The virtual I/Os
- The drive structure

4.2.1 Virtual registers

The virtual registers are composed of two sets of 8-bit and 16-bit registers. Refer to *Table 4* for a description of the virtual registers implemented in the BLDC firmware.

Table 4. Virtual registers

Name	Size	Description
VDEV_REG8_HEATSINK_TEMPERATURE	8 bits	Contains the measured heat sink temperature.
VDEV_REG16_HALL_COUNTS	16 bits	Contains the step time measured with the Hall sensors. It is expressed as a number of PWM periods.
VDEV_REG16_BEMF_COUNTS	16 bits	Contains the step time measured with the BEMF detector. It is expressed as a number of PWM periods.
VDEV_REG16_BLDC_DUTY_CYCLE_COUNTS	16 bits	Contains the number of timer counts of the control variable.
VDEV_REG16_BOARD_BUS_VOLTAGE	16 bits	Contains the measured bus voltage expressed in Volts.
VDEV_REG16_HW_ERROR_OCCURRED	16 bits	Each bit represent an error condition which occurred.
VDEV_REG16_HW_ERROR_ACTUAL	16 bits	Each bit represent an actual error condition.

4.2.2 Virtual I/Os

The virtual I/Os are low-level functions called by the high-level modules. For instance if an high-level module want to set a GPIO high-level output as test pin, it should call the virtual I/Os out8(VDEV_OUT8_LED_1, LED_ON) instead of driving directly the microcontroller register. The virtual I/Os are summarized in *Table 5*.

Table 5. Virtual I/Os

Name	Description
out8(VDEV_OUT8_DISPLAY_FLUSH,none)	This call is used to invoke a refresh of the LCD screen.
out8(VDEV_OUT8_DISPLAY_PRINTCH,none)	This call is used to refresh only the cursor displayed on the LCD (for example blinking).

Table 5. Virtual I/Os (continued)

Name	Description
out8(VDEV_OUT8_LED_1,command)	This call is used to drive the virtual LED 1 this is mapped to a real H0 pin of the evaluation board. The command can take the following values: LED_ON: Set a high state to the output turning on the related LED. LED_OFF: Set a low state of the output turning off the related LED. LED_TOGGLE: Perform a toggle on the output switching on if it is off and vice versa.
out8(VDEV_OUT8_LED_2,command)	Same as above but related to virtual LED 2 (H1 pin of the evaluation board).
out8(VDEV_OUT8_LED_3,command)	Same as before but related to virtual LED 3 (H2 pin of the evaluation board).
out8(VDEV_OUT8_LED_4,command)	Same as before but related to virtual LED 4 (H3 pin of the evaluation board).
inp8(VDEV_INP8_USER_INPUT,none)	This call is used to get the input from the user interface (Joystick and Key button)

4.2.3 Drive structure

The drive structure contains the variable and parameters related to the motor and the drive. The BLDC drive structure is show in *Table 6*.

Table 6. BLDC Drive structure

Name	Type	Description
bSpeed_Control_Mode	Variable enum	It specifies the speed control mode. It can be of two type open loop or closed loop.
bCurrent_Control_Mode	Variable enum	It specifies the current control mode. It can be of two type current mode or voltage mode.
hMeasured_rotor_speed	Variable (16 bits)	It contains the measured rotor speed express in rpm.
hTarget_rotor_speed	Variable (16 bits)	It contains the target rotor speed express in rpm.
hDuty_cycle	Variable (16 bits)	It contains the control variable for voltage mode (Duty cycle) express in timer counts.
hCurrent_reference	Variable (16 bits)	It contains the control variable for current mode (Current reference) express in mA.
hCurrent_measured	Variable (16 bits)	It contains the measured value of DC current express in mA.
bFalling_Delay	Variable (8 bits)	It contains the falling delay coefficient express in 1/255 of the step time.
bRising_Delay	Variable (8 bits)	It contains the rising delay coefficient express in 1/255 of the step time.
bDemag_Time	Variable (8 bits)	It contains the demagnetization time express in percentage of step time.

Table 6. BLDC Drive structure (continued)

Name	Type	Description
hMinimumOffTime	Variable (16 bits)	It contains the minimum off time express in ns.
hBusVoltage	Variable (16 bits)	It contains the measured DC bus voltage express in Volt.
bHeatsinkTemp	Variable (8 bits)	It contains the measured heat sink temperature express in °C.
bFastDemag	Variable (on/off)	It is used to enable or disable the fast demagnetization option.
bToggleMode	Variable (on/off)	It is used to enable or disable the toggle mode option.
bAutoDelay	Variable (on/off)	It is used to enable or disable the Auto-delay option.
bMotor_Pole_Pairs	Constant (8 bits)	It expresses the number of motor poles pairs.
hMax_Speed	Constant (16 bits)	It expresses the motor maximum speed in rpm.
hPWM_Frequency	Constant (16 bits)	It expresses the PWM signal frequency in Hz.
bSpeed_PID_sampling_time	Constant (8 bits)	It expresses the PID regulator interval of action in milliseconds.
pPID_Speed	Constant pointer	Pointer to the speed PID structure.
hCurrent_Limitation	Constant (16 bits)	It contains the value of the current limitation express in mA.

4.3 Low-level control

This section describes the implementation of the low-level drive, which is interfaced with the microcontroller (peripheral, memory,...).

4.3.1 Using the advanced control timer peripheral (TIM1)

The BLDC drive can be achieved by using STM8S 16-bit advanced control timer (TIM1) peripheral to generate the control signals for the inverter power switches as described in *Figure 35*. The OCx output is connected to the high side switch while the OCNx output is connected to the low side switch.

For a given step, four configurations are possible for each TIM1 channel output (see *Table 7*).

Figure 35. Timer output control signals

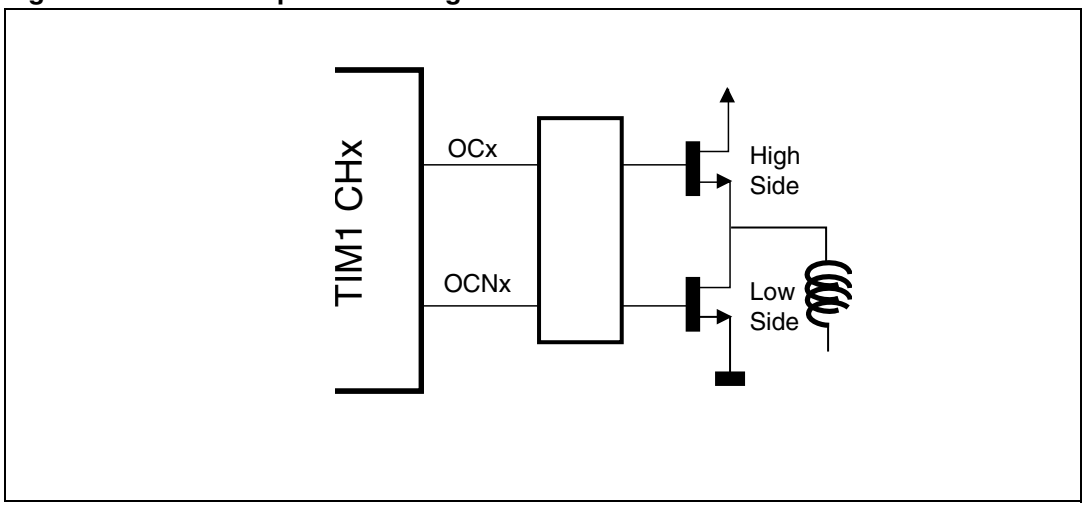


Table 7. TIM1 channel output configurations

Output configuration	High side	Low Side	Type of outputs
Phase floating	Off	Off	Independent
PWM signal applied to the high side	PWM	Off	
PWM signal applied to the low side	Off	PWM	
High side active	On	Off	Complementary
Low side active	Off	On	

The ability of TIM1 to generate complementary or independent channel outputs has been used.

Independent or complementary channel outputs

To generate independent outputs, the right channel outputs must be enabled. This is done by setting CCxE and CCxNE bits in the Capture/compare enable registers TIM1_CCERx registers as shown in *Table 8*. The PWM mode is selected by setting the OCxM[2:0] bits of the Capture/compare mode registers TIM1_CCMRx to PWM mode 1 (active in up counting until the counter reaches the compare register).

Note: When two outputs of a timer channel are independent, one of the two outputs must be "off". In this case the pin output is no more driven by the timer and the pin polarity cannot be configured by the OCxP bits. In this case the polarity of the correspondent GPIO pin must be set according to the polarity required by the hardware.

Table 8. Capture/compare enable bit register configuration

High side	Low side	CCxE	CCxNE
Off	Off	0	0
PWM	Off	1	0
Off	PWM	0	1

To generate complementary channel outputs, both outputs must be enabled. This is done by setting both CCxE and CCxNE bits in CCERx registers. Since no PWM is required on these signals, the OCxM[2:0] bits must be configured to ‘forced active’ or ‘forced inactive’ to activate the high side or the low side, respectively.

Refer to STM8S reference manual (RM0008) for a detailed description of TIM1 operation.

Current regulation/limitation

Current regulation and limitation have been implemented using the “clear enable” feature of STM8S TIM1 advanced control timer. This feature allows the asynchronous turn off of the PWM applied if a triggering signal is present on the dedicated ETR pin (external trigger). This pin is driven by an external comparator that generates the trigger event when the measured DC current is higher than the threshold.

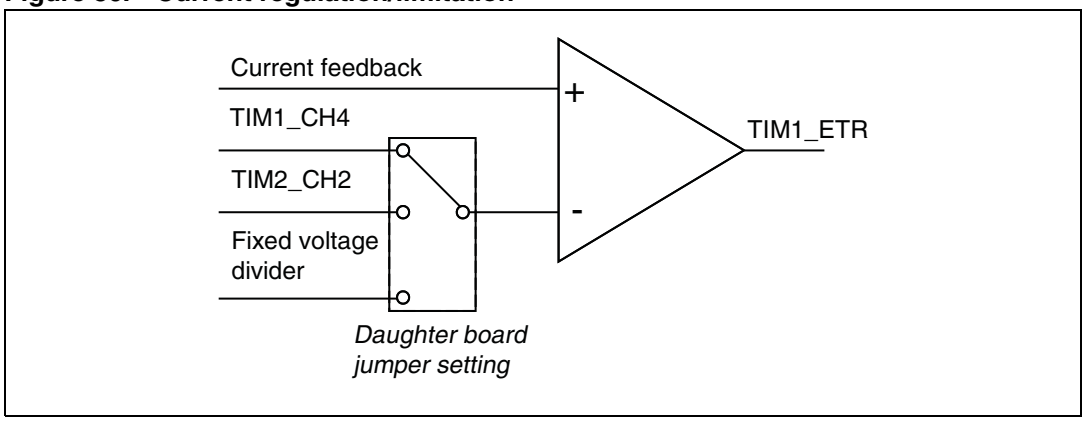
The value of comparator threshold used for current regulation or limitation can be selected using two jumpers in the daughter board (See *Figure 36*).

Three different sources can be configured:

- TIM2_CH2 output for sensorless control (current reference or current limitation)
- TIM1_CH4 output for sensored control (current reference or current limitation)
- Fixed voltage divider for both control (current limitation only)

So it is possible to free the resource needed to drive the threshold if only a current limitation is required (Voltage mode) setting the fixed voltage divider.

Figure 36. Current regulation/limitation



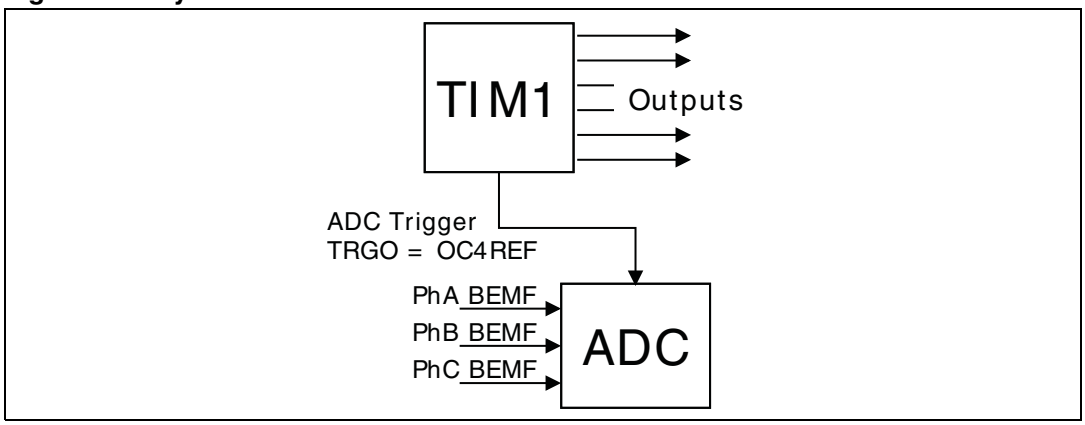
4.3.2 Using the ADC

The STM8S 10-bit analog/digital converter (ADC) is used to achieve sensorless motor control by performing BEMF zero crossing detection.

The BEMF sampling method implies that ADC sampling is performed at a specific position inside the PWM period (see *Section 2.4*). This is essential not only to perform BEMF signal sampling during on- or off-time, but also to specify the stabilization time between the sampling and the turning on and off the PWM to avoid commutation noises.

This is done by synchronizing the STM8S ADC conversion with the PWM generated by TIM1. TIM1 channel 4 is used to trigger the start of the ADC conversion (see *Figure 37*).

Figure 37. Synchronization between TIM1 and ADC



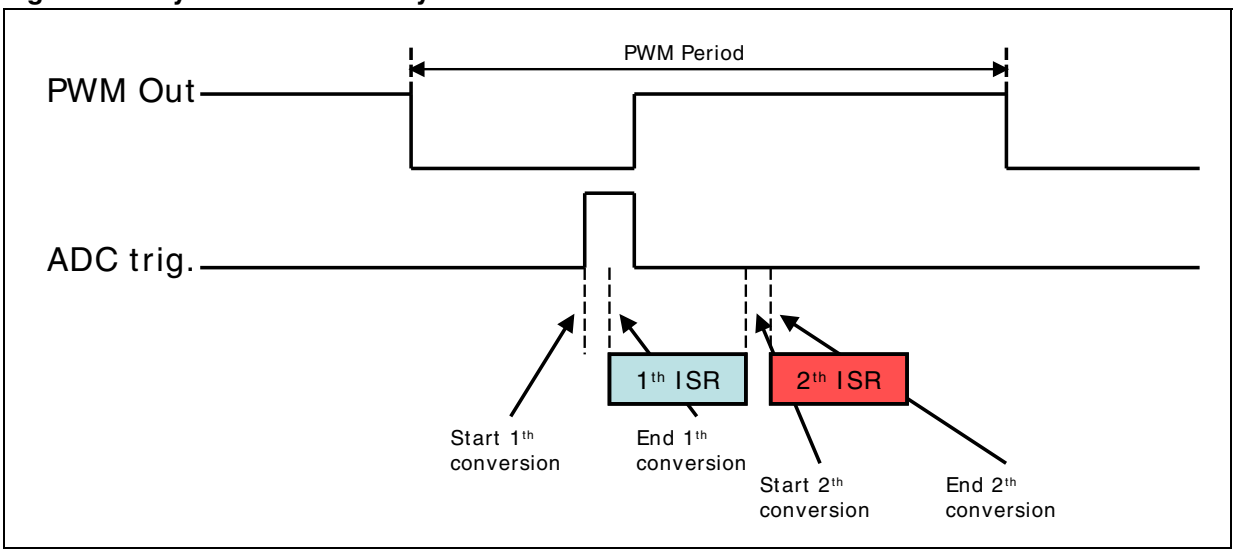
When the conversion is complete, the related interrupt routine is serviced. This routine compares the converted value with threshold to determine if a zero crossing occurred:

- BEMF conversion during T-off: the threshold is fixed and can be customized by the user using define statements (see *Section 5.1.3: BLDC drive control parameters (MC_BLDC_Drive_Param.h)*).
- BEMF conversion during T-on: the sampling value it is compared with the motor neutral voltage. The motor neutral voltage it is reconstructed starting from the bus voltage and using a different ADC conversion.

Since the STM8S microcontroller features one single ADC peripheral, it is necessary to configure it to perform all the remaining conversions plus extra conversions that may be requested by the application. This is done by performing two ADC conversions for each PWM period (see *Figure 38*):

- The first conversion is called synchronous sampling. It is triggered by the TIM1 OCREF4 signal to set the sampling point at the desired position inside the PWM period.
- The second conversion is called asynchronous sampling. It starts at the end of the interrupt routine executed after the first conversion.

Figure 38. Synchronous and asynchronous ADC conversion



Synchronous conversion

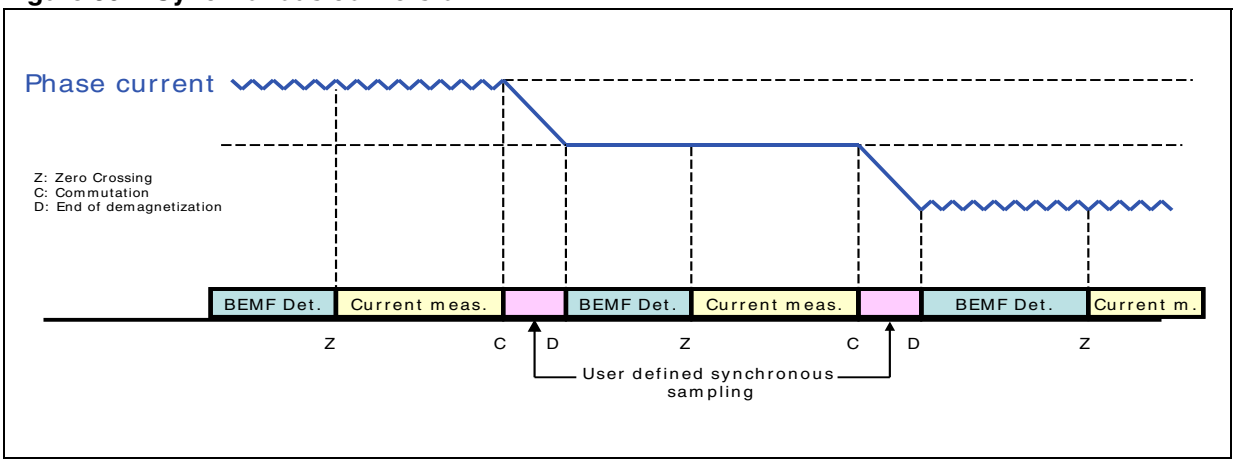
Inside each step (between C and C on *Figure 39*), the synchronous conversion is used for different purposes.

It is dedicated for BEMF zero crossing detection only between the end of demagnetization (D) and the effective “zero crossing” instant (Z). During the remaining time intervals, the synchronous conversion can be used for other purposes inside each step. The interval between zero crossing (Z) and commutation (C) has been dedicated for the current measurement.

During this interval the direct current that flows into the motor is not affected by the switching between two steps and current sampling is performed during the on time.

The remaining interval between the commutation (C) and the end of demagnetization (D) it is left for the user. It is called “user defined synchronous conversion”.

Figure 39. Synchronous conversion



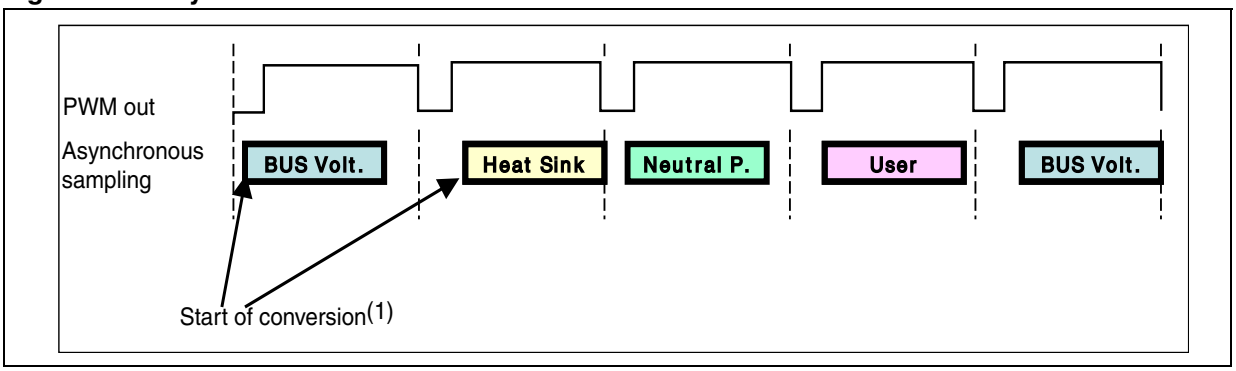
Asynchronous conversion

There is no guarantee that asynchronous conversion is performed during the T-on or T-off, and it is not possible to set a delay with respect to commutations. As a consequence asynchronous conversions are used for conversions that do not require sampling to be performed at a specific point inside the PWM period, such as bus voltage level or the heatsink temperature measurement. (see *Figure 40*).

A list of asynchronous conversions has been defined in the current firmware implementation. It includes bus voltage sampling, heatsink temperature sampling, motor neutral point sampling, plus one user defined conversion (called user defined asynchronous sampling).

Note: Two distinct bus voltage dividers and samplings (bus voltage and motor neutral point) have been implemented in the current implementation to maximize the resolution of the neutral point reconstruction while keeping the compatibility with the power board present in the kit (UM0379). It is possible to design a common voltage divider for bus voltage and motor neutral point based on the final application and use only one sampling.

Figure 40. Asynchronous conversions



1. The start of conversion depends on the conversion and execution time of the first synchronous conversion.

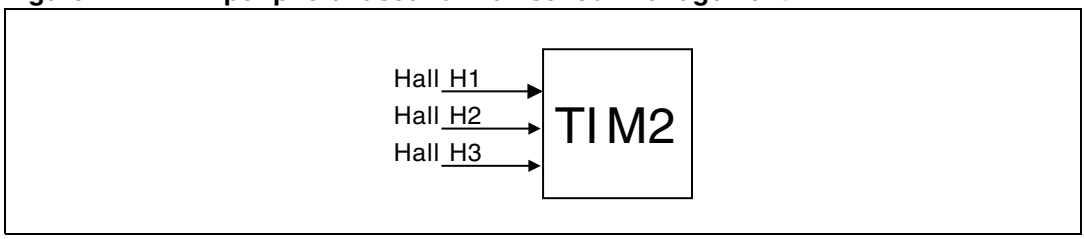
4.3.3 Hall sensor management

When the firmware is configured for sensor drive, the rotor position and speed control are performed by Hall sensors. These measurements are performed using the input capture feature of the STM8S 16-bit general purpose timer (TIM2) (see *Figure 41*).

Before starting the drive, the status of the three Hall sensor inputs is read, and the appropriate step to be applied is selected according to this configuration.

The polarity (expected edge) of the input signals is set as follows: if the value of the Hall sensor signal is high, then the expected edge corresponding to this sensor is a falling edge otherwise it is a rising edge.

Figure 41. TIM2 peripheral used for Hall sensor management

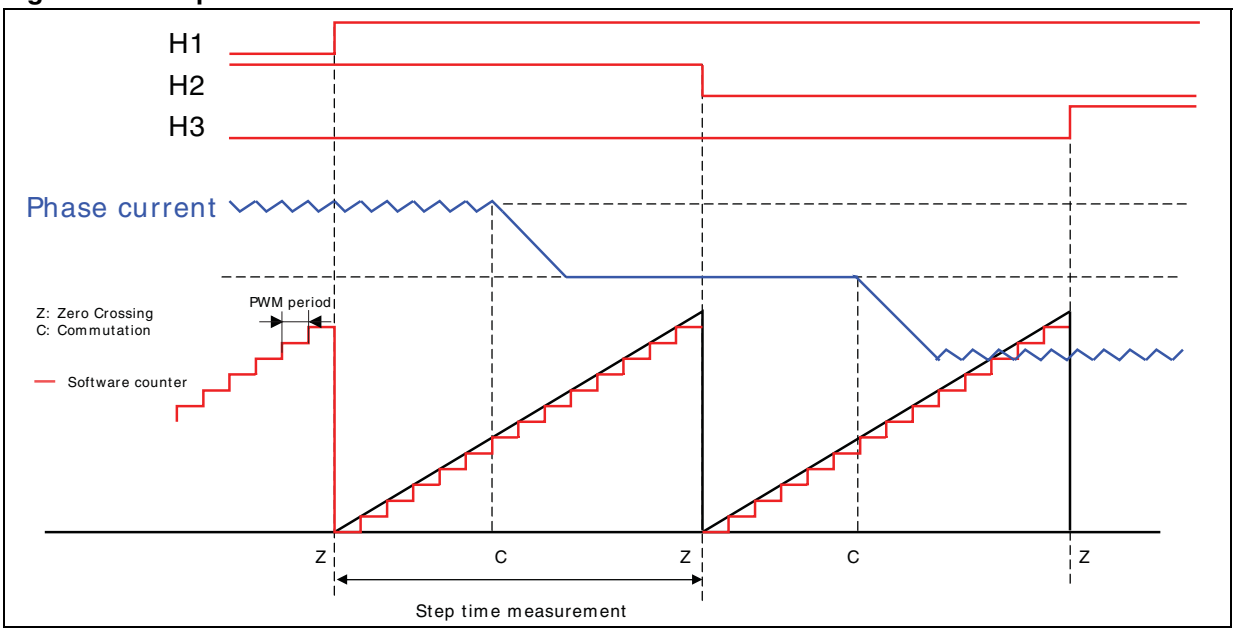


During each further capture, the step time measurement, the change of polarity of the expected new capture of this channel, and the selection of the appropriate steps (redundancy resynchronization) are performed.

As for sensorless drive, the capture of the Hall sensor signal is called “Z capture”. The speed measurement is performed using a software counter that increases each PWM period, stores its value each time a input Z capture occurs, and computes the rotor speed starting from the delta between these values (see *Section 2.2: Rotor speed measurement* and *Figure 42*).

The BLDC sensored drive is based on a commutation table that indicates the related step configuration for each combination of the Hall sensor signals represented by bits H1, H2 and H3 (see *Figure 42*). The BEMF signal is related to the Hall sensors signals. To achieve a maximum efficiency, the step configuration must be synchronized with the Hall sensor configuration. This synchronization is performed by the firmware accordingly to the commutation table.

Figure 42. Step time measurement for sensored drive



The Shinano motor is provided with three Hall sensors spatially distributed at 120° to each other, with a phase shift (electrical delta angle between maximum BEMF of phase A and H1 rising edge) of -60°. The corresponding graph of quantities of interest is given in *Figure 43*.

Figure 43 is used as starting point to extract *Table 9*:

- The first column is the index of the table. It is made up of the seven configurations of the three Hall sensor. It corresponds to the H status of *Figure 43*.
- The second column is shown for clarity but is not included in the final table. It corresponds to H1, H2, and H3 configurations of the H status (see *Figure 43*).
- The third column is the related step coming from the relation between H status and Step to be applied coming from *Figure 43*.

For example, if the configuration of the Hall sensor signals is H1 High, H2 High, and H3 Low, the H status is 6 and the step to be applied is step 2.

Depending on the spatial distribution of the Hall sensors, some configurations may not be allowed, such as configuration ‘000’ and ‘111’. In this case, the related step of the commutation table will be ‘Invalid’ (see *Table 9*). See *Section 5.1.1*, and *Section 5.1.4*, for information on how to customize Hall sensors.

The same procedure can be followed to compute the Hall sensor commutation table required for counter-clockwise direction, or for any other kind of sensor distribution such as 60 ° (see *Figure 43*, *Figure 44*, *Figure 45*, and *Figure 46*).

Figure 43. Shinano motor Hall sensor configuration- 120 °, clockwise direction

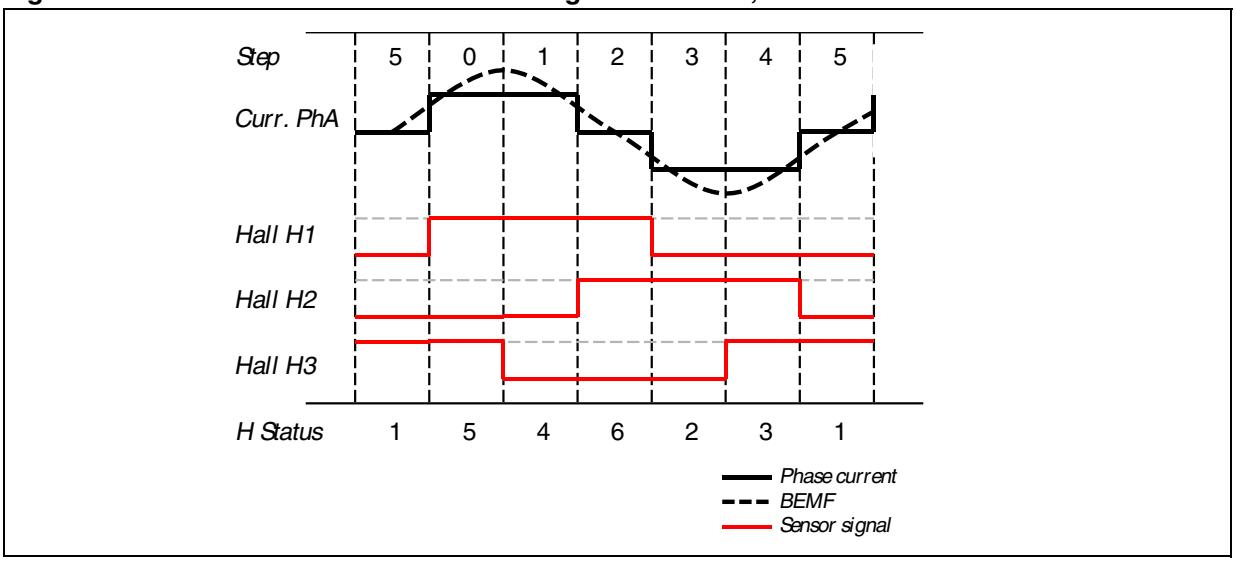


Figure 44. Shinano motor Hall sensor configuration- 120 °, counter-clockwise direction

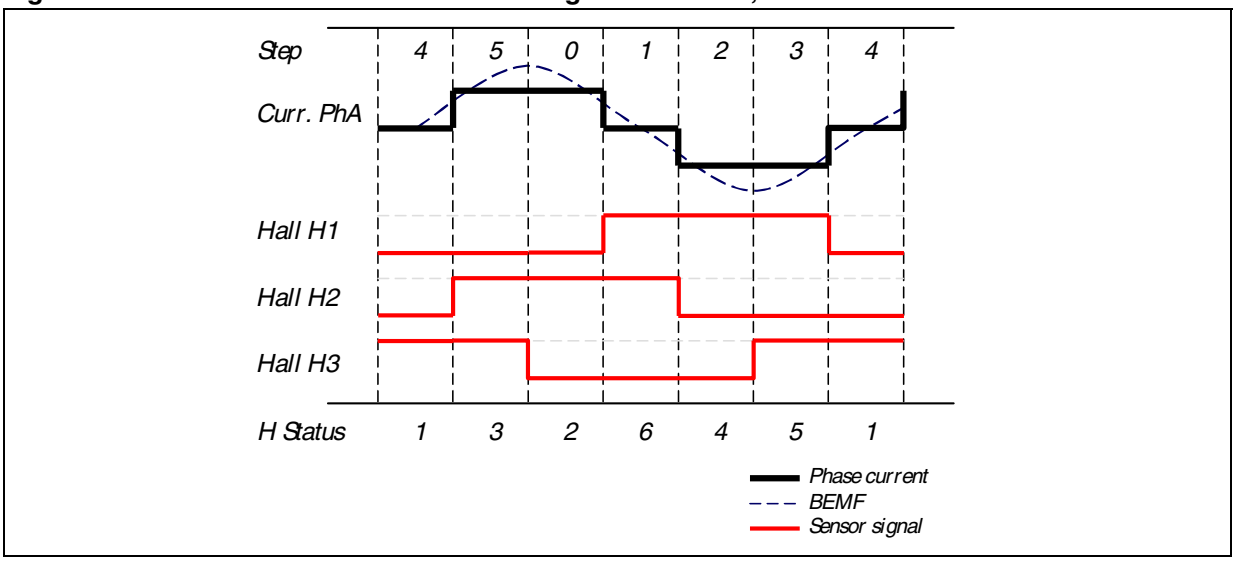


Figure 45. Hall sensor configuration- 60 °, clockwise direction

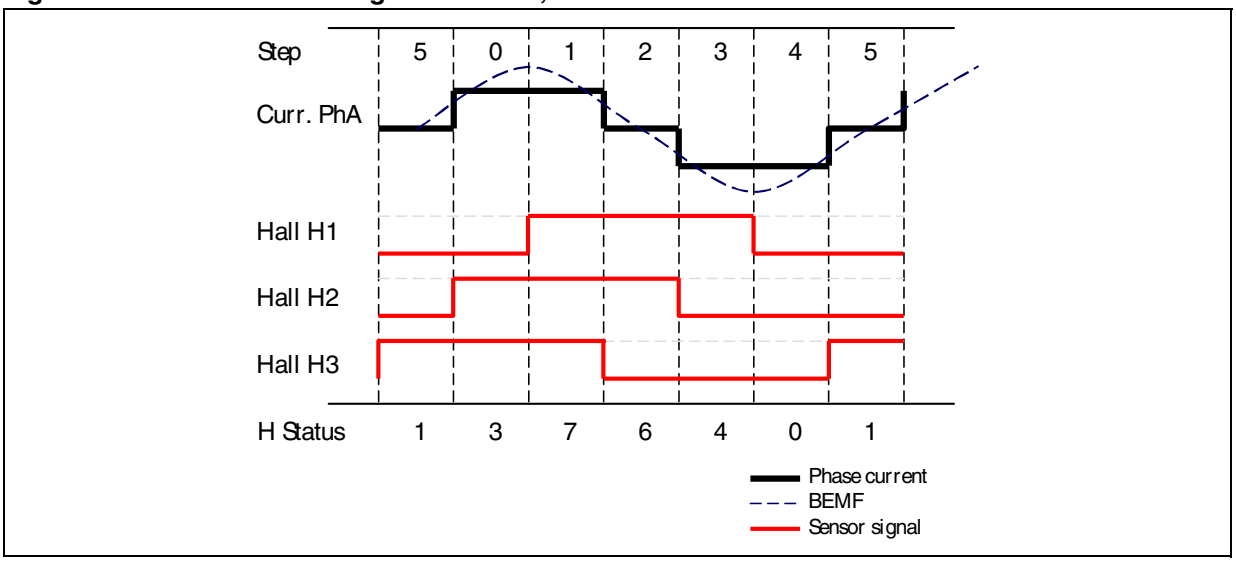


Figure 46. Hall sensor configuration- 60 °, counter-clockwise direction

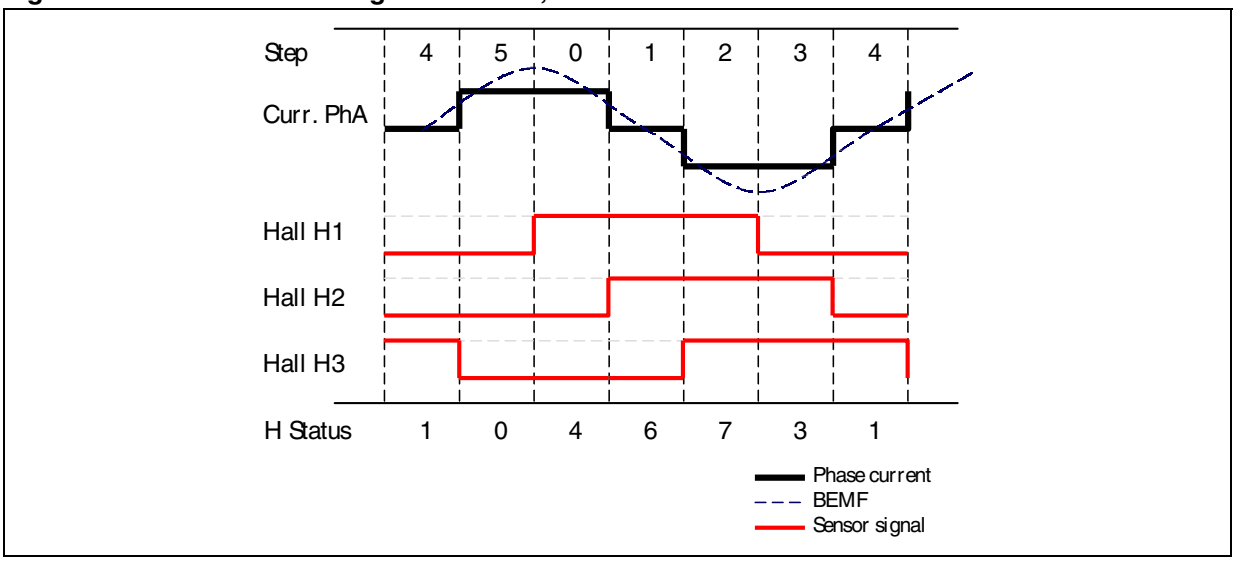


Table 9. Hall sensor commutation table for Shinano motor

H status	H1 H2 H3	Step
0	000	Not valid
1	001	5
2	010	3
3	011	4
4	100	1
5	101	0
6	110	2
7	111	Invalid

4.3.4 Dissipative brake

Because of its physical construction, the BLDC motor is able to transform kinetic energy into electrical energy just like a dynamo.

Under a limited number of conditions this property of BLDC motors has to be taken into consideration to avoid possible damage to the hardware system. For instance, a dangerous situation could arise when:

- The six inverter switches are opened and the motor is running at a speed higher than the nominal one. In this case, the amplitude of the line-to-line BEMF generated is higher than the nominal bus voltage.
- The control tries to brake. The energy is transferred from the load to the board.

Unless the used power system has regenerative capabilities, in both of these situations the inverter bulk capacitor is charged. Furthermore, depending on the rotor speed (with reference to the first situation) or on the amount of energy transferred (with reference to the second situation), the voltage across the bulk capacitors could increase to a destructive level.

A strategy for somehow dissipating the generated electrical energy is thus necessary.

Different methods could be implemented to do so, but one of them in particular, the utilization of a brake resistor, is supported by the library presented in this user manual.

Caution: If the motor is operated beyond the rated speed, it is mandatory to use a regenerative power converter or a brake resistance to prevent bus over voltage from damaging your board.

In the firmware this strategy has been implemented using the analog to digital conversion of the bus voltage value to determine if a voltage level beyond the threshold is present in the bus voltage. If this condition occurs and the brake management is enabled in the firmware the “Over voltage” fault is not more generated instead the break control pin is driven to turn on the external brake resistor in order to dissipate the extra energy.

If the brake resistor is active it is expected that the bus voltage level will decrease so its value monitored, always with the ADC conversion, to detect the dissipative brake action condition to stop.

If the value falls below a threshold the dissipative brake is stopped. In order to add a hysteresis between the switching on and off the turn off threshold is reduced compared to the turn on threshold.

See *Section 5.1.6* for details on the enabling or disabling of this feature, and *Section 5.2* for details on the hardware setup required to use that feature.

4.4 High level control

This section explains how to implement a high level BLDC drive algorithm independently from microcontroller resources and peripheral.

4.4.1 BLDC scalar control

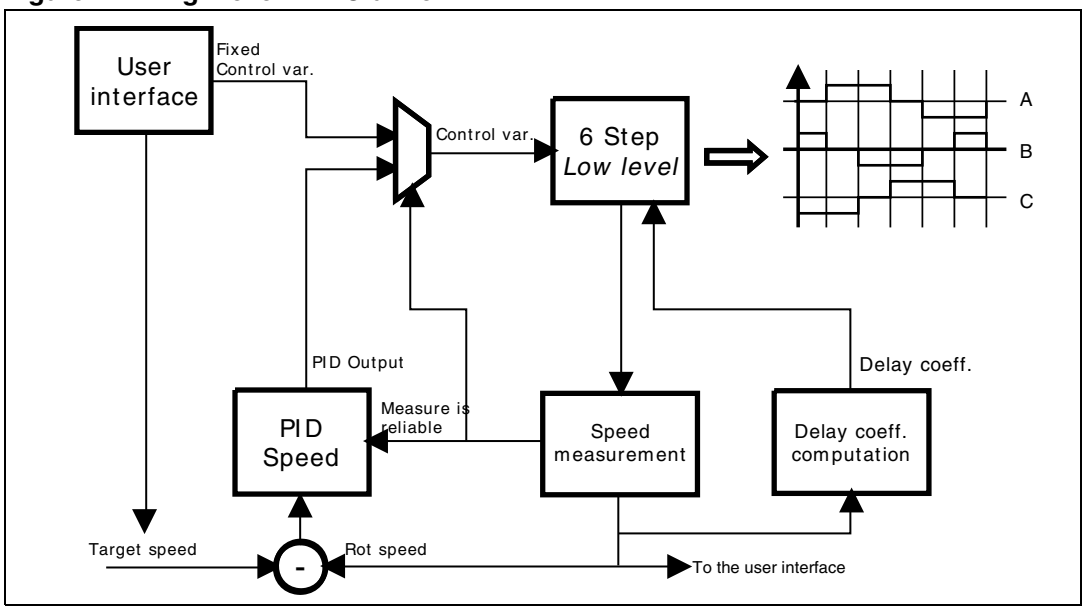
The high level BLDC drive algorithm interacts with the low level control to get the value and reliability of the rotor frequency, and to set the control variables (duty cycle and current reference), and the delay coefficient if the Auto-delay option is enabled.

The core of the high level BLDC drive consists of a 16-bit PID speed controller. The speed measurement block makes the interface between the low level and the high level drive. Starting from the feedback coming from the low level drive, it provides the rotor speed and the related reliability.

When a sensorless control is performed, the speed measurement is not immediately reliable at startup, and the drive operates in speed open loop with control variables configured through by the user interface. As soon as the rotor speed measurement becomes reliable, the speed loop is closed using the PID output. Starting from that control variable the low level generates the output signal to drive the motor to the required speed according to the value of the duty cycle to be applied (in voltage mode) or the level of current reference (in current mode). The synchronization between the BLDC motor stator and rotor is also performed by the low level drive.

In addition, the high level drive computes the delay coefficients starting from the predefined curve (see *Section 2.6*) and using the measured rotor speed if it has been configured.

Figure 47. High level BLDC drive



4.5 Virtual timers

Virtual timers are high-level hardware-independent general purpose counters. These counters are used to manage the execution of the code performed at specified time intervals, such as speed regulation.

The implementation of these virtual timers is based on a physical layer that uses the STM8S resources: the virtual timers are implemented using the TIM4 peripheral. TIM4 is configured to generate an interrupt each millisecond, and is used as time base to update each Virtual timer.

It is possible to use the virtual timers in two modes:

- Polling mode

  In this case, the end of counting has to be check using a specific function call, and the execution of the code is subject to the value returned by this function.
- Automatic mode

  At the end of counting, the specified function is automatically executed.

The virtual timers are implemented in “one-shot”. This means the counting must be restarted each time, whatever the mode.

A set of virtual timers is implemented inside each “drive firmware”. Each virtual timer is dedicated to specific operations, and is identified by a name, VTIMx, where x is the number of the virtual timer. The virtual timer names can be customized through a define statement.

Example

As an example timer number 0 can be named VTIM_KEY by using the following define statement:

```
#define VTIM_KEY VTIM0
```

4.5.1 Using BLDC virtual timers

The list of virtual timers used by the BLDC drive is given in *Table 10*.

Table 10. BLDC virtual timers

Name	Type	Description
VTIM_KEY	Polling	This virtual timer is used for two purposes: It counts the duration of the welcome message. It counts the time interval for the key repetition function. When the joystick is set in one position or the button is pressed, a fixed delay time (KEY_HOLD_TIME = 300 ms) is respected before repeating this function (KEY_REPEAT_TIME = 100 ms). This can be used to increase or decrease a field value by keeping the joystick pressed UP or DOWN.
VTIM_DISPLAY_BLINK	Polling	This virtual timer is used to count the cursor blinking frequency (DISPLAY_BLINKING_TIME = 300 ms).
VTIM_DISPLAY_REFRESH	Polling	This virtual timer is used to count the LCD refresh frequency (DISPLAY_REFRESH_TIME = 300 ms).

Table 10. BLDC virtual timers (continued)

Name	Type	Description
VTIM_USER_INTERFACE_REFRESH	Polling	This virtual timer is used to count the delay time between the visualization of the error messages when several faults occur simultaneously. This delay is set by 1 s by the firmware.
BLDC_CONTROL_TIMER	Automatic	This virtual timer is used to call the BLDC_drive function (see *Section 4.5*).
DEV_DUTY_UPDATE_TIMER	Automatic	This virtual timer is used to update the low level register.
MTC_ALIGN_RAMP_TIMER	Polling	This virtual timer is used for three purposes: It counts the duration of the bootstrap phase (see *Section 2.7*). It counts the interval of the alignment ramp (MTC_ALIGN_RAMP_TIMEOUT = 4 ms). This means that the duty cycle is incremented each 4 ms starting from 0% up to DEFAULT_ALIGN_DUTY_CYCLE during the alignment phase. It counts the duration of the active brake if that function is enabled.
MTC_ALIGN_TIMER	Polling	This virtual timer is used to count the duration of the alignment phase (see *Section 2.7*).
ADC_SAMPLE_TIMER	Automatic	This virtual timer is used to update the firmware variables related to the ADC sampled values. ADC sampling is performed every 10 ms.
HALL_CAPT_TIMEOUT_TIMER	Automatic	This virtual timer is used to identify the rotor stall condition when no more signal coming from the Hall sensor. This allows to set the zero speed.

4.5.2 Virtual timers related functions

Refer to *Section 6.2.1* for the description of the functions using the virtual timers.

5 Designing an application using the BLDC software library

Setting up an operational evaluation platform with a drive system based on the STM8/128-MCKIT and a permanent-magnet motor is quite easy. The BLDC software runs on the STM8S microcontroller on which the STM8/128-MCKIT is based.

This section explains how to quickly configure your drive system, and customize the library accordingly (if needed).

The following steps are required to perform these tasks:

1. Gather all the information regarding the hardware: motor parameters, power device features, speed/position sensor parameters, current sensors transconductance.
2. Use an IDE to edit the following high level parameters files present in the folder **STM8-MC_KIT\MC_FWLIB_SCALAR\param** (see *Figure 34*):
   - `MC_BLDC_conf.h` (see *Section 5.1.1*),
   - `MC_BLDC_Motor_Param.h` (see *Section 5.1.2*),
   - `MC_BLDC_Drive_Param.h` (see *Section 5.1.3*);
3. If the drive is performed using the Hall sensor, edit `MC_hall_param.h` parameter header files (see *Section 5.1.4*);
4. Edit, using an IDE, the following parameter header files, if you hardware is different from the STM8/128-MCKIT:
   - `MC_ControlStage_param.h` (see *Section 5.1.5*),
   - `MC_PowerStage_Param.h` (see *Section 5.1.6*);
5. It may be necessary to edit also the low level parameters header files present in the folder **STM8-MC_KIT\STM8_MC_FRAMEWORK\param**:
   - `MC_stm8s_clk_param.h` (see *Section 5.1.7*),
   - `MC_stm8s_BLDC_param.h` (see *Section 5.1.8*),
   - `MC_stm8s_port_param.h` (see *Section 5.1.9*),
   - `MC_stm8s_hall_param.h` (see *Section 5.1.10*);
6. Re-build the project and download it on the STM8Sxx microcontroller.

Note: These modifications can be performed automatically using the STM8S_MC_Firmware_Library builder (see Section A.3).

5.1 Customizing the BLDC software library parameter file

5.1.1 BLDC configuration file (MC_BLDC_conf.h)

The purpose of this file is to declare the compiler conditional compilation keys that are used throughout the entire library compilation process to select which speed/position sensor is actually used.

If this header file is not edited appropriately (no choice or undefined choice), you receive an error message when building the project. Note that you do not receive an error message if the configuration described in this header file does not match the hardware that is actually in use, or in case of wrong wiring.

The following define statement are used to choose Hall sensors or sensorless drive, depending on your application requirements:

- `#define HALL`

  Uncomment this define statement when three Hall sensors (60° or 120° distribution) are used to detect rotor position and speed.

  Also fill `MC_hall_prm.h` and `MC_stm8s_hall_param.h` (as explained in *Section 5.1.4* and *Section 5.1.10*)

- `#define SENSORLESS`

  Uncomment this define statement when the BEMF detector is used to detect rotor position and speed.

5.1.2 BLDC motor parameters (MC_BLDC_Motor_Param.h)

The `MC_BLDC_Motor_Param.h` header file contains the parameters related to the motor:

- `#define MOTOR_POLE_PAIRS`

  Defines the number of motor pole pairs.

- `#define MAX_SPEED_RPM`

  Defines the maximum rotor speed expressed in rpm.

5.1.3 BLDC drive control parameters (MC_BLDC_Drive_Param.h)

The `MC_BLDC_Drive_Param.h` header file contains the parameters related to:

- The general drive configuration and parameters
- The BEMF detector configuration and parameters
- The current regulation parameters
- Speed PID controller type, sampling time and initial values
- The optional features initial values and parameters
- The linear variation of delay coefficients according to the mechanical speed

General drive configuration and parameters

- `#define SPEED_CONTROL_MODE CLOSED_LOOP`

  Uncomment this define statement to enable the speed closed loop operation.
- `#define SPEED_CONTROL_MODE OPEN_LOOP`

  Uncomment this define statement to enable the speed open loop operation.
- `#define CURRENT_CONTROL_MODE VOLTAGE_MODE`

  Uncomment this define statement to select the Voltage mode drive (see *Section 2.1* for details).
- `#define CURRENT_CONTROL_MODE CURRENT_MODE`

  Uncomment this define statement to select the Current mode.
- `#define TARGET_ROTOR_SPEED`

  Defines the mechanical rotor speed set point (expressed in rpm) at startup in closed loop mode.
- `#define PWM_FREQUENCY`

  Defines the switching frequency (expressed in Hz) of the applied PWM signal for all configurations.
- `#define DUTY_CYCLE`

  Defines the duty cycle percentage of the applied PWM signals in open loop mode.
- `#define FALLING_DELAY` and `#define RISING_DELAY`

  Defines the values of falling and rising delay coefficients. These values are expressed in 1/256 of the step time. For example the value should be 128 (50% of 256) to set the delay coefficient to 50% of the step time.

Note: The MC_BLDC_Drive_Param.h contains two sections to set these parameters: one for the sensorless configuration and one for the Hall sensor configuration. It is recommended to edit the proper section to avoid undesired behaviors.

- `#define DEMAG_TIME`

  Defines the duration of the demagnetization time. It is expressed as a percentage of the step time.
- `#define MIN_SPEED_01HZ`

  Defines the minimum speed of the measured rotor speed (sensorless configuration) at which the closed loop is validated and the loop closed using the speed controller. This parameter is used only for closed loop operations. It is expressed in tenth of Hz of the electrical frequency.
- `#define ADC_SAMPLE_TIMEOUT`

  Defines the frequency of the update of additional ADC conversions (Bus voltage, heatsink temperature) express in milliseconds (see *Section 4.3.2*).

BEMF detector configuration and parameters

- `#define BEMF_SAMPLING_METHOD`

  This define statement is used to configure the BEMF sampling method. There are three possibilities:

  – *BEMF_SAMPLING_TOFF*: the BEMF detector is configured to perform the sampling only during the off time of the PWM signal applied (see *Section 2.4*);

  – *BEMF_SAMPLING_TON*: the BEMF detector is configured to perform the sampling only during the on time of the PWM signal applied (see *Section 2.4*);

  – *BEMF_SAMPLING_MIXED*: the BEMF detector is configured to perform the dynamic selection of the sampling method based on the value of the duty cycle to be applied. This selection is allowed only if the firmware is configured in Voltage mode.

- `#define SAMPLING_POINT_DURING_TOFF`

  This define statement is used when the BEMF sampling is performed during PWM off time to define the sampling point. This parameter sets the advance of the BEMF sampling point before the turn-on of the PWM signal. It is expressed in nanoseconds.

Note: A minimum time of 1.5 ms is required to perform the BEMF sampling using the STM8Sx ADC peripheral.

- `#define SAMPLING_POINT_DURING_TON`

  This define statement is used when the BEMF sampling is performed during PWM on time to define the sampling point. This parameter sets the delay between the turn on of the PWM signal and the BEMF sampling point. It is expressed in nanoseconds.

Note: 1 This delay is used as time filter for the turn-on noise commutation and must be set accordingly to the hardware. It is possible to set a minimum PWM T-on time by configuring properly the ETR filter when the current limitation or regulation is set (see Section 5.1.8).

2 The MC_BLDC_Drive_Param.h contains two sections to set these parameters: one for the Voltage mode configuration and one for the Current mode configuration. It is recommended to edit the proper section to avoid undesired behaviors.

- `#define BEMF_RISING_THRESHOLD_V` and `#define BEMF_FALLING_THRESHOLD_V`

  Defines the voltage thresholds used by the BEMF detector for the sampling-during-of-time method. The first statement is used for the rising edge detection, while the second is used for the falling edge detection.

- `#define BEMF_SAMPLE_COUNT`

  Defines the numerical filter used by the BEMF detector. This means that ADC conversions must cross N-times (N being `BEMF_SAMPLE_COUNT`) the threshold before validating the zero crossing event.

- `#define DUTY_CYCLE_TH_TON`

  Defines the threshold of the duty cycle percentage used by the BEMF detector to choose dynamically the sampling method. This parameter is used only if `BEMF_SAMPLING_MIXED` method is selected.

  If the instantaneous value of duty cycle if above this threshold, the BEMF sampling during on-time is applied. Otherwise the sampling during off-time is applied;

- `#define MINIMUM_OFF_TIME`

  Defines the value of the minimum off time (in nanosecond) required for the BEMF sampling during the off-time. This parameter is related to the hardware and motor. It depends on:

  – The stabilization time between the turn-off of the switch till the end of commutation noise.

  – The duration of ADC sampling (minimum 1.5 µs).

  This parameter also defines the maximum applicable duty cycle (Maxduty) to allow the BEMF sampling during off time (see *Equation 2*):

Equation 2 Maximum applicable duty cycle for off time BEMF sampling

$$Maxduty = 100 \times \left(1 - \left(MINIMUM_{OFFTIME} \times \frac{PWM\_FREQUENCY(Hz)}{10^9}\right)\right)$$

Note: If the BEMF_SAMPLING_MIXED is selected as BEMF method, then the DUTY_CYCLE_TH_TON must be lower than the maximum applicable duty cycle defined by this parameter.

- `#define ALIGN_DUTY_CYCLE`

  Defines the value of the duty cycle percentage applied during the alignment phase.

- `#define RAMP_DUTY_CYCLE`

  Defines the value of the duty cycle percentage applied during ramp up.

- `#define ALIGN_DURATION`

  Defines the duration (in millisecond) of the alignment phase.

- `#define ALIGN_SLOPE`

  Defines the slope of the duty cycle (in milliseconds) during the alignment phase. The duty cycle applied during the alignment phase starts from zero up to `ALIGN_DUTY_CYCLE`, incremented each `ALIGN_SLOPE` milliseconds. The increment is automatically computed from the other parameters.

- `#define FORCED_STARTUP_STEPS`

  Defines the number of forced steps which are always performed during ramp-up before starting the BEMF sampling.

Current regulation parameters

- `#define CURRENT_REFERENCE`

  Defines the initial value of the current reference expressed in mA. This parameter is used only in Current mode.

- `#define CURRENT_LIMITATION`

  Defines the value of the current limitation expressed in mA. This parameter is used both in Voltage and Current mode to set the maximum allowed DC current. This parameter is not used during the alignment and ramp-up phases.

- `#define STARTUP_CURRENT_LIMITATION`

  Defines the value of the current limitation (expressed in mA) used during the alignment and ramp-up phases. This parameter is used both in Voltage and Current mode to set the maximum allowed DC current.

Type of speed PID-controller, sampling time and initial values

- `#define SPEED_PID_TYPE`

  This define statement is used to configure the type of speed controller. There are two possible setting:

  – PI - sets the proportional integral regulator

  – PID - sets the proportional integral derivative regulator.

- `#define SPEED_PID_SAMPLING_TIME`

  This define statement is used to select the speed regulation loop frequency. It is expressed in milliseconds.

- `#define SPEED_KP`

  This define statement is used to configure the proportional constant of the speed loop regulation (16-bit value, adjustable from 0 to 32767).

- `#define SPEED_KI`

  This define statement is used to configure the integral constant of the speed loop regulation (16-bit value, adjustable from 0 to 32767).

- `#define SPEED_KD`

  This define statement is used to configure the derivative constant of the speed loop regulation (16-bit value, adjustable from 0 to 32767). This parameter is used only if a PID controller has been selected.

- `#define SPEED_KP_DIVISOR`

  This define statement is used to configure the scaling factor of the proportional gain for the speed regulation loop (16-bit power-of-two value).

- `#define SPEED_KI_DIVISOR`

  This define statement is used to configure the scaling factor of the integral gain for the speed regulation loop (16-bit power-of-two value).

- `#define SPEED_KD_DIVISOR`

  This define statement is used to configure the scaling factor of the differential gain for the speed regulation loop (16-bit power-of-two value). This parameter is used only if PID controller has been selected.

- `#define SPEED_OUT_MAX`

  This define statement sets the positive saturation value of the speed controller output. This value should be equal to the maximum value applicable for the control variable.

  For Voltage mode, `SPEED_OUT_MAX` should be equal to the maximum value of timer counter. It is related to the PWM frequency and CPU frequency (see *Equation 3*):

Equation 3 Maximum timer counter in Voltage mode

$$MaxtimerFreq = \frac{CPUFreq(Hz)}{PWMFreq(Hz))}$$

For Current mode, `SPEED_OUT_MAX` should be set to the `CURRENT_LIMITATION` value.

Optional features initial values and parameters

- `#define FAST_DEMAG`

  Defines the starting configuration of the fast demagnetization option (see *Section 2.5*).

  1: Fast demagnetization enabled

  0: Fast demagnetization disabled

- `#define TOGGLE_MODE`

  Defines the starting configuration of the toggle mode option (see *Section 3.6.7*).

  1: Toggle mode enabled

  0: Toggle mode disabled

- `#define ACTIVE_BRAKE`

  Uncomment this define statement to enable the active brake function (see *Section 2.8*).

- `#define BRAKE_DURATION`

  If the active brake function is enabled, use this define statement to specify the duration (in milliseconds) of the active brake action

- `#define BRAKE_DUTY`

  If the active brake function is enabled, use this define statement to specify the percentage of the applied duty cycle.

- `#define AUTO_DELAY`

  Defines the starting configuration for the Auto-delay option. If this function is enabled, the value of the delay coefficients are computed starting from a predefined curve (see *Section 2.6*).

  1: Auto-delay feature enabled

  0: Auto-delay feature disabled

- `#define RISE_FALL_DELAY_LINK`

  Comment this define statement to change the rising and falling coefficients independently.

  When left uncommented, the two coefficients are linked and have the same value if changed by the user. This setting is not used if the Auto-delay function is enabled.

- `#define DEBUG_PINS`

  Uncomment this define statement to enable the debug pins. When this feature is enabled, the test pins are configured to generate in real time dedicated function of the drive, such as C, D, Z, Auto switch, BEMF detection during on-time events. This is very useful when debugging the application. *Table 11* shows the signal output on the test pins when the debug option is enabled. See also *Section 5.1.8* for details on how to configure the test pins.

Table 11. Debug pins description

Name	Default pin	Description
Z Event	H0	This signal toggles each time a zero crossing event is detected.
C/D Events	H1	In sensorless configuration, this signal shows a rising edge for each commutation and a falling edge at the end of demagnetization.

Table 11. Debug pins description (continued)

Name	Default pin	Description
Auto switch	H2	This signal becomes high when the drive switches to Auto-commutation mode (see *Section 2.7*).
BEMF sampling during on time	H3	This signal becomes high when the BEMF sampling mode is performed during the on time. Otherwise it is performed during the off-time.

Linear variation of delay coefficients according to mechanical speed

If the Auto-delay function is enabled then the rising and falling coefficient will be computed starting from a predefined curve. This curve is set by the user using the following defines. These settings will not be used if the “Auto delay” function is disabled (see *Section 2.6*).

- `#define Freq_Min`

  Defines the minimum frequency of the rotor speed curve. It is expressed in rpm.
- `#define Rising_Fmin and #define Falling_Fmin`

  Defines the delay coefficients applied for rotor speeds lower than or equal to the minimum frequency `Freq_Min`. All the delay coefficients are expressed as 1/256 of the step time.
- `#define F_1`

  Defines the motor speed for the first intermediate value of the curve (see *Figure 13*). It is expressed in rpm.
- `#define Rising_F_1` and `#define Falling_F_1`

  Defines the delay coefficients for the first intermediate value of the curve.
- `#define F_2`

  Defines the motor speed for the second intermediate value of the curve. It is expressed in rpm.
- `#define Rising_F_2` and `#define Falling_F_2`

  Defines the delay coefficients for the second intermediate value of the curve.
- `#define Freq_Max`

  Defines the maximum frequency of the rotor speed curve. It is expressed in rpm.
- `#define Rising_Fmax` and `#define Falling_Fmax`

  Defines the delay coefficients applied for rotor speeds higher than or equal to the maximum frequency `Freq_Max`.

5.1.4 Hall sensor parameters (MC_hall_param.h)

The `MC_hall_param.h` header file contains the parameters related to the Hall sensors. These settings are used only in sensored configuration:

- `#define HALL_SENSORS_PLACEMENT`

  Defines the Hall sensor distribution. It can be set to `DEGREES_60` or to `DEGREES_120`. The user can select the required sensor distribution and arrange the hall sensors connection in order to reproduce the configuration described in the corresponding figure (*Figure 43*, *Figure 44*, *Figure 45*, and *Figure 46*). In this case it is possible to use the default values of `HALL_SENSOR_STEPS_CW_xxx` and `HALL_SENSOR_STEPS_CCW_xxx`. Otherwise the user should edit the

HALL_SENSOR_STEPS_CW_xxx and HALL_SENSOR_STEPS_CCW_xxx commutation tables according to its own configuration.

- #define HALL_SENSOR_STEPS_CW_xxx and HALL_SENSOR_STEPS_CCW_xxx

  These define statements set the Hall sensor commutation tables required for the drive. Refer to *Section 4.3.3* for details on the commutation table.

- #define HALL_MAX_SPEED_01HZ

  Defines the rotor mechanical frequency above which speed feedback is not realistic. This function can be used to discriminate glitches.

- #define HALL_MIN_SPEED_01HZ

  Defines the rotor mechanical frequency below which speed feedback is not realistic. This function can be used to discriminate too low frequencies.

- #define HALL_CAPT_TIMEOUT_MS

  Defines the duration of the Hall sensor signal timeout in order to detect the zero speed.

- #define HALL_MAX_ERROR_NUMBER

  Defines the number of speed error measurement occurrences before signaling the error on speed feedback.

5.1.5 Control stage parameters (MC_ControlStage_param.h)

The MC_ControlStage_param.h header file contains the parameters related to the control stage. These settings must be modified if the firmware is used with a customized hardware different from the one of the kit, or to disable some library features in order to reduce code size and CPU occupation:

- #define DISPLAY

  Uncomment this define statement to select the control board LCD as display.

- #define DAC_FUNCTIONALITY

  The DAC functionality is a debug option which can be used to analyze the behaviors of up to two variables inside the code. The variables to be analyzed should not vary more than 20 kHz. See *Section A.1* for details on how to customize it.

Note: The DAC functionality cannot be set together with the dissipative brake option.

- `#define TIM1_CHxN_REMAP`

  Uncomment this define statement to remap the TIM1_CH1N, TIM1_CH2N, TIM1_CH3_N and TIM1_ETR pins. This remapping is necessary if the STM8S features less than 80 pins.

- `#define BKIN`

  Comment this define statement to disable the emergency input feature of the advanced control timer.

- `#define JOYSTICK`

  Comment this define statement to disable the joystick input.

- `#define SET_TARGET_SPEED_BY_POTENTIOMETER`

  Uncomment this define statement to use the potentiometer RV1 available on the MB631. This potentiometer allows to set the target rotor speed. In this case the rotor speed cannot be modified using the joystick.

- `#define AUTO_START_UP`

  Uncomment this define statement to disable the KEY button management. The motor is start to run automatically few second after the reset.

- `#define ENABLE_OPTION_BYTE_PROGRAMMING`

  Comment this define statement to disable in-application option-byte re-programming.

Note: Disabling this option allows to decrease the firmware size. In this case the option bytes can be programmed off-line by using the STVP tool.

5.1.6 Power stage parameters (MC_PowerStage_Param.h)

The `MC_PowerStage_param.h` header file contains the parameters related to the power stage. These settings must be modified if the firmware is used with a customized hardware different from the one of the kit, or to enable/disable unused library features.

- `#define RS_M`

  Defines the value of the power board shunt resistor (in mΩ) used for the current management.

- `#define AOP`

  Defines the value of the voltage gains of the power board amplification stage used for the current management.

- `#define DISSIPATIVE_BRAKE`

  Uncomment this define statement to enable the dissipative brake function (see *Section 4.3.4*). The next define statement must be edited when this feature is enabled.

- `#define DISSIPATIVE_BRAKE_POL`

  This define statement is used to set the polarity of the dissipative brake signal. It should be set according to the dissipative brake hardware implementation. This setting is not used if the dissipative brake function is disabled. There are two available options:

  – DISSIPATIVE_BRAKE_ACTIVE_HIGH: the braking action is triggered by a high logic level of the dissipative brake control signal.

  – DISSIPATIVE_BRAKE_ACTIVE_LOW: the braking action is triggered by a low logic level of the dissipative brake control signal.

- `#define BUS_VOLTAGE_MEASUREMENT`

  This define statement is used to configure the firmware to perform DC bus voltage measurement. If the hardware does not support bus voltage measurement, or if you want to disable this feature, leave this define statement uncommented. The bus voltage will not be measured by the firmware, and will be assumed constant and equal to the value specified in the next define statement.

- `#define BUS_VOLTAGE_VALUE`

  Defines the constant value of the bus voltage if the bus voltage measurement feature has been disabled. This setting is not used if the bus voltage measurement function is enabled.

- `#define BUS_ADC_CONV_RATIO`

  Defines the DC bus voltage partitioning ratio performed by the hardware to allow the bus voltage measurement. This setting is not used if the bus voltage measurement function is disabled.

- `#define EXPECTED_MCU_VOLTAGE`

  Defines the reference value of ADC conversions. ADC conversions are usually performed using a voltage reference that is identical to the microcontroller power supply voltage (5 V). To increase the resolution, it is possible to design a customized hardware that uses a lower ADC reference value. In this case `EXPECTED_MCU_VOLTAGE` contains the required value used for the computation of the converted values. The precision of the measurement can also be improved by setting `EXPECTED_MCU_VOLTAGE` to the appropriate value. For example if the microcontroller measured power supply voltage is 5.1 V, it is possible to set

EXPECTED_MCU_VOLTAGE to 5.1 to maximize the precision in the computation of the converted values.

- #define MAX_BUS_VOLTAGE and #define MIN_BUS_VOLTAGE

  These two values (expressed in Volt) set the bus DC voltage range. If the bus voltage exceeds OVERVOLTAGE_THRESHOLD_V or is below UNDERVOLTAGE_THRESHOLD_V, the corresponding error event is generated and is kept as long as the bus voltage remains outside the allowed range.

  In addition, if DISSIPATIVE_BRAKE is defined, an overvoltage event will be handled by activating the brake resistor, and the corresponding error message will not be issued.

- #define NTC_THRESHOLD_C and #define NTC_HYSTERIS_C

  These two values (expressed in °C) are used to set the power device operating temperature range (measured at heatsink). If the measured temperature exceeds NTC_THRESHOLD_C, the corresponding error event is generated and is kept as long as the measured temperature remains above NTC_THRESHOLD_C - NTC_HYSTERESIS_C.

- #define TEMP_SENS_ALPHA, #define TEMP_SENS_BETA, and #define TEMP_T0

  These three values are used to characterize the transduction curve between temperature sensor value (expressed in °C) and the ADC converted value. This curve is assumed to be linear (see *Figure 48*).

- #define BKIN_POLARITY

  When the firmware runs on a customized hardware, these define statements allow to configure the polarity of the Break input. The polarity can be set to ACTIVE_HIGH or ACTIVE_LOW.

- #define PWM_U_LOW_SIDE_POLARITY, #define PWM_U_HIGH_SIDE_POLARITY, #define PWM_V_LOW_SIDE_POLARITY, #define PWM_V_HIGH_SIDE_POLARITY, #define PWM_W_LOW_SIDE_POLARITY and #define PWM_W_HIGH_SIDE_POLARITY

  When the firmware runs on a customized hardware, these define statements allow to configure the polarity of the PWM output. The polarity can be set to ACTIVE_HIGH or ACTIVE_LOW.

- #define PWM_U_HIGH_SIDE_IDLE_STATE, #define PWM_U_LOW_SIDE_IDLE_STATE, #define PWM_V_HIGH_SIDE_IDLE_STATE, #define PWM_V_LOW_SIDE_IDLE_STATE, #define PWM_W_HIGH_SIDE_IDLE_STATE, #define PWM_W_LOW_SIDE_IDLE_STATE

  When the firmware runs on a customized hardware, these define statements allow to configure the status of the PWM output during the idle state. The status can be set to ACTIVE or INACTIVE.

- #define HEAT_SINK_TEMPERATURE_MEASUREMENT

  This define statement is used to configure the firmware to perform heat sink temperature measurement. If the hardware does not support this feature, or if you want to disable it, leave this define statement uncommented. The heat sink temperature will

not be measured by the firmware, and will be assumed constant and equal to the value specified in the next define statement.

- `#define HEAT_SINK_TEMPERATURE_VALUE`

  Defines the constant value of the heat sink temperature if the heat sink temperature measurement feature has been disabled. This setting is not used if the heat sink temperature measurement function is enabled.

- `#define PWM_LOWSIDE_OUTPUT_ENABLE`

  Comment this define statement to enable the control of the low side switches using standard GPIOs instead of TIM1 outputs. In this case the GPIOs that are used must be defined by changing the following define statements in `MC_stm8s_BLDC_param.h` file:

  #define LS_A_PORT

  #define LS_A_PIN

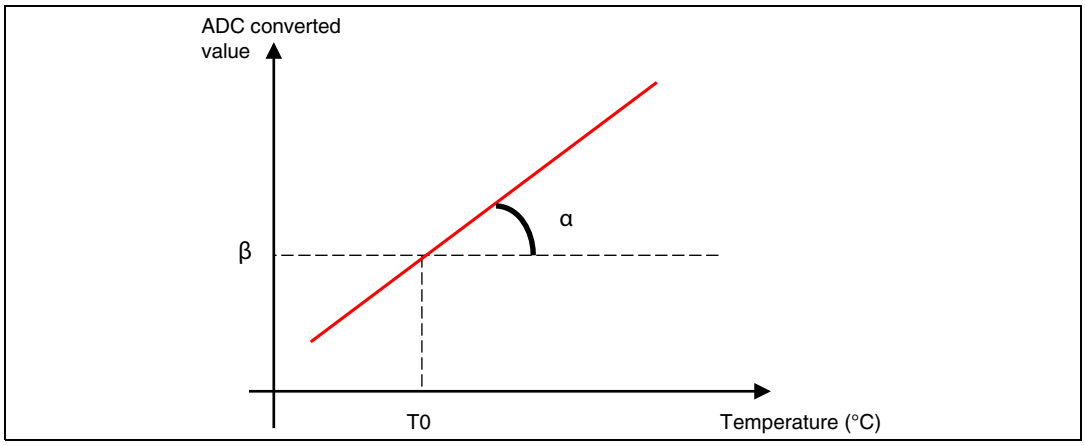
  #define LS_B_PORT

  #define LS_B_PIN

  #define LS_C_PORT

  #define LS_C_PIN

Figure 48. Transduction curve between the temperature sensor and the ADC converted



This curve represents *Equation 4*, where α is defined using `TEMP_SENS_ALPHA`, β is defined using `TEMP_SENS_BETA` and T0 is defined using `TEMP_T0`.

Equation 4 Transduction equation

$$ADC = (\alpha \times t) + \beta - \alpha \times T_0$$

5.1.7 Microcontroller clock definition (MC_stm8s_clk_param.h)

The `MC_stm8s_clk_param.h` header file contains the define statements dedicated to the microcontroller and its peripherals:

- `#define STM8_FREQ_MHZ`

  This define statement is used to set the CPU frequency (express in MHz). It can be set to 16 or 24. When it is set to 24 MHz, the firmware is configured to use the external

oscillator with 1 wait state Flash latency. Otherwise the internal oscillator is used with 0 wait state.

5.1.8 Microcontroller specific BLDC drive parameters (MC_stm8s_BLDC_param.h)

The `MC_stm8s_BLDC_param.h` header file contains the define statements related to the BLDC drive.

- `#define RAMP_VALUExx`

  Defines the values of the ramp-up step durations. These values are pre-computed. They are expressed in PWM periods. They can be customized through the STM8S_MC_Firmware_Library builder (see *Section A.3*).

- `#define PHASE_A_BEMF_ADC_CHAN`, `#define PHASE_B_BEMF_ADC_CHAN`, and `#define PHASE_C_BEMF_ADC_CHAN`

  When the firmware runs on a customized hardware, these define statements allow to configure the channels used to perform the BEMF samplings.

Note: Make sure that these pins are not used for other purposes inside the firmware, and are not configured as outputs.

- `#define ADC_CURRENT_CHANNEL`, *`#define ADC_USER_SYNC_CHANNEL`*, `#define ADC_BUS_CHANNEL`, `#define ADC_NEUTRAL_POINT_CHANNEL`, `#define ADC_TEMP_CHANNEL`, and `#define ADC_USER_ASYNC_CHANNEL`

  When the firmware runs on a customized hardware, these define statements allow to configure the channels used to perform the other ADC samplings (see *Section 4.3.2*).

Note: Make sure that these pins are not used for other purposes inside the firmware, and are not configured as outputs.

- `#define CURRENT_FILTER`

  This define statement is used to configure the filter applied to the ETR signal. The ETR signal is used to manage the current limitation/regulation in the BLDC drive (see *Section 4.3.1*). This filter is used to prevent unwanted shutdown of the PWM signal which may be caused by glitches. Using a larger filter may lead to a less precise current control. It can also be used to define the minimum T-on of the PWM signal required for BEMF sampling (see *Section 2.4*).

  Table 12 gives the values of the allowed filters.

Table 12. External filters

#define CURRENT_FILTER value	Filter for F_CPU= 16 MHz (in µs)	Filter for F_CPU = 24 Mhz (in µs)
CURRENT_FILTER_NOFILTER	0.0625	0.0417
CURRENT_FILTER_F_N2	0.1250	0.0833
CURRENT_FILTER_F_N4	0.2500	0.1667
CURRENT_FILTER_F_N8	0.5000	0.3333
CURRENT_FILTER_F2_N6	0.7500	0.5000
CURRENT_FILTER_F2_N8	1.0000	0.6667
CURRENT_FILTER_F4_N6	1.5000	1.0000
CURRENT_FILTER_F4_N8	2.0000	1.3333
CURRENT_FILTER_F8_N6	3.0000	2.0000
CURRENT_FILTER_F8_N8	4.0000	2.6667
CURRENT_FILTER_F16_N5	5.0000	3.3333
CURRENT_FILTER_F16_N6	6.0000	4.0000
CURRENT_FILTER_F16_N8	8.0000	5.3333
CURRENT_FILTER_F32_N5	10.0000	6.6667
CURRENT_FILTER_F32_N6	12.0000	8.0000
CURRENT_FILTER_F32_N8	16.0000	10.67

- `#define MCI_CONTROL_PINS`, `#define MCI_CONTROL_DDR`, and `#define MCI_CONTROL_DR`

  When the firmware runs on a customized hardware, these define statements can be used to configure the control pins which enable the dividers used for the BEMF sampling during T-on (see *Section 2.4*).

  The first define statement specifies the pins used for OR-ing the BITx which are used (for example BIT4|BIT5|BIT6).

  The second define statement specifies the port used, by replacing the x character in the GPIOI->DDR string by the appropriate letter. If the port used is port I, set GPIOI->DDR.

  The third define statement also specifies the port used. The value should be identical to the second define statement, and the x character in the GPIOI->ODR string should be replaced by the same letter of above.

Note: Three different pins must be used on the same port.

- `#define Z_DEBUG_PORT` and `#define Z_DEBUG_PIN`

  These define statement can be used to set the mapping between the Z event debug signal (see *Section 5.1.3*) and the test pins (see *Section 5.1.9*).

  The first define statement specifies the test pin which is used, by replacing the x character in the DEBUGx_PORT string by the proper number. For instance if the debug pin used is pin 0, set DEBUG0_PORT.

  The second define statement also specifies the test pin by replacing the x character of the DEBUGx_PIN string by the same number as in DEBUGx_PIN. This setting is not used if the debug feature is not enabled.

- `#define C_D_DEBUG_PORT` and `#define C_D_DEBUG_PIN`

  These define statement can be used to set the mapping between the C-D event debug signal (see *Section 5.1.3*) and the test pins (see *Section 5.1.9*).

  The first define statement specifies the test pin by replacing the x character in the DEBUGx_PORT string by the proper number. For instance, if the debug pin used is pin 0, set DEBUG0_PORT.

  The second define statement also specifies the test pin used by replacing the x character in the DEBUGx_PIN string by the same number as in the first define statement. This setting is not used if the debug feature is disabled.

- `#define AUTO_SWITCH_PORT` and `#define AUTO_SWITCH_PIN`

  These define statement can be used to set the mapping between the debug Auto-switch signal (see *Section 5.1.3*) and the debug pins (see *Section 5.1.9*).

  The first define statement specifies the test pin by replacing the x character in the DEBUGx_PORT string by the proper number. For instance if the test pin used is pin 0, set DEBUG0_PORT.

  The second define statement also specifies the debug pin by replacing the x character of the DEBUGx_PIN string with the same number as in the first define statement. This setting is not used if the debug feature is disabled.

- `#define PWM_ON_SW_PORT` and `#define PWM_ON_SW_PIN`

  These define statement define the mapping between the BEMF-sampling-during-on-time debug signal (see *Section 5.1.3*) and the debug pins (see *Section 5.1.9*).

  The first define statement specifies the debug pin by replacing the x character in the DEBUGx_PORT string by the proper number. For instance if the debug pin used is the pin 0 set DEBUG0_PORT.

  The second define also specifies the debug pin used by replacing the x character of the DEBUGx_PIN string by the same number of the first define statement. This setting is not used if the debug feature is disabled.

5.1.9 Port pins definition parameters (MC_stm8s_port_param.h)

The `MC_stm8s_port_param.h` header file contains the define statements related to the definitions of the pins and ports used for the motor control related signals:

- `#define DEBUGx_PORT` and `#define DEBUGx_PIN`

  When the firmware runs on a customized hardware, these define statements can be used to configure the ports and pins used for the debug signals.

  The first define statement specifies the port: replace the x character in the GPIOx string by the proper letter. For instance, set GPIOH if port H is used.

  The second define statement specifies the pin: replace the x character in the GPIO_PIN_x string with the proper number. For instance, set GPIO_PIN_1 if pin 1 is used.

- `#define KEY_UP_PORT, #define KEY_UP_BIT, #define KEY_DOWN_PORT, #define KEY_DOWN_BIT, #define KEY_LEFT_PORT, #define KEY_LEFT_BIT, #define KEY_RIGHT_PORT, #define KEY_RIGHT_BIT, #define KEY_UP_PORT, #define KEY_UP_BIT, #define USER_BUTTON_PORT, #define USER_BUTTON_BIT`

  When the firmware runs on a customized hardware, these define statements can be used to configure the ports and pins used for the user interface input signals (joystick and button).

  Define the port by setting the xxx_PORT define statements to GPIOx, where x specifies the port. For instance, set GPIOH if port H is used.

  Define the pin by setting the xxx_BIT define statements to GPIO_PIN_x, where x with specifies the pin. For instance, set GPIO_PIN_1 if pin 1 is used.

- `#define DISSIPATIVE_BRAKE_PORT, #define DISSIPATIVE_BRAKE_BIT`

  When the firmware runs on a customized hardware, these define statements can be used to configure the port and pin used for the dissipative brake signal (see *Section 4.3.4*).

  In the first define statement, set the port by replacing the x character in the GPIOx string with the proper letter. For instance set GPIOH if port H is used.

  In the second define statement, specify the pin by replacing the x character in the GPIO_PIN_x string with the proper number. For instance set GPIO_PIN_1 if pin 1 is used.

5.1.10 Hall parameter microcontroller interfaces (MC_stm8s_hall_param.h)

The `MC_stm8s_hall_param.h` header file contains the define statement related to the definitions of the pins and ports used to configure the Hall sensors. The pins are related to TIM2 inputs.

- `#define TIM2_CH3_REMAP`

  This define statement allows to remap TIM2 channel 3.

- `#define HALL_FILTER`

  This define statement configures the filter applied to the Input capture filter used for the Hall sensor signal. This filter is used to avoid false commutations caused by noise glitches. *Table 13* gives the values of the allowed filters.

Table 13. Hall sensor filters

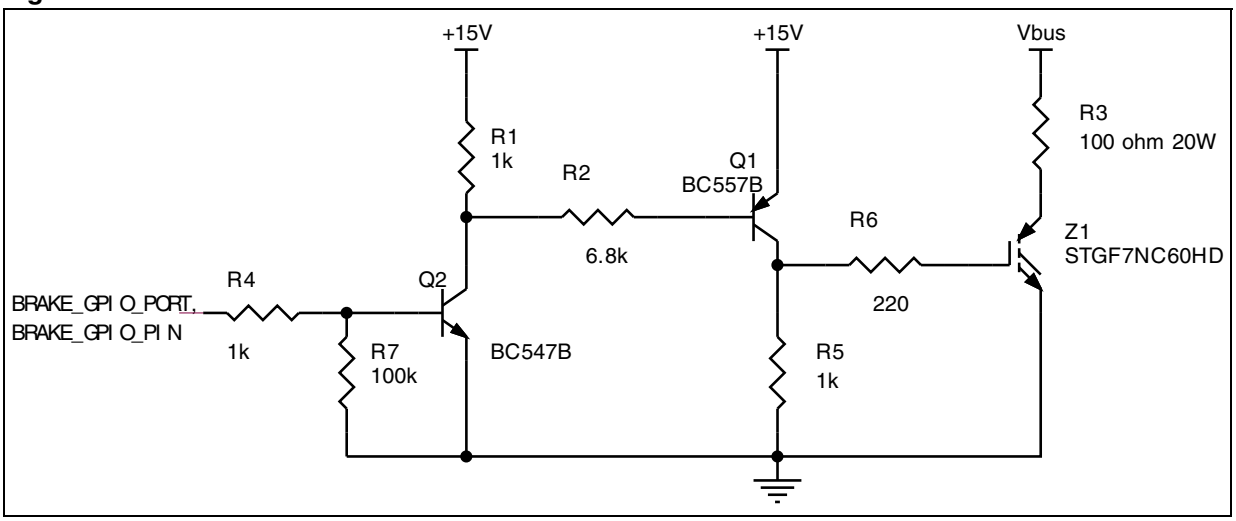
#define HALL_FILTER value	Filter for F_CPU= 16 MHz (in µs)	Filter for F_CPU = 24 Mhz (in µs)
HALL_FILTER_NOFILTER	0.0625	0.0417
HALL_FILTER_F_N2	0.1250	0.0833
HALL_FILTER_F_N4	0.2500	0.1667
HALL_FILTER_F_N8	0.5000	0.3333
HALL_FILTER_F2_N6	0.7500	0.5000
HALL_FILTER_F2_N8	1.0000	0.6667
HALL_FILTER_F4_N6	1.5000	1.0000
HALL_FILTER_F4_N8	2.0000	1.3333
HALL_FILTER_F8_N6	3.0000	2.0000
HALL_FILTER_F8_N8	4.0000	2.6667
HALL_FILTER_F16_N5	5.0000	3.3333
HALL_FILTER_F16_N6	6.0000	4.0000
HALL_FILTER_F16_N8	8.0000	5.3333
HALL_FILTER_F32_N5	10.0000	6.6667
HALL_FILTER_F32_N6	12.0000	8.0000
HALL_FILTER_F32_N8	16.0000	10.67

5.2 Setting up the system when using a brake resistor

To make the MB459 board suitable for the management of a brake resistor, additional components must be soldered on its wrapping area.

Figure 49 gives an example of circuit used for brake hardware implementation.

Figure 49. Brake resistor circuit



1. The resistor should be carefully dimensioned in terms of both resistance and sustainable power.
2. In the MB459B the pin 23 of the MC connector (J7) that carries the signal for brake implementation is positioned close to the wrapping area.

6 Library functions

6.1 Function description conventions

Functions are described in the format given below:

Synopsis	Lists the prototype declarations.
Description	Describes the functions specifically with a brief explanation of how they are executed.
Input	Gives the format and units.
Returns	Gives the value returned by the function, including when an input value is out of range or an error code is returned.
Note	Indicates the limits of the function or specific requirements that must be taken into account before implementation.

Some of these sections may not be included if not applicable (for example, no parameters or obvious use).

6.2 Modules description

The firmware is composed of a set of modules that are logically subdivided in three groups:

- High level motor control (MC) modules
- Low level MC modules
- Standard library

The last two groups contain functions related to the microcontroller while the first group contains hardware independent functions as described in *Section 4.2*.

6.2.1 High level MC modules

The high level MC modules are stored under the folder **MC_FWLIB_SCALAR**. They are composed if the following modules:

- `MC_BLDC_Drive`: this module contains all the functions related to the electrical drive and engine control (see *Section : MC_BLDC_Drive.c module*).
- `MC_BLDC_Motor`: this module is the “holder” of the drive structure. It contains all the function used to interact (retrieve or set parameters) directly with the BLDC drive structure, or to provide the reference of that structure for other modules.
- `MC_BLDC_User_Interface`: this module is the “holder” of the user interface specific to the BLDC drive. It contains a function used to provide the reference of this structure for other modules. The interaction with this structure is managed also by `MC_User_Interface.c`.
- `MC_User_Interface`: this module is used to manage the user interface. The user interface has been implemented using a joystick, a button, and an LCD display (see

Section 3.6). It interacts with MC_keys and MC_display modules. The user interface is adapted to the drive by the MC_BLDC_User_Interface module.

- MC_dev: this module makes the interface between high level and low level modules. For instance it initializes the low level modules by calling each specific initialization functions (dev_clkInit, dev_portInit, etc...).
- MC_display: this module manages the display of the information on the 15 rows x 2 lines LCD. This module is developed over the low level virtual I/Os functions (see *Section 4.2*).
- MC_Keys: this module manages the button and joystick (4 directions plus a center button). This module is developed over the low level "virtual I/Os" functions (see *Section 4.2*).
- MC_pid_regulators: this module manages all the application regulators. They can be either PID or PI regulators. It is used to instantiate a regulator structure and to execute it.
- MC_StateMachine: this module manages the main application state machine through the StateMachineExec function that is used to execute the state machine.
- MC_vtimer: this module is used to manage the virtual timers as explained in *Section 4.5.2*.
- Main: this is the main application firmware module. It is used to execute the state machine in a infinite loop.

Note: The exhaustive description of the high level MC modules is out of the scope of this user manual. Only the functions of the MC_BLDC_Drive and MC_vtimer modules will be described in details.

MC_BLDC_Drive.c module

The high level BLDC drive module manages the calls to the state machine functions, and interacts with the low level modules through the virtual registers and the drive structure.

The MC_BLDC_Drive.c functions are listed in the MC_drive.h header file:

- driveInit
- driveIdle
- driveStartUpInit
- driveStartUp
- driveRun
- driveStop
- driveWait
- driveFault
- BLDC_Drive
- GetSpeed_01HZ
- BLDCDelayCoefComputation

driveInit

Synopsis void driveInit(pvdev_device_t pdevice);

Description This function initializes local pointers to the virtual registers, drive structure, and the other variables of interest required for the other module functions.

The pointer to the drive structure is retrieved using the Get_BLDC_Struct function of `MC_BLDC_Motor` module.

The function initializes the low level drive module by calling the dev_driveInit(pdevice) function.

The other variables of interest are:
- Hertz to RPM conversion factor based on motor poles pairs,
- local pointer of speed PID structure,
- sampling time duration.

It also starts the BLDC_CONTROL_TIMER virtual timer which calls the BLDC_Drive function every sampling time period (see *Section 4.5*).

Input pdevice: pointer to a structure containing virtual registers

Returns None.

Note It is called by the relative state machine function.

driveIdle

Synopsis void driveIdle(void);

Description The purpose of this function is to call the low level function relative to this state and to update the local state variable DriveState.

Input None.

Returns None.

Note It is called by the relative state machine function.

driveStartUpInit

Synopsis MC_FuncRetVal_t driveStartUpInit(void);

Description The purpose of this function is to reset the local drive variables, to update the local state variable `DriveState,` and to call the low level function relative to this state.

The local drive variable are:
- Validation of the measured speed flag (bValidatedMeasuredSpeed),
- Speed PID integral sum,
- Drivestatus: this variable is used to report a fault condition coming from the BLDC_Drive function.

Input	None.
Returns	The returned value is the one coming from the low level function: – FUNCTION_RUNNING: no state change is required by this function, – FUNCTION_ENDED: the state change is requested by this function, – FUNCTION_ERROR: an error condition occurs and must be reported.
Note	It is called by the relative state machine function.

driveStartUp

Synopsis	MC_FuncRetVal_t driveStartUp(void);
Description	The purpose of this function is to call the low level function relative to this state and to update the local state variable `DriveState`.
Input	None.
Returns	The returned value is the one coming from the low level function: – FUNCTION_RUNNING: no state change is required by this function, – FUNCTION_ENDED: the state change is requested by this function, – FUNCTION_ERROR: an error condition occurs and must be reported.
Note	It is called by the relative state machine function.

driveRun

Synopsis	MC_FuncRetVal_t driveRun(void);
Description	The purpose of this function is to call the low level function relative to this state and to update the local state variable `DriveState`. The DriveStatus is checked to detect fault condition coming from the low level functions. If this condition is detected the return value is FUNCTION_ERROR otherwise the returned value is the one coming from the low level function.
Input	None.
Returns	– FUNCTION_RUNNING: no state change is required by this function, – FUNCTION_ENDED: the state change is requested by this function, – FUNCTION_ERROR: an error condition occurs and must be reported.
Note	It is called by the relative state machine function.

driveStop

Synopsis MC_FuncRetVal_t driveStartUp(void);

Description The purpose of this function is to call the low level function relative to this state and to update the local state variable `DriveState`.

Input None.

Returns The returned value is the one coming from the low level function:
– FUNCTION_RUNNING: no state change is required by this function,
– FUNCTION_ENDED: the state change is requested by this function,
– FUNCTION_ERROR: an error condition occurs and must be reported.

Note It is called by the relative state machine function.

driveWait

Synopsis MC_FuncRetVal_t driveStartUp(void);

Description The purpose of this function is to call the low level function relative to this state and to update the local state variable `DriveState`.

Input None.

Returns The returned value is the one coming from the low level function:
– FUNCTION_RUNNING: no state change is required by this function,
– FUNCTION_ENDED: the state change is requested by this function,
– FUNCTION_ERROR: an error condition occurs and must be reported.

Note It is called by the relative state machine function.

driveFault

Synopsis MC_FuncRetVal_t driveStartUp(void);

Description The purpose of this function is to call the low level function relative to this state and to update the local state variable `DriveState`.

Input None.

Returns FUNCTION_ENDED is always returned indicating that a state change is requested.

Note It is called by the relative state machine function.

BLDC_Drive

Synopsis void BLDC_Drive(void);

Description The purpose of this function is to:
- Update the speed measurement calling the GetSpeed_01HZ function,
- Validate it,
- Convert it into rpm,
- Manage the rotor direction based on the sign of the target speed,
- Call the delay computation BLDCDelayCoefComputation function if required,
- Execute the speed regulation if required,
- Refresh the control variables
- Update the DAC values if required.

Input None.

Returns None.

Note This function is automatically called by the BLDC_CONTROL_TIMER virtual timer at each sampling time period (see *Section 4.5*).

GetSpeed_01HZ

Synopsis u16 GetSpeed_01HZ(void);

Description The purpose of this function is to compute the measured rotor speed using the values coming from the low level function. These values are stored inside the virtual register of the related sensor (see *Section 4.3* and *Table 4*).

Input None.

Returns Measured electrical rotor speed value express in tenth of Hz.

Note The value returned by this function is not already validated.

BLDCDelayCoefComputation

Synopsis void BLDCDelayCoefComputation(u16 Motor_Frequency);

Description The purpose of this function is to compute the delay coefficient accordingly the predefined curve (see *Section 2.6*).

Input Mechanical rotor speed express in rpm.

Returns None.

Note The values computed are stored in the drive structure.

MC_vtimer modules

vtimer_SetTimer

Synopsis void vtimer_SetTimer(VtimerName_t name,timer_res_t msec,void* pCallback);

Description The purpose of this function is to initialize the virtual timer.

Input `name` is the reference to the virtual timer to be initialized. It can be a mnemonic or a relative number.

`msec` is the duration of the virtual timer expressed in milliseconds.

`pCallBack` is the pointer to the function to be executed at the end of the counting. It must be set to '0' when the Virtual timer operates in polling mode.

Returns None.

vtimer_TimerElapsed

Synopsis u8 vtimer_TimerElapsed(VtimerName_t name);

Description The purpose of this function is to check the "end of counting" of a virtual timer operating in polling mode.

Input Name is the reference to the virtual timer to be checked. It can be a mnemonic or a relative number.

Returns '0' if the time interval has not elapsed

'1' at the end of the counting of the requested virtual timer.

vtimer_KillTimer

Synopsis void vtimer_KillTimer(VtimerName_t name);

Description This function stops a Virtual timer operating in automatic mode. An "automatic" Virtual timer can be used to call a function at regular time intervals. To do this inside the called function, the Virtual timer must be reinitialized. The vtimer_KillTimer function is used to exit from this loop.

Input Name is the reference to the virtual timer to be stopped. It can be a mnemonic or a relative number.

Returns None.

6.2.2 Low level MC modules

The low level MC modules are stored in the folder `STM8_MC_FRAMEWORK`. They are composed of the following modules:

- `MC_stm8s_BLDC_drive`: this module contains all the functions related to the low level electrical drive and engine control.
- `MC_stm8s_BLDC_it`: this module contains all the interrupt service routines defined inside the interrupt vector, available for the application, and not related to motor control drive. The interrupt service routines used by the motor control firmware are defined inside each module.
- `MC_stm8s_clk`: this module sets the microcontroller clock.
- `MC_stm8s_DAC`: this module manages the digital to analog function implemented for debugging purposes. It uses TIM3 as described in *Section A.1*.
- `MC_stm8s_display`: this module contains the low level functions that interact with the LCD display. This module has been developed on top of `MC_stm8s_lcd` module. The exported function is dev_displayInit used to configure the hardware for the LCD display, dev_displayClear that clears the LCD screen, dev_displayFlush that outputs on the LCD the data already formatted by `MC_display module`, and dev_displayPrintch used to refresh the cursor.
- `MC_stm8s_keys`: this module is used only to initialize the hardware (dev_keysInit) since the low level functions that manage the keys are implemented inside the `vdev_ios`.
- `MC_stm8s_port`: this module is used only to initialize the GPIOs of the microcontroller that are used by the application.
- `MC_stm8s_vtimer`: this module is used to manage the low level virtual timers. It is used only for the initialization of the hardware (dev_vtimerInit), and contains the TIM4 interrupt service routine.
- `vdev_ios`: this module is used to manage the low level input/output functionality (see *Table 5: Virtual I/Os*).

Appendix A Additional information

A.1 DAC configuration

The DAC functionality is implemented by using two (PD2 and PD0 pins) TIM3 output compare channels, and by modulating the duty cycle of the generated 62.5 kHz PWM signal. To properly filter the generated signals without introducing important delays on the waveforms, it is recommended to use a appropriate first order low-pass filter (e.g. with a 1 kΩ resistor and a 33 nF capacitor). The two DAC outputs are used to monitor the measured rotor speed and the control variable (controller output).

The DAC outputs can also be used to monitor two user-defined variables such as user_var1 or user_var2 (see example below). The `MC_BLDC_Drive.c` file must be modified as follows:

```
#ifdef DAC_FUNCTIONALITY
  dev_DACUpdateValues(DAC_CH_1,(u8)(user_var1));
  dev_DACUpdateValues(DAC_CH_2,(u8)(user_var2));
#endif
```

Assuming that the DAC implemented has an 8-bit resolution, a suitable scaling factor should be applied to user defined variables.

Note: The DAC cannot be used together with the dissipative brake function.

See *Section 5.1.5* for details on how to enable the DAC.

A.2 Motor control related CPU load

Table 14 gives the percentage of the microcontroller workload required by the BLDC firmware for the specified configuration.

Table 14. Workload

Configuration	Workload
Sensorless closed loop voltage mode	32%
Sensorless closed loop current mode	27%
Sensor closed loop current mode	16.6%

A.3 STM8 motor control builder GUI

The STM8 motor control builder GUI is not part of the motor control kit. Please check http://www.st.com/mcu/inchtml-pages-stm8.html for availability.

Revision history

Table 15. Document revision history

Date	Revision	Changes
25-Jun-2009	1	Initial release.

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZED ST REPRESENTATIVE, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER’S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com