UM0712
User manual

STM8S three-phase AC induction motor control software library V1.0

Introduction

This user manual describes the alternate current induction motor (ACIM) scalar software library developed for STM8S microcontrollers.

These 8-bit, ST microcontrollers (STM8S) come with a set of peripherals that make them suitable for performing both PM and AC induction motor scalar control.

The present document describes the STM8S software library developed to control induction motors in open loop or speed-closed loop depending whether they are equipped with a tachogenerator speed sensor or not. The control of the brushless direct current (BLDC) motor, equipped with three Hall sensors or sensorless, in six-step mode is described in UM0708.

The ACIM motor software library is made of several C modules and is fitted with STVD workspaces. It is used to quickly evaluate both the MCU and the available tools. When it is used with the STM8/128-MCKIT motor control starter kit and an AC induction motor, a motor can be made to run in a very short time. The ACIM library also eliminates the need for time-consuming development of low-level drive and speed regulation algorithms by providing ready-to-use functions that allow the user to concentrate on the application layer.

A prerequisite for using this library is basic knowledge of C programming, AC motor drives and power inverter hardware. In-depth know-how of STM8S 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 ACIM software library V1.0 features

ACIM voltage mode scalar controls

- Speed open loop
  - Adjustable target rotor speed, V/f ratio, and slip speed via the user interface (UI)
- Speed open loop and load compensation
  - Adjustable target rotor speed via the UI
  - Varied V/f ratio and slip speed according to the characteristic torque curve of the load.
- Speed open loop and load compensation, tachogenerator sensing:
  - Adjustable target rotor speed via the UI
  - Slip speed varying according to the characteristic torque curve of the load
  - Rotor speed checked to validate startup and max/min run speed
- Speed closed loop, V/f and slip control:
  - Adjustable target rotor speed via the UI
  - Rotor speed measured and provided as feedback of the speed loop control
  - Optimized dynamics drive
  - Parameters tuning mode (proportional-integral-derivative regulator (PID) gains, V/f ratio, startup V/f ratio, startup slip).
- Speed closed loop, maximum torque per ampere (MTPA) control:
  - Adjustable target rotor speed via the UI
  - Rotor speed measured and provided as feedback of the speed loop control
  - Optimized efficiency drive
  - Parameter tuning mode (PID gains, V/f ratio, optimum slip, startup V/f ratio, startup slip).

Additional features

- Three-phase center-aligned PWM sine waves and third harmonics synthesis, adjustable switching frequency, dead time, output refresh rate.
- DC bus voltage measurement and ripple compensation
- Automatic drive adaptation to AC mains voltage
- Dead time effect compensation
- DC bus brake resistor management
- Heatsink temperature measurement
- Fault handling for overcurrent (shunt resistor network required), DC bus overvoltage/undervoltage, heatsink overtemperature.
- The UI is provided through LCD and joystick
- Two-channel, virtual DAC functionality, for real-time tracing of software variables
- Firmware is compatible with STM8s motor control builder GUI

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.

Library source code is released free of charge if used in the final application based on ST products.

Table 1. ROM and RAM requirements

Configuration	ROM (Kbytes)	RAM (Kbytes)
Speed open loop	6.13	0.18
Speed open loop, load compensation	6.32	0.18
Speed open loop, load compensation, tacho sensing	7.35	0.21
Speed closed loop, V/f + slip control/MTPA control	7.78	0.21

1.4 Development tools

The present 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 or use the ACIM software library.

For rapid implementation and evaluation of the ACIM library, it is recommended to acquire the STM8/128-MCKIT motor control starter kit.

1.4.1 Toolchains

This 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 (then select downloads).

A complete software package consists of:

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

The choice of the C toolchain is up to the user. However, only COSMIC 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**, **STM8_STVD_COSMIC_ACIM.stp**).

In addition, the STM8S motor control builder GUI (see *Section A.4*) allows these libraries to be customized through variables corresponding to the motor you are using. This makes the first utilization of the library significantly easier (see *Section 5: Designing an application using the ACIM software library*).

1.4.2 Programming tools

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 htt://www.st.com for information on STVP and RLink.

1.5 Reference documents

- User manual UM0708: STM8S three-phase AC induction motor software library V1.0.
- Reference manual RM0016: STM8S microcontroller family
- STM8S103xxx datasheet: Access line,16 MHz STM8S 8-bit MCU, up to 8 Kbytes Flash, data EEPROM,10-bit ADC, 3 timers, UART, SPI, I²C.
- STM8S105xx datasheet: Access line, 16 MHz STM8S 8-bit MCU, up to 32 Kbytes Flash, integrated EEPROM,10-bit ADC, timers, UART, SPI, I²C.
- STM8S20xxx datasheet: Performance line, 24 MHz STM8S 8-bit MCU, up to 128 Kbytes Flash, integrated EEPROM,10-bit ADC, timers, 2 UARTs, SPI, I²C, CAN.
- STM8S903K3 datasheet: 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.
- Programming manual 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 (MB631)
- User manual UM0709: STM8/128-MCKIT motor control starter kit
- User manual UM0483: STM32F103xx AC induction motor IFOC software library v2.0

2 Introduction to STM8S ACIM scalar control

2.1 Introduction to ACIM theory

An ACIM has a polyphase stator, identical to that of a synchronous machine. The windings can be wye- or star-connected and they are embedded in slots around the inside circumference. In a two-pole, three-phase motor, the stator windings are displaced 120 ° with respect to each other. The windings are symmetrical in that each of them has the same number of turns and the same resistance.

Conversely, the rotor structure characterizes two different kinds of machines: A *squirrel cage* induction machine and a *wound rotor* machine. In the squirrel cage induction machine, the rotor consists of a series of conducting bars, short-circuited at each end by conducting end rings. In the wound rotor machine, the rotor has a polyphase winding similar to that of the stator (with the same number of phases and poles) and its terminals abut insulated rings mounted on the shaft. Carbon brushes make these terminals available to the user.

Squirrel cage induction machines are, by far, the most commonly used type of electric motor thanks to their extreme simplicity and ruggedness. The great majority of induction machines are operated as motors.

Induction motors owe their name to their working principle that the rotor voltage, which produces the rotor current and magnetic field, is induced in the rotor windings.

The MMF (magneto-motive force) and the magnetic flux produced by a balanced set of three-phase steady-state currents, rotates about the air-gap at a mechanical speed of ω_{es}/p, where ω_{es} is the stator current frequency and p is the number of pole pairs.

If the rotor mechanical speed (ω_r) is different from the speed of this rotating air-gap flux, three-phase currents are *induced* in the short-circuited rotor windings.

The frequency (ω_{er}) of these currents is equal to the difference of the stator current frequency and the rotor speed, so that:

Equation 1

$$\omega_{er} = \omega_{es} - \omega_r \times p$$

Consequently, the rotor currents produce a rotor magnetic flux which rotates about the air-gap at a speed which, seen from the rotor is equal to that of rotor currents, but, seen from the stator reference frame is equal to ω_{es}/p. The electromagnetic torque is then produced due to the interaction of the two synchronous magnetic fields.

The difference between the air-gap flux electrical speed ($\omega_e = \omega_{es}$) and the electrical rotor speed is the so-called *slip speed* of the rotor, defined in *Equation 2*.

Equation 2

$$\omega_{se} = \omega_e - \omega_{rel}$$

Note: The air-gap flux electrical speed is also described as the synchronous speed.

Usually the slip is also expressed as a fraction of the synchronous speed (*Equation 3*).

Equation 3

$$s = (\omega_e - \omega_{rel}) / \omega_e$$

However, there is an upper limit to the motor's speed: If the rotor is running at synchronous speed (*Equation 4*) there is no induced voltage. For this reason, induction motors are also called asynchronous motors and the torque is defined as an *asynchronous torque*, because it is produced at any rotor speed except for the synchronous speed.

Equation 4

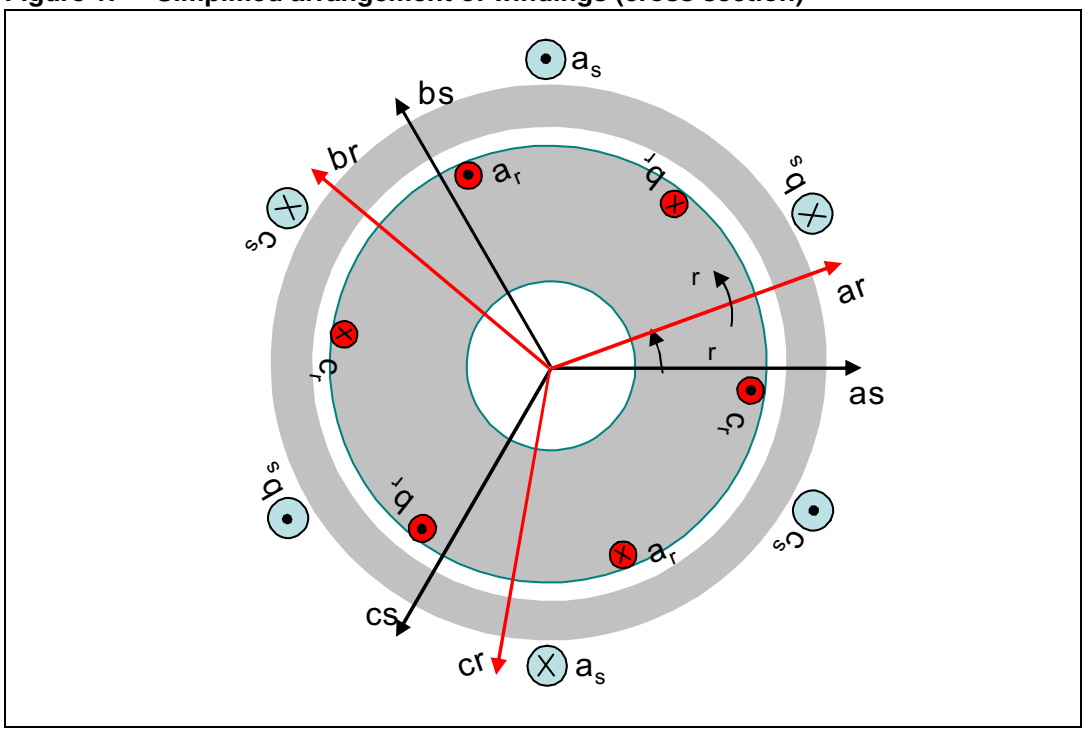
$$\omega_r = \omega_{es} / p$$

Consequently, s = 0

On the other hand, if the rotor is stationary (s = 1), rotor induced voltage has the same stator frequency. In this case, the motor behaves as a transformer with short-circuited secondary winding. All other working conditions fall somewhere between these two limits.

The winding arrangement of a 2-pole induction motor is shown in *Figure 1*.

Figure 1. Simplified arrangement of windings (cross section)

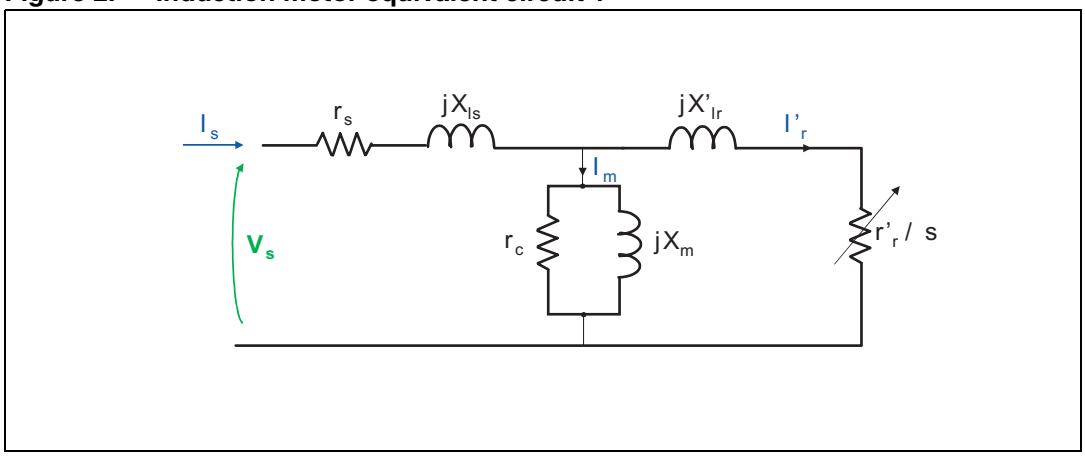


2.2 ACIM steady state electrical circuit

The equivalent electrical circuit of an induction motor is very similar to that of a transformer. Since rotor voltages and currents are induced in the same way, the only difference between the two is that in an induction motor, the secondary winding is short-circuited and the load, or working condition, is represented as a function of slip speed.

Figure 2 shows the equivalent T circuit of one phase of an induction motor (phase “a”). Voltages and currents of the remaining two phases are simply shifted in time by ±120 °. The motor is considered to be wye-connected.

Figure 2. Induction motor equivalent circuit 1



1. Legend: V_s is the line-to-neutral stator applied voltage
   I_s I the stator phase current
   I_m the magnetizing current
   I_r the referred (or equivalent) rotor current
   r_s is the stator resistance
   r'_r is the referred rotor resistance
   Xl_s is the stator leakage reactance
   Xl'_r is the referred rotor leakage reactance
   X_m is the magnetizing reactance
   R_c is the core-loss equivalent resistance

The rotor resistance and leakage inductance (and their referred values) are not directly measurable on squirrel cage motors. Specific measurements, such as no load and locked rotor tests, provide methods to indirectly obtain these parameters.

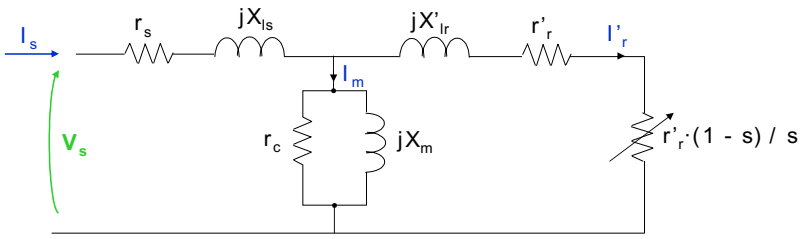
In the equivalent T circuit, the stator current is divided into two components: The magnetizing current I_m and the load current I'_r. The magnetizing current is required to create the air-gap flux which is produced by the combined effect of stator and rotor currents.

The magnetizing current (I_m) can be further divided into a component which is responsible for core-losses and a pure magnetizing current.

The load current (I'_r) is responsible for the rotor current. It is a function of the applied load and the working point condition. The combined effect of load and rotor resistance appears as the equivalent resistance r'_r/s.

The equivalent circuit can be redrawn to separate representation of the rotor resistance from the effect of applied load and working conditions (see *Figure 3*).

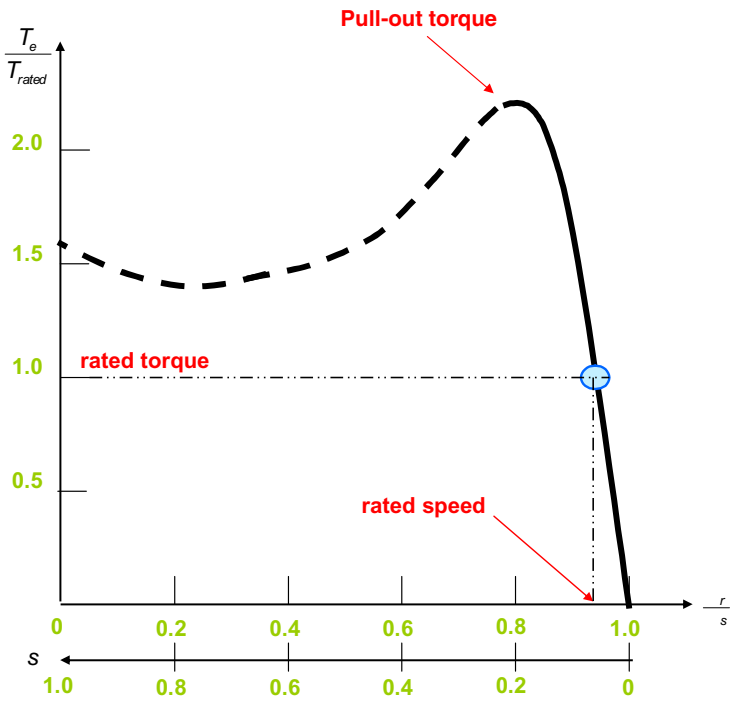
Figure 3. Induction motor equivalent circuit 2



2.3 Electromagnetic torque characteristic curve

The electromagnetic torque-speed characteristic curve is shown in *Figure 4* where V_s and ω_e are held constant at their nominal values.

Figure 4. Electromagnetic torque-speed characteristic

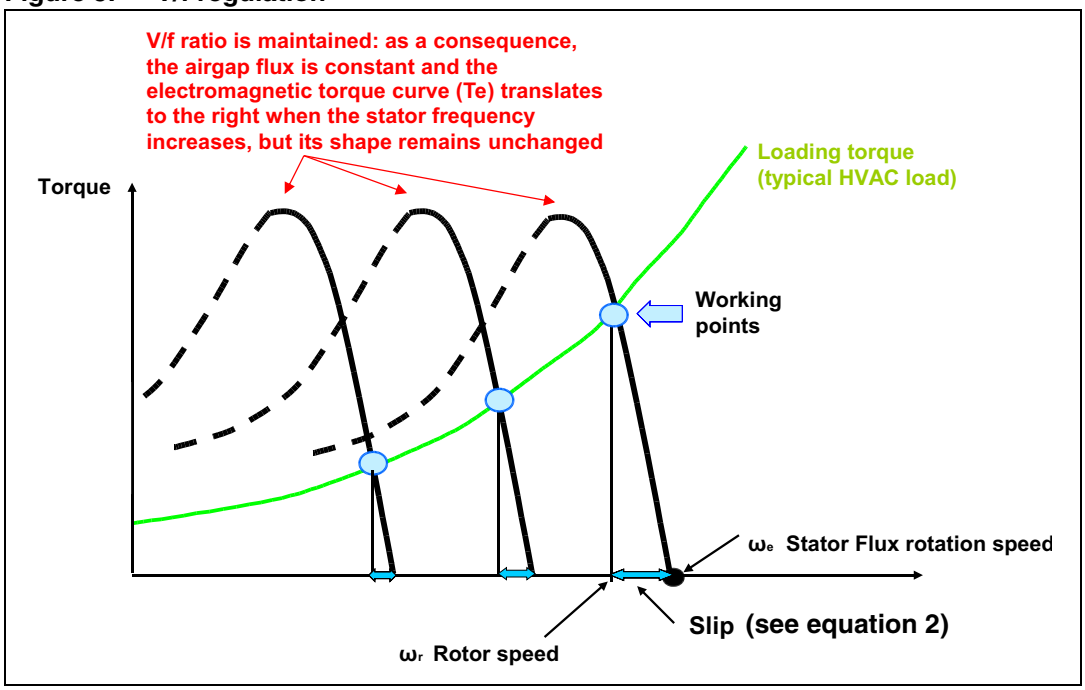


Speed is represented on a rated scale and also represented simultaneously as a function of the slip. Torque (T_e) is displayed as a fraction of its rated value.

The curve has three peculiar regions:

- The low-slip region at the right-hand side of the curve, characterized by a linear relation of T_e and I_r with slip and negligible inductive reactance. The normal operating range of an induction motor is inside this region, where the energy conversion efficiency is also optimal.
- In the moderate slip region, the inductive reactance is no longer negligible, thus the rotor current is less than proportional with slip. The displacement angle (the angle between the rotor and air-gap flux) departs from its optimum value of 90 °. The torque increases up to a maximum value, called pull-out torque, and then decreases. This region can be used just for short term overloaded operations.
- The third region, characterized by high-slip, is an unstable and inefficient working area.

Figure 5. V/f regulation



Starting with the induction motor equivalent circuit (*Figure 3*), it can be seen that, at steady state and within the low- and medium-slip regions, air-gap flux is directly controlled by the ratio of stator phase voltage (V_s) and stator electrical frequency (f).

Furthermore, under a constant V/f ratio (constant air-gap flux), the electromagnetic torque curve shifts to the right as the stator frequency is increased (*Figure 5*).

This concept gives rise to the drive strategy, whereby varying the frequency and the voltage applied through a DC-AC inverter, the motor is able to supply the nominal torque at nominal conditions of stator and rotor currents over its complete speed range.

The speed range is typically divided into two regions. In the first region (up to the nominal speed), the motor can produce its nominal torque. The second region (beyond the nominal speed) is characterized by constant power flow, where the torque decreases with inverse proportionality to the speed.

Depending on the application and the availability of a speed sensor, two main operating modes can be selected: Speed closed loop control (see *Section 2.4*) and speed open loop control (see *Section 2.5*).

Both operating methods are described as scalar controls in voltage mode (stator current amplitude is controlled at steady state through modulation of the applied voltage).

2.4 Speed closed loop control

Applications which demand good speed accuracy and/or fast responses to either load torque variation or speed reference variation, require a closed loop controller and a speed sensing device. The current ACIM firmware library (V1.0) includes a module whereby the motor speed can be read via a tachogenerator sensor.

To enable speed closed loop mode, see *Section 5.1.1: ACIM configuration file: MC_ACIM_conf.h*. Part of the drive is a negative feedback PID speed controller: the actual rotor speed is measured and compared with a reference speed to produce the error signal. This signal is processed by the controller to generate the command signal which keeps the speed error to a minimum.

This firmware library provides two closed loop control modes: V/F control and slip regulation (see *Section 2.4.1*) and MTPA (see *Section 2.4.2*).

2.4.1 V/f control and slip regulation

This control mode is designed to optimize the dynamic response of the system if the motor is inside the first speed region (constant torque).

To configure the firmware library accordingly, see *Speed closed loop mode and related define statements* in *Section 5.1.3*.

The operating principles and control scheme of V/f and slip regulation are shown in *Figure 6* and *Figure 7*. To prevent the inevitable delay due to the field circuit dynamic, the magnetizing flux is maintained at its nominal value by keeping the ratio between the frequency and the applied stator phase voltage constant.

Under such conditions, it is possible to demonstrate, with good approximation, that the electromagnetic torque is a function of the slip speed (see *Equation 2*).

Figure 6. V/f and slip regulation

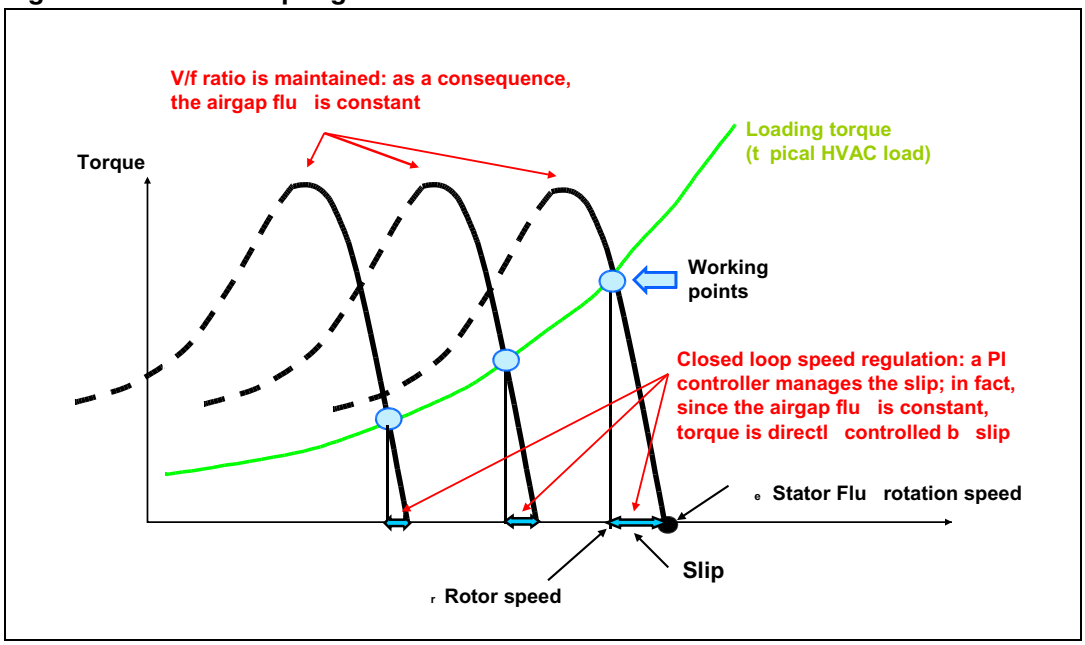
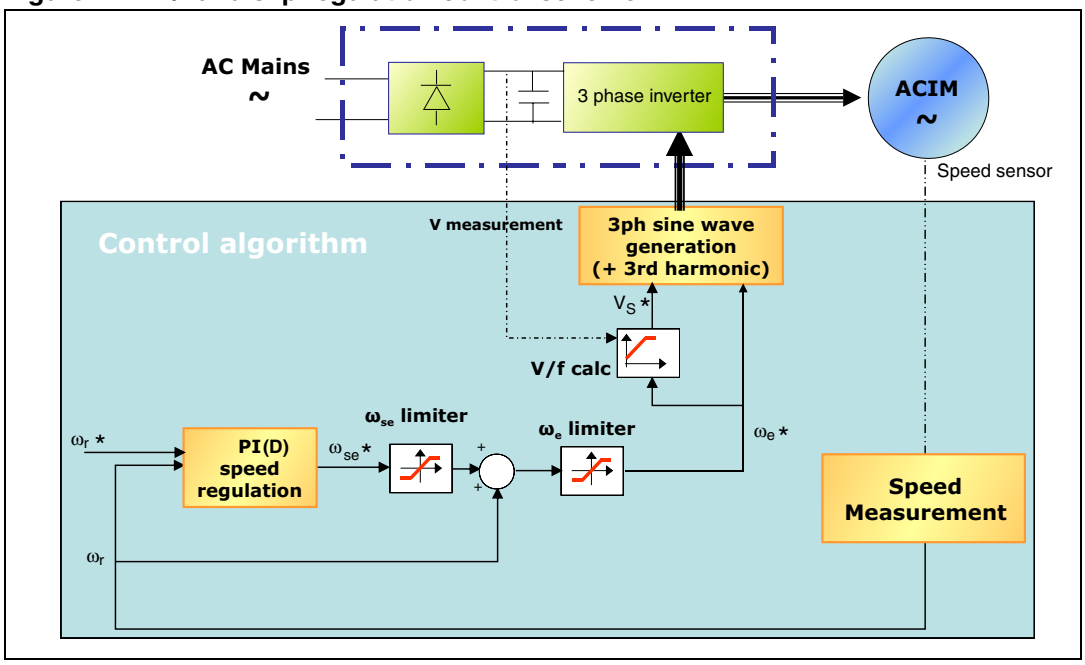


Figure 7. V/f and slip regulation control scheme



1. Legend: V_s^* = Voltage amplitude
   ω_e^* = Reference stator speed
   ω_{se}^* = Slip speed reference value
   ω_r^* = Target rotor speed
   ω_r = Measured rotor speed

Figure 7 shows that the output of the PID speed controller is a precise reference slip speed value (ω_{se}^*) which, added to the measured rotor speed (ω_r), is the reference stator speed (ω_e^*).

The PID speed controller can be tuned in real-time by using the LCD user interface (see *Figure 24* and *Section 3.3.2: Speed regulator parameters*).

Furthermore, under nominal flux conditions, stator current increases linearly with slip speed, therefore, the slip speed saturation block limits the stator current within its nominal value.

The resulting voltage amplitude and reference stator speed are applied to the motor by means of pulse width modulation of the three-phase inverter. This is implemented by the three-phase sine wave and third harmonic generation block (see *Section 2.7: Three-phase PWM sine wave and third harmonic generation*).

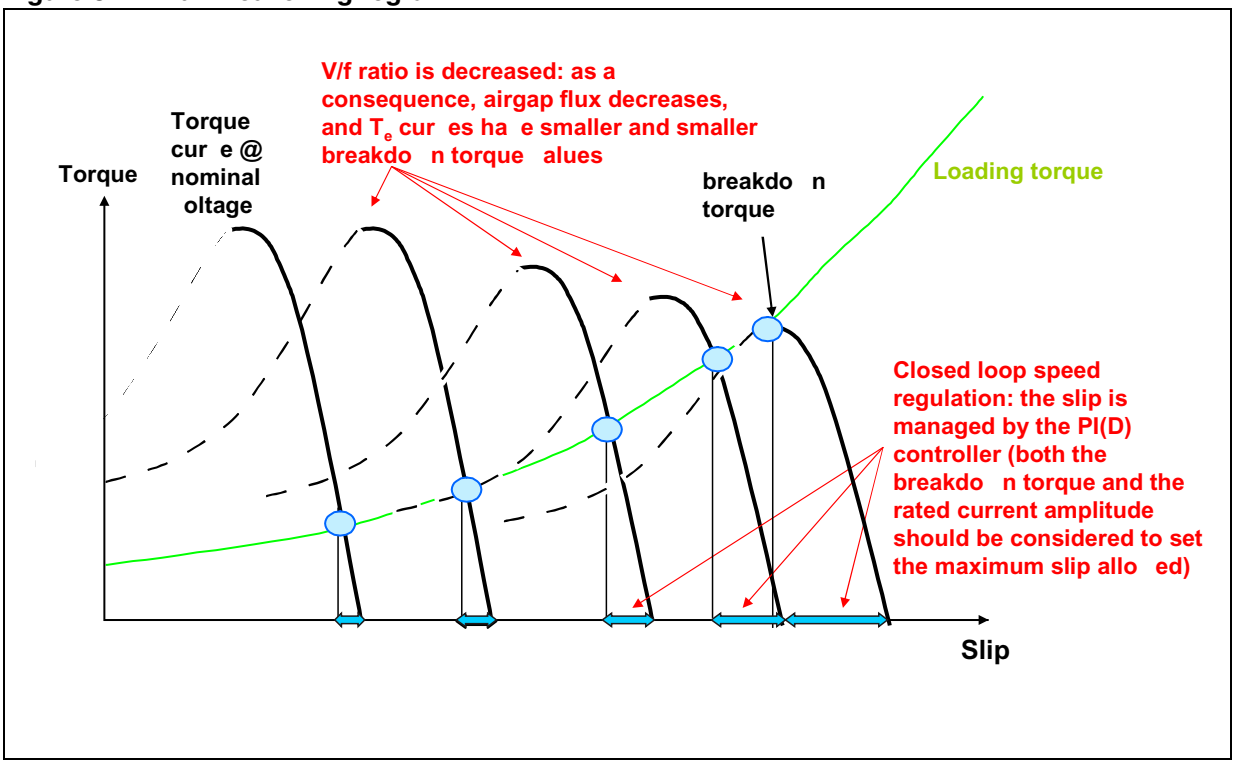
Since the V/f ratio is kept constant, the higher the stator frequency required to reach the target speed, the higher the applied stator phase voltage. At a certain point, under nominal load torque, a maximum inverter modulation index (nominal motor speed) is required. Following this, the power flow reaches its maximum.

Higher speeds can be attained but, at a cost of gradually reducing the magnetizing flux and hence the load applied. Such operating mode, called flux weakening, is executed in the second speed region (constant power).

In this second speed region (see *Figure 8*), keeping the stator phase voltage at its maximum value progressively increases the stator frequency results in reduced V/f ratios. Consequently, the air gap flux falls and the electromagnetic torque curves have progressively smaller breakdown torque values.

Simultaneously, taking into account that the magnetizing current (I_m) falls in proportion to the V/f ratio, the slip speed saturation value can be increased while respecting the motor nominal current and torque curve breakdown values.

Figure 8. Flux weakening region



2.4.2 Maximum torque per ampere (MTPA) control

MTPA control mode is designed to optimize system energy efficiency (see *Section A.3*, reference 1) as long as the motor is inside the first speed region (constant torque).

To configure the firmware library accordingly, see *Speed closed loop mode and related define statements* in *Section 5.1.3*.

MTPA mode aims at maximizing the ratio between the electromagnetic torque produced and copper losses.

By transforming the ACIM equations on a dq0 reference frame, synchronous with the rotor flux, it is possible to demonstrate that the electromagnetic torque (T_e) and the rotor flux slip frequency ($\omega_{s\lambda r}$) can be expressed as in *Equation 5* and *Equation 6* (see user manual UM0483: STM32F103xx AC induction motor IFOC software library v2.0).

Equation 5

$$T_e = \frac{3}{2} \overline{P} \frac{(L_m)^2}{L_r} \times i_{qs} \times i_{ds}$$

Equation 6

$$\omega_{s\lambda_r} = \frac{i_{qs}}{\tau_r \times i_{ds}}$$

Where:
L_m = magnetizing inductance
L_r = rotor inductance
τ_r = rotor electrical time constant
p = number of pole pairs
i_{qs} and i_{ds} = the two components of the transformed three-phase stator current system

At steady state, which is the validity domain of the scalar control, the rotor flux slip frequency is equal to the slip frequency (*Equation 7*).

Equation 7

$$\omega_{s\lambda_\gamma} = \omega_{se}$$

Moreover, given any instantaneous motor current amplitude (I_s), *Equation 8* holds true.

Equation 8

$$I_s = \sqrt{(i_{qs})^2 + (i_{ds})^2}$$

Under these conditions, electromagnetic torque is maximized for *Equation 9* and *Equation 10*.

Equation 9

$$i_{qs} = i_{ds}$$

Equation 10

$$\omega_{seopt} = \frac{1}{\tau_r}$$

Therefore, if the slip frequency is kept constant at the inverse of the rotor electrical time constant, the MTPA target is attained. In MTPA mode, once the slip frequency is constant, the control variable is the V/f ratio, which hence is the output of the PID speed controller.

MTPA mode and its control are shown in *Figure 9* and *Figure 10*.

The PID gains can be tuned in real-time, by using the LCD user interface (see *Figure 25* in *Section 3.3.2: Speed regulator parameters*).

Figure 9. MTPA mode strategy

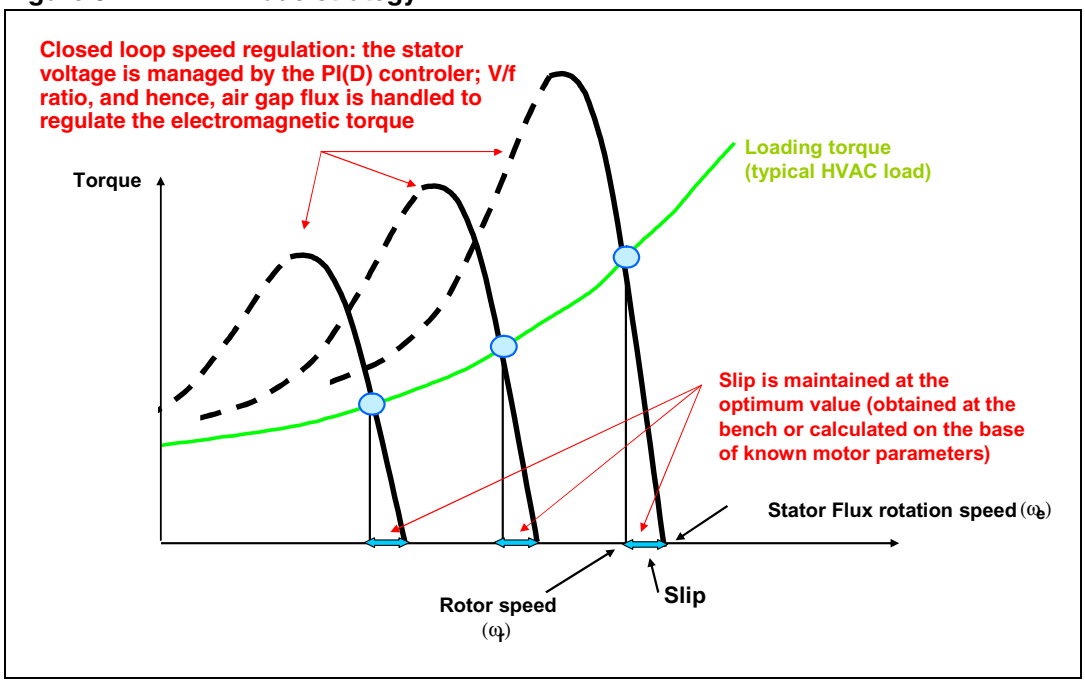
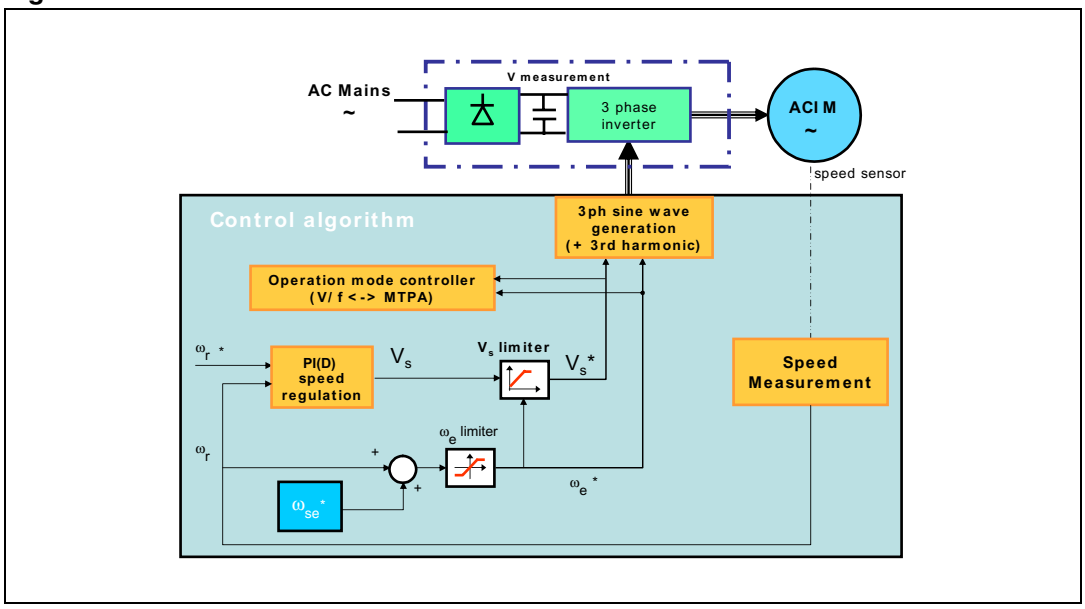


Figure 10. MTPA mode control scheme



1. Legend: V_s^* = Voltage amplitude
   ω_e^* = Reference stator speed
   ω_{se}^* = Slip speed reference value
   ω_r^* = Target rotor speed
   ω_r = Measured rotor speed

The MTPA mode strategy has a clear drawback in the necessary delay needed to build up the magnetizing flux in response to a required torque variation.

The magnetizing flux can be increased through variation of the V/f ratio, up to the point when it assumes its nominal value (it is not convenient to increase the magnetizing flux after this point because magnetic saturation of the stator iron occurs).

For these reasons, MTPA mode switches automatically to constant V/f and slip regulation mode described in *Section 2.4.1*. The PID gains of the speed controller used in the latter mode are taken into account and can be tuned in real-time via the LCD user interface (see *Figure 24* in *Section 3.3.2: Speed regulator parameters*).

However, while in constant V/f and slip regulation mode, if the required slip frequency is less than the optimum value ($\omega_{se\ opt}$), the PID speed controller switches back to MTPA mode (the “constant slip” area).

Such switching from one mode to another is the task of the “operation mode controller” (see *Figure 10*). The LCD user interface shows, in real-time, the current operating mode (see *Section 3.3.5: Control strategy parameters*).

The resulting voltage amplitude and reference stator speed are applied to the motor by means of pulse width modulation of the three-phase inverter. This is implemented by the three-phase sine wave and third harmonic generation block (see *Section 2.7: Three-phase PWM sine wave and third harmonic generation*).

2.5 Speed open loop control

In some applications, utilization of a speed sensor is not convenient and the load torque has a slight variation with speed. In such conditions, speed open loop control can be activated (see *Section 5.1.1: ACIM configuration file: MC_ACIM_conf.h*).

To configure the firmware library accordingly, see *Speed open loop mode and related define statements* in *Section 5.1.3*.

The drive consists of applying a constant V/f ratio, so as to supply the required magnetizing current, and a constant slip frequency.

Motor speed variation is attained by varying the stator frequency as a function of target speed and slip frequency according to *Equation 11*.

Equation 11

$$\omega_e = \omega_{target} + \omega_{se}$$

The V/f ratio and slip frequency applied can be modified in real-time by using the LCD user interface (see *Section 3.3.3: Drive strategy parameters*).

This firmware library provides two open loop control modes:

- Pure open loop mode (described above)
- Open loop and load compensation mode (see *Section 5.1.3: ACIM drive control define statements: MC_ACIM_Drive_Param.h*). This mode aims to compensate the speed error due to a known load variation with speed.

For tuning, debug purposes, or application requirements, the tachogenerator sensing module can be activated if a speed sensor is available. Therefore, the actual motor speed is displayed in the LCD user interface (`#define SPEED_OPEN_LOOP_TACHO_SENSING`, see *Section 5.1.1*)

2.5.1 Load compensation

For applications where the load torque has a known characteristic curve and the speed accuracy is not critical, speed open loop control plus load compensation can be chosen as a suitable drive.

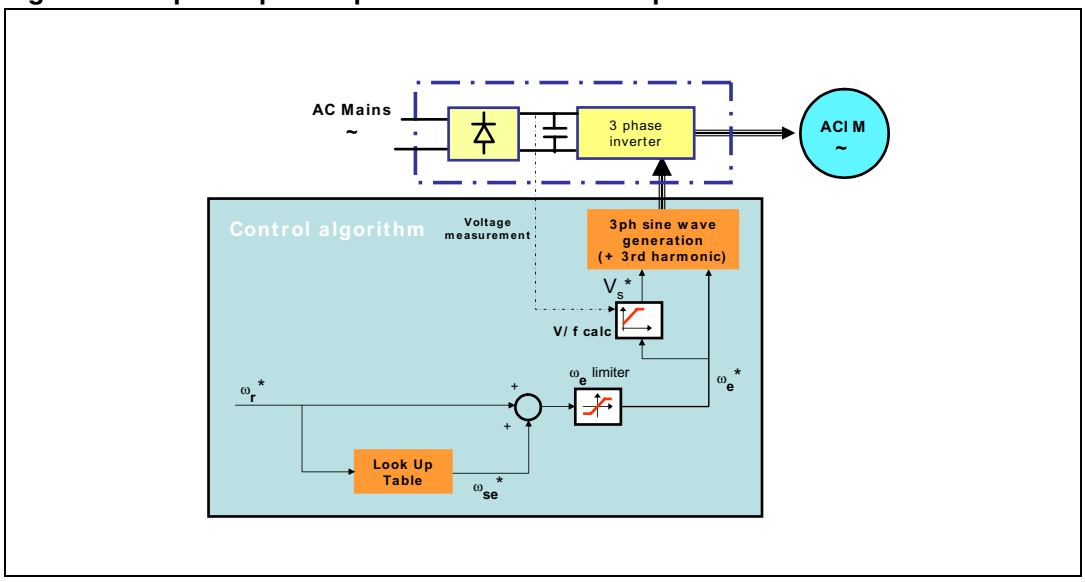
To configure the firmware library accordingly, see *Speed open loop mode and related define statements* in *Section 5.1.3*.

Speed open loop control with load compensation control scheme is shown in *Figure 11*.

The correct slip speed frequency for each target rotor speed is selected via a look up table which has been built offline by the STM8s motor control builder GUI (see *Section A.4*). The look up table is based on the specific load torque characteristic curve provided by the user.

The resulting reference stator speed (ω_e*) and the voltage amplitude (V_s*), calculated according to the settled V/f ratio, are applied to the motor by means of pulse width modulation of the three-phase inverter. This is implemented by the three-phase sine wave and third harmonic generation block (see *Section 2.7: Three-phase PWM sine wave and third harmonic generation*).

Figure 11. Speed open loop control with load compensation



1. Legend: V_s^* = Voltage amplitude
   ω_e^* = Reference stator speed
   ω_{se}^* = Slip speed reference value
   ω_r^* = Target rotor speed

2.6 Startup strategy

The closed loop startup strategy implemented is displayed in *Figure 12*.

A constant slip frequency, equal to that specified in define statement `STARTUP_SLIP` (see *Section 5.1.3: ACIM drive control define statements: MC_ACIM_Drive_Param.h*) is applied to the motor. The resulting stator frequency applied is given in *Equation 12*.

Equation 12

$$\omega_s = \omega_{se} + \omega_{r\_measured}$$

The stator voltage (V_s) increases linearly as a function of time as long as the relation below holds true:

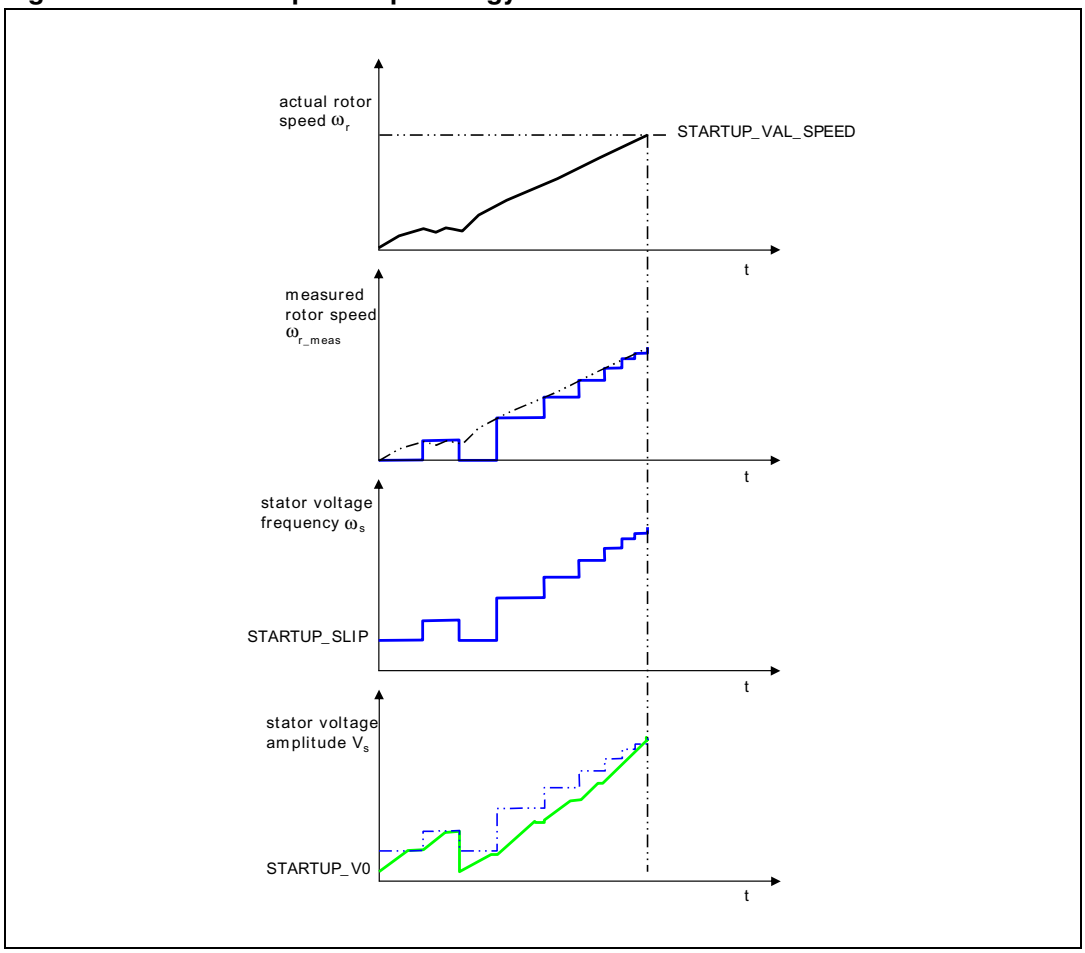
$$\frac{V_s}{\omega_s} < \text{STARTUP\_V\_F\_RATIO}$$

The voltage boost, required at low-motor frequencies to compensate for stator voltage drop, is specified through the define statement `STARTUP_V0` (see *Section 5.1.2: ACIM motor define statements: MC_ACIM_Motor_Param.h*).

As soon as the rotor measured speed is greater than STARTUP_VAL_SPEED (see *Startup phase related define statements* in *Section 5.1.3*), the drive switches from start to run state and the control schemes explained in *Section 2.4: Speed closed loop control* are executed.

If the validation speed explained above is not measured within a period defined by the define statement `STARTUP_DURATION`, an error message “startup failed” is generated and the state machine moves from START state to FAULT state.

Figure 12. Closed loop startup strategy



1. Legend: ω_r = Actual rotor speed
   ω_{r_meas} = Measured rotor speed
   V_s = Stator voltage amplitude

2.7 Three-phase PWM sine wave and third harmonic generation

To fulfill the basic AC induction motor voltage needs, the reference PWM modulating signal can be a pure sine wave (left graph of *Figure 13*). However, this kind of modulation makes poor usage of the DC bus voltage.

If we consider Vbus as the bus voltage after mains rectification, the maximum available voltage on a motor using a standard three-phase power inverter is around 86% of Vbus.

Equation 13

$$V_{phase-neutral} = V_{bus} / 2$$

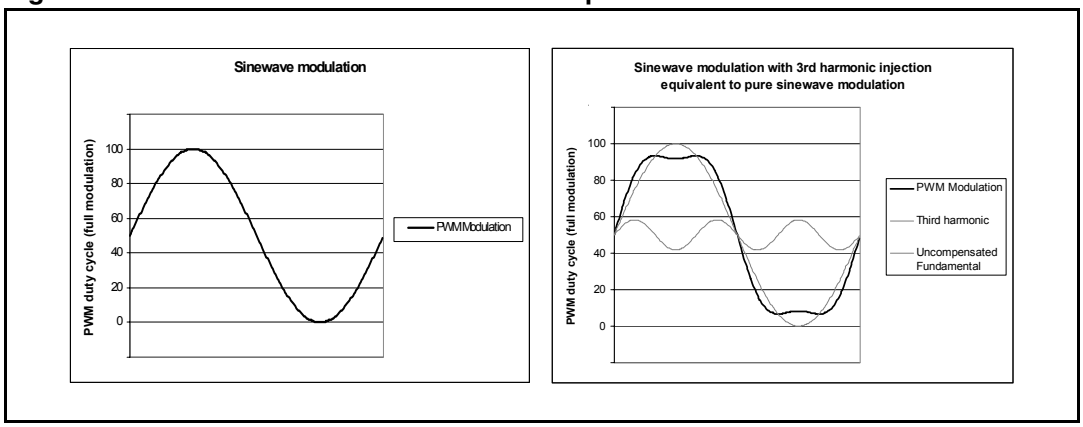
with

$$V_{neutral} = V_{bus} / 2$$

$$V_{phase-phase} = \sqrt{3} \times V_{pk} = (\sqrt{3}) / 2 \times V_{bus}$$

Adding a third harmonic modulation to the reference sine wave fundamental, decreases the overall amplitude of the resulting PWM modulation. PWM duty cycle reaches neither 0% nor 100% (right graph of *Figure 13*). This is due to the fact that the minimum of the third harmonic corresponds to the maximum of the fundamental and vice versa.

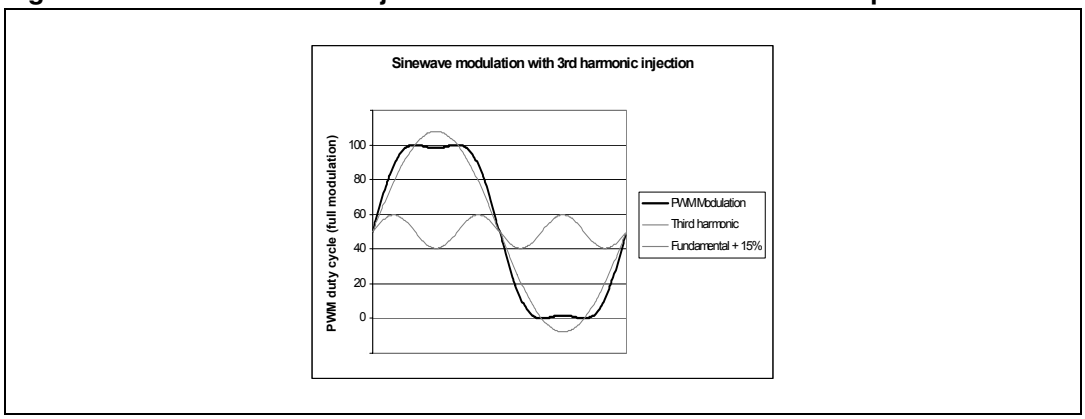
Figure 13. Pure sine wave modulation and equivalent with third harmonic added



Consequently, this allows the fundamental and the resulting third harmonic signal amplitude to be increased up to the point were the modulating signal reaches the DC bus limits (100% PWM modulation): See *Figure 14*.

By applying an appropriate coefficient to the third harmonic component, the fundamental amplitude can be further increased by 15 %.

Figure 14. Third harmonic injection with increased fundamental amplitude

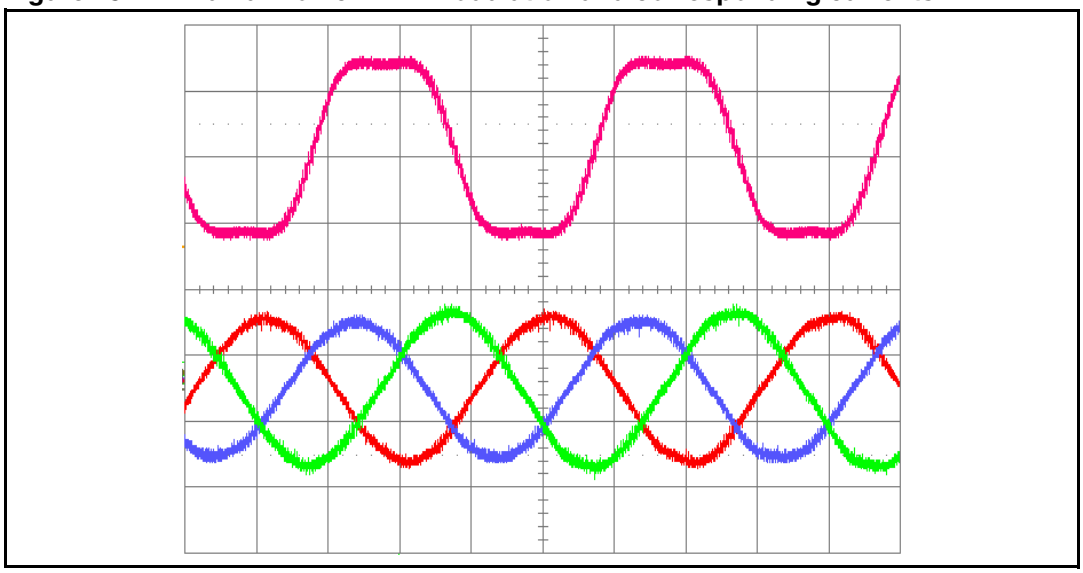


Finally, when considering phase to phase voltage on the motor, third harmonic components are mutually cancelled out (a 120 ° phase-shift on the fundamental corresponds to a 360 ° shift for the third harmonic). This results in the following:

- Sinusoidal voltage (and currents) on the motor, meaning no extra iron losses due to current harmonics.
- Phase to phase voltage which is 15% higher than with pure sine wave PWM modulation.

Figure 15 (top curve) shows the filtered PWM modulation on one of the three half-bridges. while *Figure 15* (bottom curve) shows the corresponding currents in the three motor phases.

Figure 15. Third harmonic PWM modulation and corresponding currents



To summarize, third harmonic injection allows:

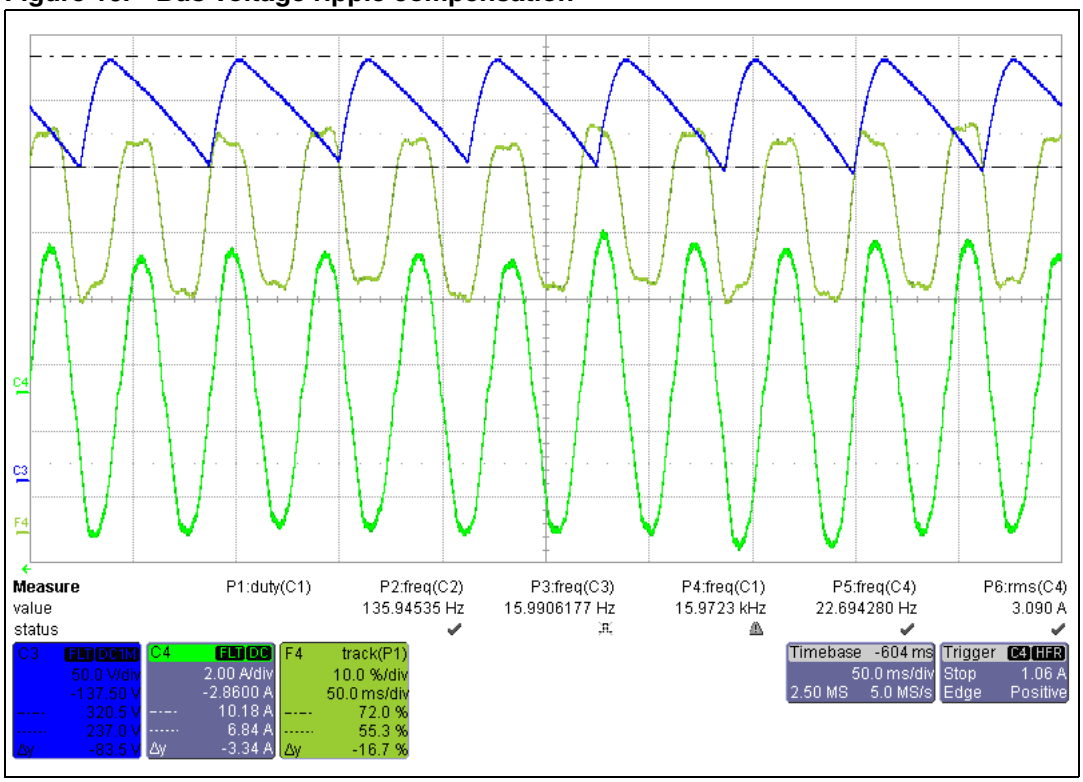
- A decrease in the diameter of the copper winding in the motor for a given power rating
- An increase in the current in the motor for a given frequency, therefore providing more output power.
- An increase in the maximum reachable speed for a given motor, as long as the mechanics (mainly ball-bearings) are suited for higher speed operations.

2.8 Bus voltage ripple cancellation

The motor control software can be configured to continuously measure the DC bus voltage (see *Section 5.1.6: Power stage define statements: MC_PowerStage_Param.h*). Since the sampling frequency is equal to the inverter PWM switching frequency, the DC bus ripple can be compensated with a very high resolution, thus generating clean phase voltages.

Figure 16 shows a power-demanding operating condition to highlight such functionality. C3 measures the DC bus, F4 is the duty cycle applied to a phase, and C4 is the phase current. The duty cycle applied is considerably distorted due to the very high oscillation of the DC voltage.

Figure 16. Bus voltage ripple compensation



3 Running the demonstration program

The ACIM motor control software library includes a demonstration program which allows a SELNI AC induction motor to be driven by the user in sensored mode. In this way a set of parameters can be changed via a user interface.

3.1 ACIM user interface

The ACIM user interface has been developed to display drive variables and to 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 2* for the list of joystick actions and conventions)
- A push button (KEY button)

Table 2. Joystick actions and 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

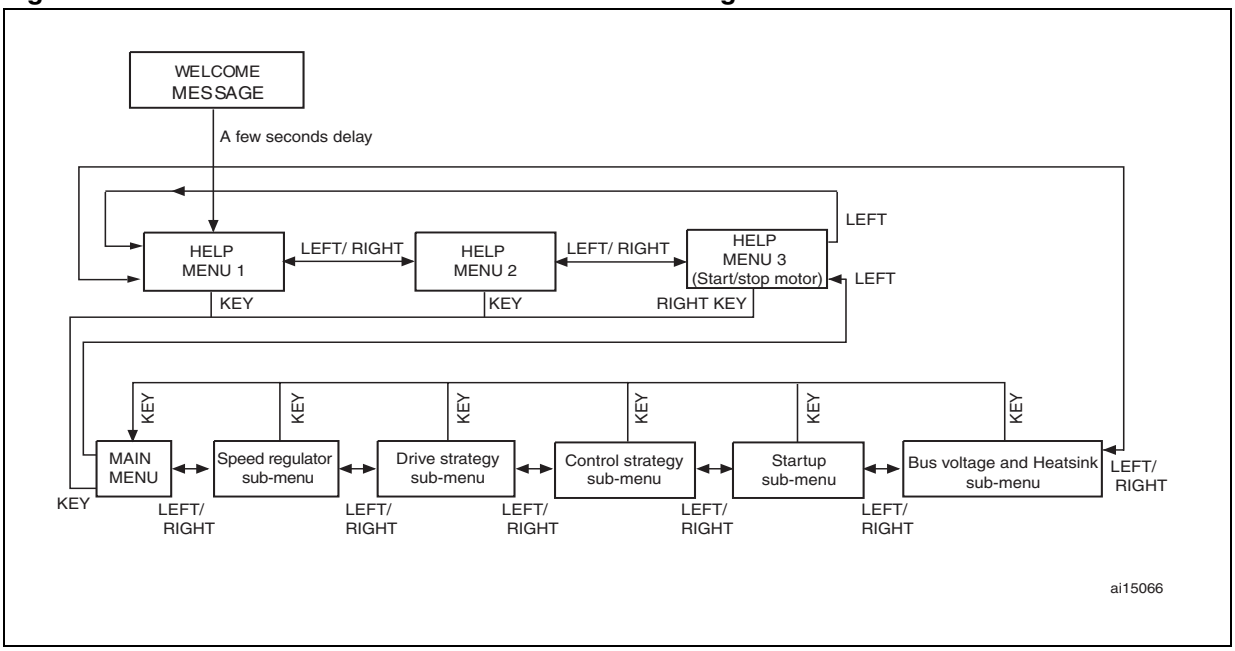
The demonstration program user interface is based on a circular navigation menu, with submenus, item selection and back capability.

Figure 17 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 17. ACIM user interface menu structure and navigation.

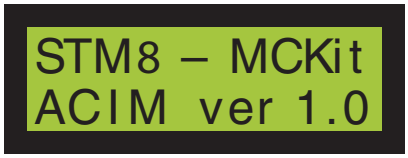


3.2 Getting started with the ACIM user interface

3.2.1 Welcome message

After the STM8/128-MCKIT motor control starter kit is powered on or reset, a welcome message is displayed on the LCD screen to inform the user about the firmware code loaded on the board. Refer to *Figure 18* for the ACIM drive welcome message.

Figure 18. ACIM drive welcome message

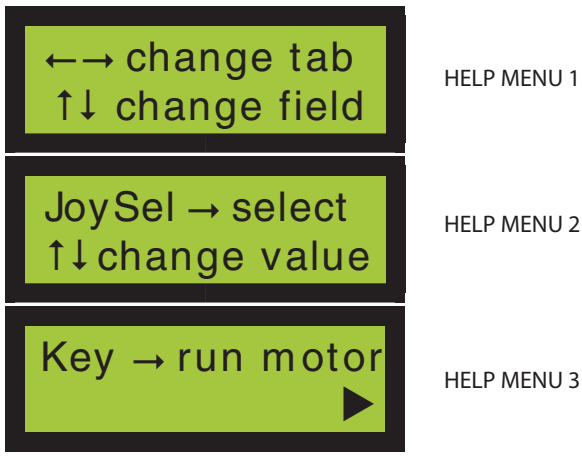


3.2.2 Help menus

After a few seconds, the LCD screen displays the first help message (see *Figure 19*). The user can navigate to the next help menu by pressing the joystick RIGHT. To go back to the previous help menu, press the joystick LEFT.

The KEY button can be pressed any time to start and stop the ACIM motor. When the KEY button is pressed, the user interface is automatically switched to the run motor menu. This action can be performed regardless of which sub-menu is selected.

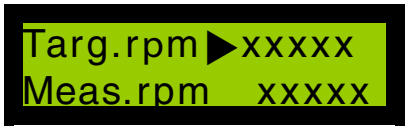
Figure 19. Help menu



3.2.3 Main menu: Changing the target and measured rotor speed

To enter the main menu, press the joystick RIGHT once more from help menu 3 (see *Figure 17*). 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. Once in the main menu, the target rotor speed and the measured speed can be displayed (see *Figure 20*).

Figure 20. Main window, showing target rotor speed and measured speed



1. To set the target rotor speed, press JOYSEL when the **Targ.rpm** function is active (▶ displayed and blinking).
2. After pressing JOYSEL, the arrow changes in to a double arrow ↕ to indicate that the value can be changed (see *Figure 21*).
3. Press the joystick UP and DOWN to increase and decrease the value.

Figure 21. Selecting the target rotor speed

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

3.3 Using the ACIM user interface sub-menus for motor control

3.3.1 ACIM user interface sub-menus

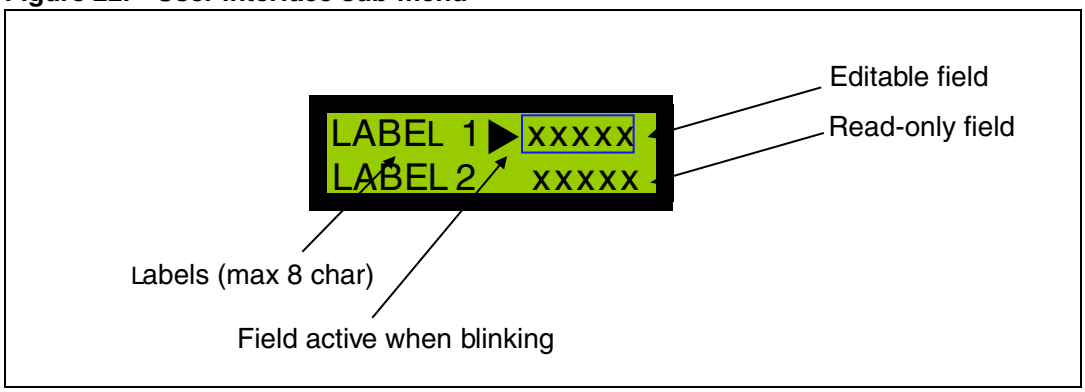
Press the joystick RIGHT or LEFT to navigate between sub-menus.

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 (for example, ampere, voltage, temperature). *Figure 22* shows the structure of a typical sub-menu, however, the corresponding unit is not displayed in this case.

The field can be editable or read-only:

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

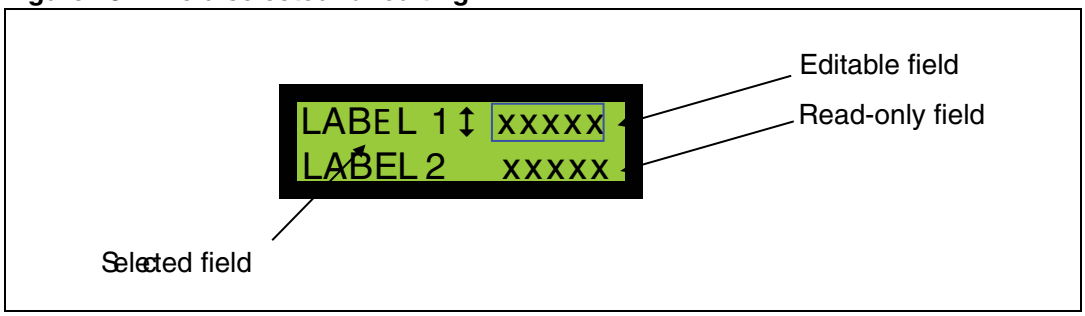
Figure 22. User interface sub-menu



To select and modify an 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. Press JOYSEL to select an active field. The cursor changes to an up/down arrow (↕)
3. Change the value by pressing the joystick UP or DOWN (see *Figure 23*)
4. To exit from edit mode, press JOYSEL again or change sub-menu

Figure 23. Field selected for editing



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

3.3.2 Speed regulator parameters

Overview

Two different speed closed loop mode drive are implemented and selectable (see *Section 2.4: Speed closed loop control*):

- V/f control and slip regulation
- MTPA control

Before starting the motor, either mode can be activated via the user interface (see *Section 3.3.5: Control strategy parameters*).

If MTPA is enabled (optimized drive efficiency), both the PI regulators should be tuned (see *Figure 24* and *Figure 25*), and both the optimum slip and V/f ratio should be settled (see *Section 3.3.3: Drive strategy parameters*).

If MTPA is disabled (optimized drive dynamics), only one PI regulator should be tuned (see *Figure 24*), and only V/f ratio should be settled (see *Section 3.3.3*).

For both situations, startup parameters should be specified (see *Section 3.3.4: Startup parameters*).

Note: Both the proportional and integral gains can be adjusted via the LCD user interface. The differential gain cannot be adjusted in this way. It is configured through VF_KD and/or MTPA_KD define statements in MC_ACIM_Drive_Param.h (see Section 5.1.3).

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) regulator, 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.

PID regulator parameter adjustment is performed by means of two sub-menus which are displayed in succession. The first sub-menu is used to configure constant flux control and slip regulation (see *Figure 24* below) via the PID regulator in the control scheme of *Figure 7: V/f and slip regulation control scheme*. The second sub-menu is used to configure constant slip control and flux regulation (see *Figure 25* below) via the PID regulator in the control scheme of *Figure 10: MTPA mode control scheme*.

The PID gains applied, are the ratio between the values defined through the LCD screen and the divisors settled in the `MC_ACIM_Drive_Param.h` file (*Section 5.1.3*).

Example

```
#define VF_KP_DIVISOR 128 // (unit none)
#define VF_KI_DIVISOR 512 // (unit none)
#define VF_KD_DIVISOR 16 // (unit none)
```

Figure 24. Constant flux control and slip regulation (via PID)

Figure 25. Constant slip control and flux regulation (via PID)

3.3.3 Drive strategy parameters

The sub-menu shown in *Figure 26* can be configured only when the firmware is in speed closed loop mode. This menu allows the parameters related to drive strategy to be adjusted while the motor is running (see*Section 2.4.1: V/f control and slip regulation* and *Section 2.4.2: Maximum torque per ampere (MTPA) control*).

Figure 26. Drive strategy parameters

"MTPA slip" is displayed as one tenth of a Hertz, while the "V/f ratio" value is multiplied by 1000.

3.3.4 Startup parameters

The sub-menu shown in *Figure 27* can be configured in both speed open loop and speed closed loop modes. This menu allows the parameters related to the startup phase to be adjusted while the motor is running.

Figure 27. Startup parameters

When the firmware is configured in speed closed loop mode, the “StUpVfR” parameter represents the maximum V/f ratio applied during startup. When the firmware is configured in speed open loop mode, StUpVfR is a constant ratio.

The “StUpSlip” parameter is the slip frequency maintained during startup (see *Section 2.6: Startup strategy* for further details about the startup procedure).

3.3.5 Control strategy parameters

The sub-menu shown in *Figure 28* can be configured only when the firmware is in speed closed loop mode. This menu allows the user to select a control strategy between MTPA (optimized efficiency) and V/f and slip regulation (optimized dynamics). Such a choice can be made only when the state machine is in idle state (before sending a start command).

In *Figure 28*, “MTPA mode” is editable and allows the control strategy to be selected. Switch it on, to enable the MTPA control. Switch it off to enable V/f and slip regulation.

The second line "MTPA" shows the mode which is currently in use.

Figure 28. Control strategy parameters

3.3.6 Displaying the DC bus voltage and heatsink temperature parameters

The sub-menu in *Figure 29* shows the measured DC bus voltage and the heatsink temperature. No parameters are selectable.

Figure 29. Bus voltage and heatsink temperature parameters

The “BUS DC” value is the rectified input voltage expressed in volts. The “Heatsink” value is the temperature measured by the negative temperature coefficient (NTC) resistor when it is placed on the power stage close to the heatsink of the power switches. Heatsink is expressed in degrees Celsius.

3.3.7 Fault messages

This section provides a description of all the fault messages that can be displayed on the LCD screen when using the ACIM firmware together with the STM8/128-MCKIT motor control starter kit.

There are seven different fault sources when using the ACIM firmware in conjunction with the STM8/128-MCKIT motor control starter kit:

Over current error

If a low-level is detected on the PWM-peripheral-dedicated pin (BKIN) while the STM8/128-MCKIT motor control starter kit is being used, either the hardware over temperature protection or the hardware overcurrent protection has been triggered.

Over temperature error

This fault message is displayed when an over temperature has been detected on the dedicated analog channel. The intervention threshold (NTC_THRESHOLD_C) and the related hysteresis (NTC_HYSTERESIS_C) are specified in the `MC_PowerStage_Param.h` header file (see *Section 5.1.6*).

Bus over voltage error

This fault message is displayed only if the DISSIPATIVE_BRAKE definition is commented (default setting) in the `MC_PowerStage_Param.h` configuration header file (see *Section 5.1.6*). It means that an over voltage has been detected on the dedicated analog channel. The intervention threshold (MAX_BUS_VOLTAGE) is specified in the `MC_PowerStage_Param.h` header file.

If the DISSIPATIVE_BRAKE definition is not commented in the `MC_PowerStage_Param.h` configuration header file, 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 supposedly able to dissipate the excess voltage across the bus capacitors.

Bus undervoltage error

A fault is detected when the bus voltage is below 18 V DC. This threshold is specified in the `MC_PowerStage_Param.h` header file by the `MIN_BUS_VOLTAGE` define statement.

Startup failed error

This fault message is displayed only if the firmware is configured in speed sensor mode. It signals that the startup output condition has not been fulfilled during motor acceleration (see also *Section 2.6: Startup strategy*).

Speed feedback error

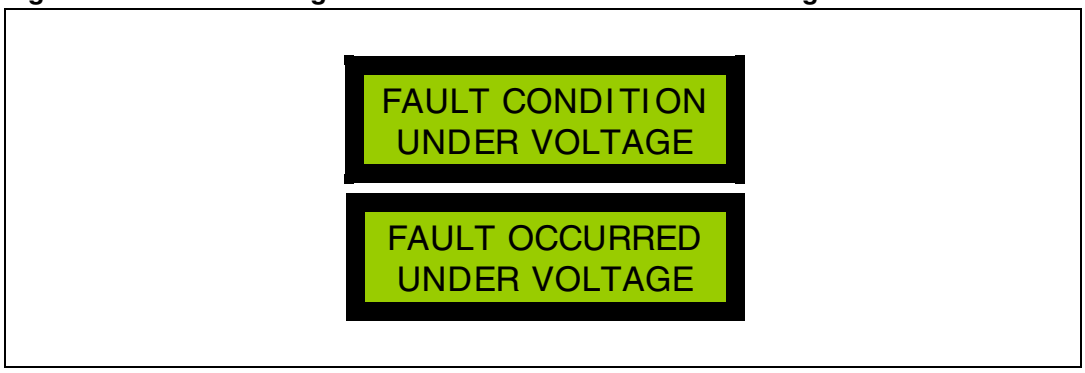
An error on the speed/position feedback has occurred.

Motor running error

This fault can occur only if the firmware is in speed sensor mode. It signals that the user is trying to start the motor when it is not still.

Figure 30 shows a typical error message, displayed on the LCD screen.

Figure 30. Error message shown in the event of an undervoltage fault



The message “FAULT CONDITION” is visible when the fault condition is still present.

The message “FAULT OCCURRED” is visible if the source of the fault has disappeared. This indicates to the user that the fault has occurred.

If several fault conditions occur concurrently, they are displayed in the LDC screen one after the other.

If the source of all faults disappears, pressing the KEY button causes the main state machine to switch from the fault to the idle state.

4 Getting started with the STM8S ACIM firmware

4.1 Application state machine

The motor control firmware library was developed around the state machine which is presented in *Figure 31*. The state machine is implemented in `MC_StateMacchine.c` module. It is composed of 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. This state is used to perform the main initializations.

Idle*

When the system is in idle state, the motor is stopped, and is waiting 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 initializations.

Start*

In this state the motor ramps up

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 system is in stop state when the motor is stopped.

Wait*

The system is in wait state when the motor is stopped. It remains in this state until a 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 while the fault condition is present.

Fault over*

When the fault condition disappears, the system enters fault over state to indicate which error occurred. The system also waits for 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

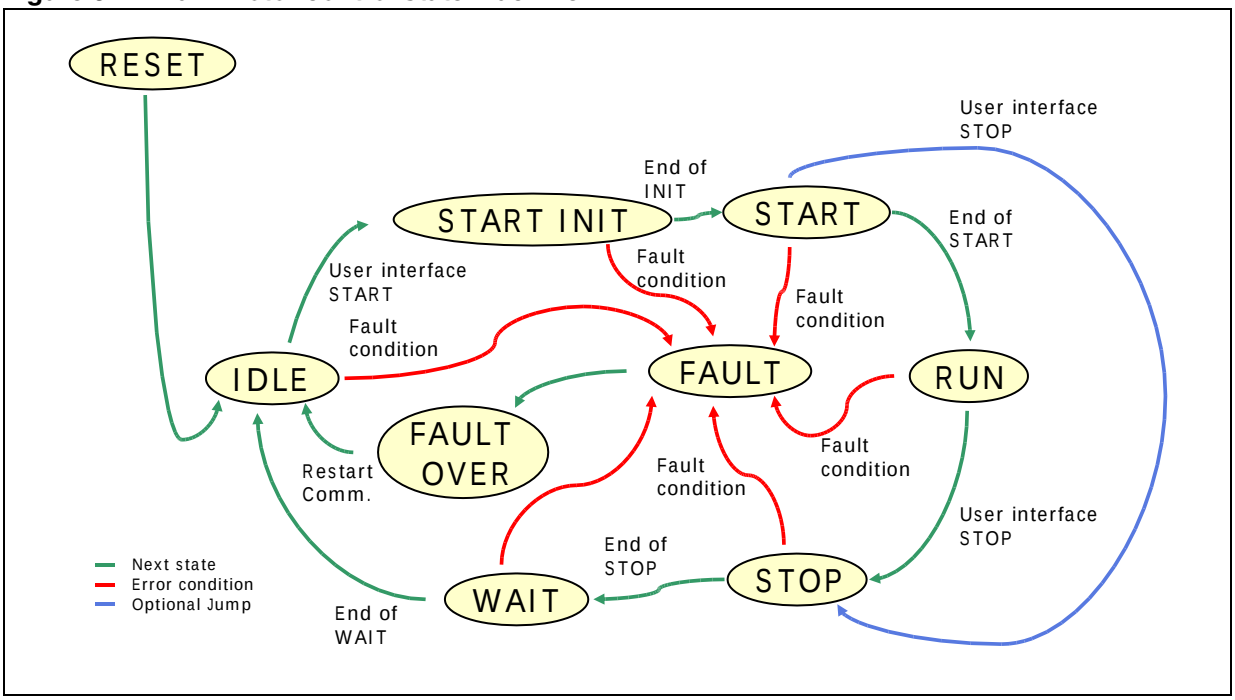
Each state corresponds to the execution of the related state machine function. The change of state is performed according to the value returned by that state machine function.

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

- ***State remain***: No change in state is required by the state machine function
- ***Next state***: The natural flow of the state machine is being followed (for example, idle -> init start -> start -> run). The natural flow is symbolized by green lines in *Figure 31*
- ***Optional jump***: A path deviation caused by user action has occurred (for example, start -> stop). Optional jumps are symbolized by blue lines in *Figure 31*.
- ***Error condition***: A fault condition has occurred (for example startup failure, hardware fault). The error conditions are 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 31. Main motor control state machine

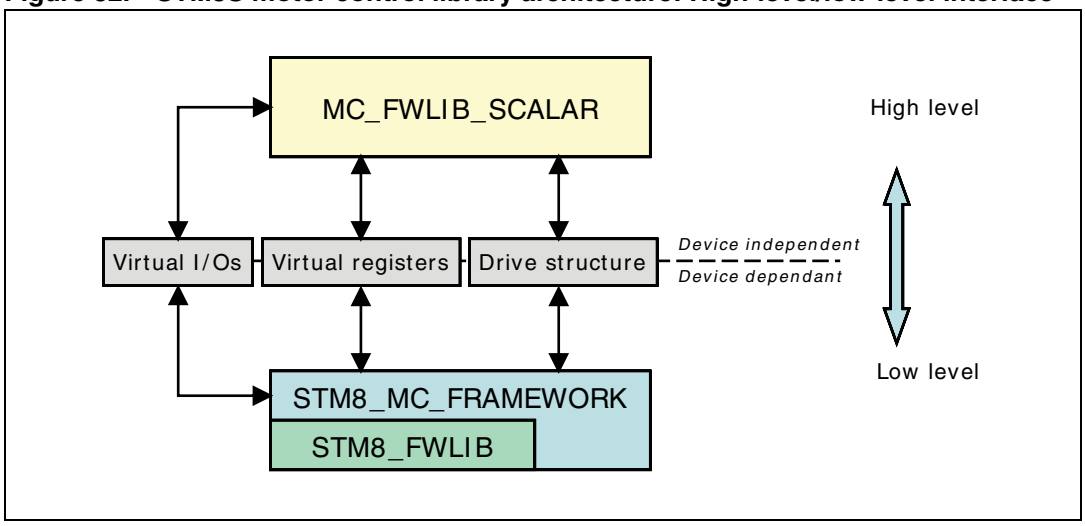


4.2 Library architecture

The STM8S ACIM motor control library has been logically divided into three different parts (*Figure 32*):

- 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

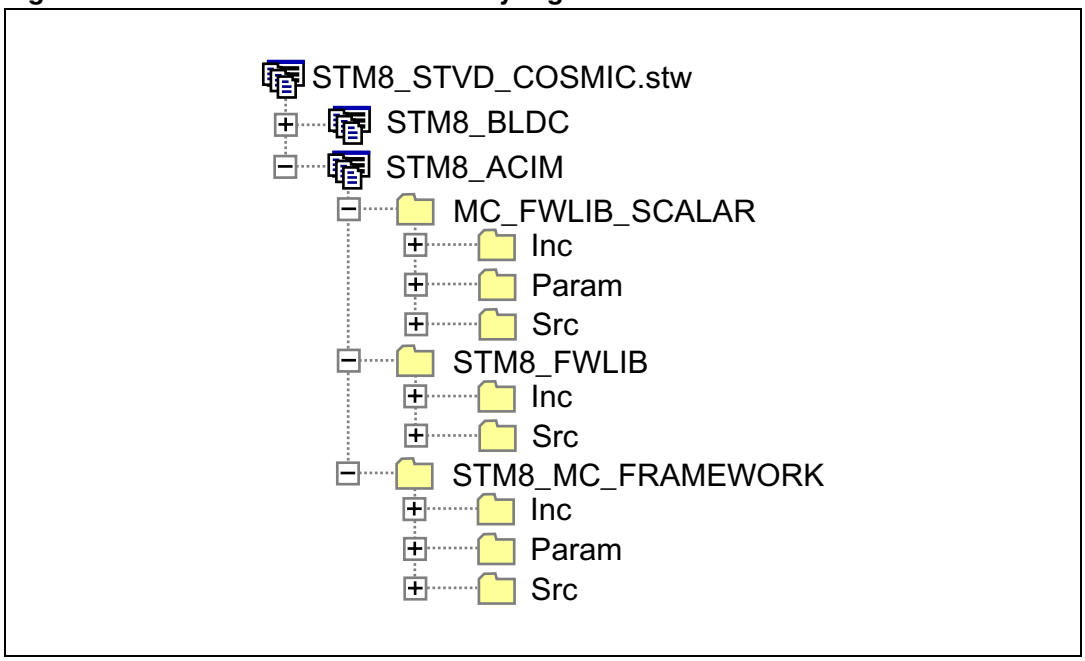
Figure 32. STM8S motor control library architecture: High-level/low-level interface



Each part of the library is in turn divided into is divided in three sublevels as shown in *Figure 33*:

- **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 contain everything necessary to customize the motor control drives. The **param** folder is not present in the **STM8_FWLIB**.

Figure 33. STM8S motor control library organization



The high-level modules contain the device-independent algorithms, while the low-level modules contain the hardware-dependent code. This 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 32*):

- 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 3* for a description of the virtual registers implemented in the ACIM firmware.

Table 3. Virtual registers

Name	Size	Description
VDEV_REG8_TACHO_PRESCALER	8 bit	Contains the prescaler value used in the speed measurement performed with the Tacho sensor
VDEV_REG8_TACHO_PULSE_NUMBER	8 bit	Contains the number of pulses to be taken into account for the speed measurement using the Tacho sensor
VDEV_REG8_ACIM_MODULATION_INDEX	8 bit	Contains the modulation index to be applied for the ACIM drive
VDEV_REG8_ACIM_MAX_MODULATION_INDEX	8 bit	Contains the maximum modulation index applicable for the ACIM drive
VDEV_REG8_HEATSINK_TEMPERATURE	8 bit	Contains the measured heat sink temperature
VDEV_REG16_ACIM_FREQUENCY	16 bit	Contains the reference stator frequency to be applied to the motor
VDEV_REG16_HALL_TACHO_COUNTS	16 bit	Contains the number of time counts between the first and last tacho signal edges, as measured in the last time period
VDEV_REG16_BOARD_BUS_VOLTAGE	16 bit	Contains the measured bus voltage expressed in volts
VDEV_REG16_HW_ERROR_OCCURRED	16 bit	Each bit represents an error condition that has already occurred
VDEV_REG16_HW_ERROR_ACTUAL	16 bit	Each bit represents a current error condition

4.2.2 Virtual I/Os

The virtual I/Os are low-level functions that are called by the high-level modules. For instance, if a high-level module wants to set a GPIO high-level output as test pin, it should call the virtual I/O: Device.ios.out8(VDEV_OUT8_LED_1, LED_ON) instead of driving directly the microcontroller register itself. The virtual I/Os implemented in the ACIM firmware are summarized in *Table 4*.

Table 4. Virtual I/Os

Name	Description
out8(VDEV_OUT8_DISPLAY_FLUSH,none)	Call used to invoke a refresh of the LCD screen
out8(VDEV_OUT8_DISPLAY_PRINTCH,none)	Call used to refresh the cursor displayed on the LCD (for example blinking).
out8(VDEV_OUT8_LED_1,command)	Call used to drive the virtual LED 1 which is mapped to a real H0 pin of the MB631 evaluation board. The command can take one of the values below, and is used to drive the HO pin in question: LED_ON: Sets a high state to the output turning on the related LED. LED_OFF: Sets a low state to the output turning off the related LED. LED_TOGGLE: Performs a toggle on the output switching it on or off.
out8(VDEV_OUT8_LED_2,command)	Same as above but, related to virtual LED 2 (H1 pin of the MB631 evaluation board).
out8(VDEV_OUT8_LED_3,command)	Same as above but, related to virtual LED 3 (H2 pin of the MB631 evaluation board).
out8(VDEV_OUT8_LED_4,command)	Same as above but, related to virtual LED 4 (H3 pin of the MB631 evaluation board).
inp8(VDEV_INP8_USER_INPUT,none)	Call used to get the input from the user interface (joystick and key button in the implementation of the MB631 evaluation board).

4.2.3 Drive structure

The drive structure contains the variables and parameters related to the motor and the drive. The ACIM drive structure for speed closed loop control is shown in *Table 5*.

Table 5. ACIM drive structure for speed closed loop control

Name	Type	Description
Control_Mode	Enum variable	Specifies the control mode selected: `SPEED_CLOSEDLOOP_MTA` or `SPEED_CLOSEDLOOP_VF`
Actual_Control_Mode	Enum variable	Specifies the actual control mode depending on the control mode selected and actual operating conditions: `SPEED_CLOSEDLOOP_MTA` or `SPEED_CLOSEDLOOP_VF`
hTarget_rotor_speed_RPM	Variable 16bit	Contains the target mechanical rotor speed, expressed in rpm
hTarget_rotor_speed_HzEl	Variable 16bit	Contains the target electrical rotor speed expressed in Hz*10
bDirection	Variable 8bit	Stores the spin direction selected at startup
hMeasured_rotor_speed_RPM	Variable 16bit	Contains the measured mechanical rotor speed, expressed in rpm
hMTPAslip	Variable 16bit	Contains the optimum slip to be maintained in the first control area of the MTPA drive
hVFConstant	Variable 16bit	Contains the V/f ratio conversion constant
hStartUpVFConstant	Variable 16bit	Contains the startup V/f ratio conversion constant
hStartUpSlip	Variable 16bit	Contains the slip frequency to be applied at startup
hBusVoltage	Variable 16bit	Contains the measured bus voltage expressed in volts
bHeatsinkTemp	Variable 8bit	Contains the measured heat sink temperature expressed in degrees Celsius
hUserADC	Variable 16bit	Contains the A/D conversion result of the user-selected channel
hPWM_Frequency	Constant 16bit	`#define PWM_FREQUENCY`: See description in `MC_ACIM_Drive_Param.h`
bPWM_RefreshRate	Constant 8bit	`#define PWM_REFRESH_RATE`: See description in `MC_ACIM_Drive_Param.h`)
hPWM_Prescaler	Constant 16bit	Contains the correct TIM1 prescaler to maintain the PWM resolution between 8 and 9 bits (depending on selected PWM frequency)
hPWM_Timer_ARR	Constant 16bit	Contains the correct TIM1 reload register to maintain the PWM resolution between 8 and 9 bits and unbiased output (depending on selected PWM frequency)

Table 5. ACIM drive structure for speed closed loop control (continued)

Name	Type	Description
bPWM_Timer_MMI	Constant 8bit	Contains the maximum PWM output resolution, i.e. the maximum modulation index (depending on selected PWM frequency)
bPWM_DeadTime	Constant 8bit	Specifies the parameter to be written in the TIM1 dead-time register (TIM1_DTR) according to `#define DEAD_TIME_NS`
hV_Constant	Constant 16bit	Contains the conversion factor to transform volts to a PWM modulation index
hHz_to_DPP_Conv	Constant 16bit	Contains the conversion factor to transform electrical Hz*10 to DPP (digits per PWM)
bMotor_Pole_Pairs	Constant 8bit	Expresses the number of motor pole pairs
bRPM_to_Hz_Conv	Constant 8bit	Contains the conversion factor to transform rpm to electrical Hz*10
hRPM_to_Hz_Ampl	Constant 16bit	Contains the amplification factor to enhance the rpm to electrical Hz*10 conversions
hDigit_to_BusV_Conv	Constant 16bit	Contains the conversion factor for DC bus voltage measurements
bNTC_alpha	Constant 8bit	`#define TEMP_SENS_ALPHA`: See description in `MC_PowerStage_Param.h`
bNTC_beta	Constant 8bit	`#define TEMP_SENS_BETA`: See description in `MC_PowerStage_Param.h`
bStartup_Vo	Constant 8bit	`#define STARTUP_VO`: See description in `MC_ACIM_Motor_Param.h`
hMax_Speed	Constant 16bit	`#define MAX_SPEED_RPM`: See description in `MC_ACIM_Motor_Param.h`
hMin_run_speed	Constant 16bit	`#define MIN_RUN_SPEED`: See description in `MC_ACIM_Drive_Param.h`
hStall_speed	Constant 16bit	`#define STALL_SPEED`: See description in `MC_ACIM_Drive_Param.h`
hStartup_val_speed	Constant 16bit	`#define STARTUP_VAL_SPEED`: See description in `MC_ACIM_Drive_Param.h`
hStartup_duration	Constant 16bit	`#define STARTUP_DURATION`: See description `in MC_ACIM_Drive_Param.h`
hStartUpFinalSpeed_HzEl	Constant 16bit	`#define STARTUP_FINAL_SPEED`: See description in `MC_ACIM_Drive_Param.h`
bControlLoop_Period_ms	Constant 8bit	Expresses the PID regulator action interval expressed in milliseconds
pPID_VF_Struct	Constant pointer	Pointer to the speed PID structure
pPID_MTPA_Struct	Constant pointer	Pointer to the speed PID structure

The ACIM drive structure for speed open loop control is show in *Table 6*.

Table 6. ACIM drive structure for speed open loop control

Name	Type	Description
Control_Mode	Enum variable	Specifies the control mode selected: SPEED_OPENLOOP or SPEED_OPENLOOP_LOAD_COMPENSATION
hTarget_rotor_speed_RPM	Variable 16bit	Contains the target mechanical rotor speed, expressed in rpm
hTarget_rotor_speed_HzEl	Variable 16bit	Contains the target electrical rotor speed expressed in Hz*10
bDirection	Variable 8bit	Stores the spin direction selected at startup
hMeasured_rotor_speed_RPM	Variable 16bit	Contains the measured mechanical rotor speed, expressed in rpm
hActual_rotor_speed_HzEl	Variable 16bit	Contains the estimated electrical rotor speed expressed in Hz*10
hAccelerationSlope	Variable 16bit	Defines the acceleration slope constant, which is a function of selected acceleration (`#define OPEN_LOOP_ACCELERATION_SLOPE`), motor pole pairs, and control loop period
hSlip	Variable 16bit	Defines the slip frequency (Hz*10) to be applied
hVFConstant	Variable 16bit	Contains the V/f ratio conversion constant
hStartUpVFConstant	Variable 16bit	Contains the startup V/f ratio conversion constant
hStartUpSlip	Variable 16bit	Contains the slip frequency to be applied at startup
hBusVoltage	Variable 16bit	Contains the measured bus voltage expressed in volts
bHeatsinkTemp	Variable 8bit	Contains the measured heat sink temperature expressed in degrees Celsius
hUserADC	Variable 16bit	Contains the A/D conversion result of the user-selected channel
hPWM_Frequency	Constant 16bit	`#define PWM_FREQUENCY`: See description in `MC_ACIM_Drive_Param.h`)
bPWM_RefreshRate	Constant 8bit	`#define PWM_REFRESH_RATE`: See description in `MC_ACIM_Drive_Param.h`
hPWM_Prescaler	Constant 16bit	Contains the correct TIM1 prescaler to maintain the PWM resolution between 7 and 8 bits (depending on selected PWM frequency)
hPWM_Timer_ARR	Constant 16bit	Contains the correct TIM1 reload register to maintain the PWM resolution between 7 and 8 bits and unbiased output (depending on selected PWM frequency)
bPWM_Timer_MMI	Constant 8bit	Contains the maximum PWM output resolution, i.e. the maximum modulation index (depending on selected PWM frequency)

Table 6. ACIM drive structure for speed open loop control (continued)

Name	Type	Description
bPWM_DeadTime	Constant 8bit	Specifies the parameter to be written in the TIM1 dead-time register (TIM1_DTR) according to `#define DEAD_TIME_NS`
hV_Constant	Constant 16bit	Contains the conversion factor to transform volts to a PWM modulation index
hHz_to_DPP_Conv	Constant 16bit	Contains the conversion factor to transform electrical Hz*10 to DPP (digits per PWM)
bMotor_Pole_Pairs	Constant 8bit	Expresses the number of motor pole pairs
bRPM_to_Hz_Conv	Constant 8bit	Contains the conversion factor to transform rpm to electrical Hz*10
hRPM_to_Hz_Ampl	Constant 16bit	Contains the amplification factor to enhance the rpm to electrical Hz*10 conversions
hDigit_to_BusV_Conv	Constant 16bit	Contains the conversion factor for DC bus voltage measurements
bNTC_alpha	Constant 8bit	`#define TEMP_SENS_ALPHA`: See description in `MC_PowerStage_Param.h`
bNTC_beta	Constant 8bit	`#define TEMP_SENS_BETA`: See description in `MC_PowerStage_Param.h`
bStartup_Vo	Constant 8bit	`#define STARTUP_VO`: See description in `MC_ACIM_Motor_Param.h`
hMax_Speed	Constant 16bit	`#define MAX_SPEED_RPM`: See description in `MC_ACIM_Motor_Param.h`
hMin_run_speed	Constant 16bit	`#define MIN_RUN_SPEED`: See description in `MC_ACIM_Drive_Param.h`
hStall_speed	Constant 16bit	`#define STALL_SPEED`: See description in `MC_ACIM_Drive_Param.h`
hStartup_val_speed	Constant 16bit	`#define STARTUP_VAL_SPEED`: See description in `MC_ACIM_Drive_Param.h`
hStartup_duration	Constant 16bit	`#define STARTUP_DURATION`: See description in `MC_ACIM_Drive_Param.h`
hStartUpFinalSpeed_HzEl	Constant 16bit	`#define STARTUP_FINAL_SPEED`: See description in `MC_ACIM_Drive_Param.h`
bControlLoop_Period_ms	Constant 8bit	Expresses the periodicity of the open loop control in milliseconds

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 Combined utilization of ADC and TIM1 for motor driving

The STM8S ADC and advanced timer (TIM1) peripherals are used in close combination in the current firmware library example. Their utilization is explained jointly below.

The low-level shell of the driving strategy is implemented in the firmware library source file `MC_stm8s_ACIM_drive.c`.

In the ACIM motor control firmware library, the generated PWM pattern is center-aligned. This is the best arrangement for reducing magnetostriction noise and switching losses. In addition, TIM1 has edge aligned PWM capability.

The three-phase sine wave with third harmonic injection is updated (duty cycles are updated) with a frequency related to the selected PWM frequency and the define statement `PWM_REFRESH_RATE`. The sine wave reference look up table is stored in the Flash memory (const u8 SINE3RDHARM[256], `MC_stm8s_ACIM_Param.h`). It has an 8-bit resolution (± 127).

TIM1 has 16-bit resolution and a maximum frequency which is tied to the CPU frequency (24 MHz max for performance line STM8s, 16 MHz max for access line STM8s).

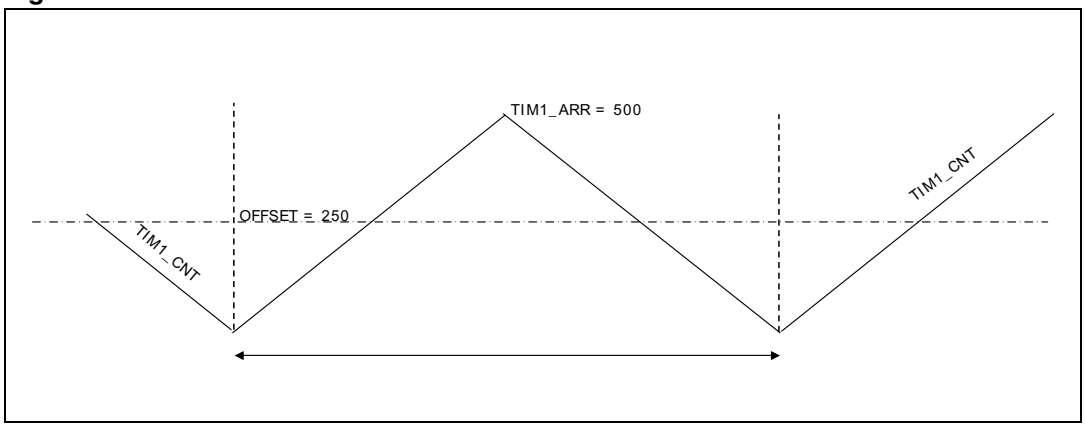
The maximum duty cycle resolution achievable using the current firmware library ranges from 8 bits to 9 bits. This range allows the generation of a complete array of PWM frequencies (according to user needs), while simultaneously maintaining fixed and speed optimized PWM output calculation.

Example

When the CPU frequency is 24 MHz, the total number of timer counts required to generate a 12-kHz PWM frequency is 2000. In this case, the timer reload register (ARR) is 2000 due to the center-aligned pattern. Therefore, the offset value to have a zero output, is 500. This gives an output resolution of approximately 10 bits (±500) but, using 24-bit calculations.

On the other hand, by keeping the output resolution between 8 and 9 bits, the calculations required for PWM update can be sped up. *Figure 34* shows that by decreasing the TIM1 speed to 12 MHz (with a prescaler value TIM1_PSCR =1), the reload register is 500 and the output resolution is about 9 bits. This allows 16-bit intermediate calculations.

Figure 34. TIM1 initialization



1. Legend: PWM frequency = 12 kHz
   PWM period = 83.3 µs
   CPU frequency = 24 MHz
   TIM1 frequency = 12 MHz (TIM1_PSCR = 1)
   Total TIM1 counts required = 1000

According to the specified PWM frequency (as specified in *Section 5.1.3: ACIM drive control define statements: MC_ACIM_Drive_Param.h*), the timer prescaler is automatically chosen in accordance with *Table 7* and *Table 8*. For instance, if the CPU frequency is 16 MHz and the desired PWM frequency is 6 kHz, the prescaler adopted is automatically 2. The amplitude resolution can be calculated using *Equation 14*.

Equation 14

$$\text{PWM amplitude resolution} = \frac{\text{CPUfrequency}}{4 \times \text{PWMfrequency} \times (\text{TIM1\_PSCR} + 1)} = \pm 222 \text{counts}$$

Table 7. PWM amplitude resolution @CPU frequency 24 MHz

PWM frequency (Hz)	9-bit resolution @ Hz	Min resolution (counts)	TIM_PSCR used
23437-46875	23437	±128 @ 46875 Hz	0
11718-23436	11718	±128 @ 23436 Hz	1
7812-11717	7812	±170 @ 11717 Hz	2
5859-7811	5859	±192 @ 7811 Hz	3
4687-5858	4687	±204 @ 5858 Hz	4
3906-4686	3906	±213 @ 4686 Hz	5
3348-3905	3348	±219 @ 3905 Hz	6

Table 8. PWM amplitude resolution @CPU frequency 16 MHz

PWM frequency (Hz)	9-bit resolution @ Hz	Min resolution (counts)	TIM_PSCR used
15625-31250	15625	±128 @ 31250 Hz	0
7812-15624	7812	±128 @ 15624 Hz	1
5208-7811	5208	±170 @ 7811 Hz	2
3906-5207	3906	±192 @ 5207 Hz	3
3125-3905	3125	±204 @ 3905 Hz	4
2604-3124	2604	±213 @ 3124 Hz	5
2232-2603	2232	±219 @ 2603 Hz	6

The output frequency resolution can be calculated using *Equation 15*.

Equation 15

$$\text{PWM frequency resolution} = \frac{\text{PWMfrequency}}{\text{PWM\_REFRESH\_RATE} \times 65536}$$

Clearly, the higher the `PWM_REFRESH_RATE` define statement (which can be adjusted as explained in *Section 5.1.3: ACIM drive control define statements: MC_ACIM_Drive_Param.h*), the better the resolution, but the lower the maximum achievable stator frequency.

Example

To synthesize a three-phase sine wave with at least 12 points having selected a PWM frequency of 16 kHz, and a PWM refresh rate of 3, the maximum stator frequency is about 444 Hz.

Figure 35 shows the scheme adopted in the current firmware library for ADC and TIM1 peripheral management. The output compare registers, TIM1_CCR1-2-3, containing the new PMW duty cycles, are updated automatically at counter underflow and in consideration of the `PWM_REFRESH_RATE` define statement. If TIM1 is configured in center-aligned mode, it is capable of a double update (counter overflow and underflow events). This feature of TIM1 can be used to improve the frequency resolution. However, it has not been used in this version of the ACIM firmware library.

Figure 36 shows the most important functions implemented in the current firmware, together with their connections. It also shows the main connections with other modules.

The TIM1 update interrupt function (TIM1_UPD_OVF_TRG_BRK_IRQHandler) is used to run the ADC manager routine. Depending on the `HEATSINK_SAMPLING_FREQUENCY` and `USER_ADC_SAMPLING_FREQUENCY` define statements (see *Section 5.1.8: Microcontroller specific ACIM drive define statements: MC_stm8s_ACIM_param.h*), it evaluates whether or not to convert the heatsink temperature and/or the user defined signal in the next PWM period. Conversions are started as soon as their sequence is established.

An ADC end-of-conversion interrupt (ADC2_IRQHandler function) is awakened after each single conversion has finished. The purpose of the handler, in this case, is to store the result of the conversion and to start a single conversion if instructed by the ADC manager routine. Following this and with reference to the `BUS_SAMPLING_FREQUENCY` define statement (*Section 5.1.3: ACIM drive control define statements: MC_ACIM_Drive_Param.h*), the measured DC bus voltage is averaged. Buffer length is defined by the `BUSVOLT_BUFFER_SIZE` define statement (see *Section 5.1.8: Microcontroller specific ACIM drive define statements: MC_stm8s_ACIM_param.h*).

It is placed in the VDEV_REG16_BOARD_BUS_VOLTAGE virtual register. Meanwhile, the measured heatsink temperature is checked against its upper threshold and stored in the VDEV_REG8_HEATSINK_TEMPERATURE virtual register.

Conversely, TIM1_CCR4 is used to trigger the A/D conversion of the DC bus voltage. Since the center of the PWM pattern is mainly clear from power device switching, this area is chosen for bus voltage reading by selecting `TIM1_CCR4 = TIM1_ARR -1`.

As soon as the triggered conversion has finished, the related ADC EOC handler is entered. Its assigned tasks in this case are:

- Compensate the DC bus ripple through variation of the modulation index required by the drive (received through the function dev_driveRun and the virtual register VDEV_REG8_ACIM_MODULATION_INDEX).
- Update the three-phase sine wave generated on the basis of the electrical frequency required by the drive (received through the function dev_driveRun and the virtual register VDEV_REG16_ACIM_FREQUENCY).
- Update the corresponding TIM1 output compare registers: TIM1_CCR1-2-3
- Check if the converted DC bus voltage is higher than the overvoltage threshold defined in `MC_PowerStage_Param.h` (see *Section 5.1.6: Power stage define statements: MC_PowerStage_Param.h*).

  Note: The brake resistor is switched on (hysteresis control) or a fault (overvoltage) is generated depending on whether DISSIPATIVE_BRAKE is commented.

Figure 35. TIM1 and ADC utilization

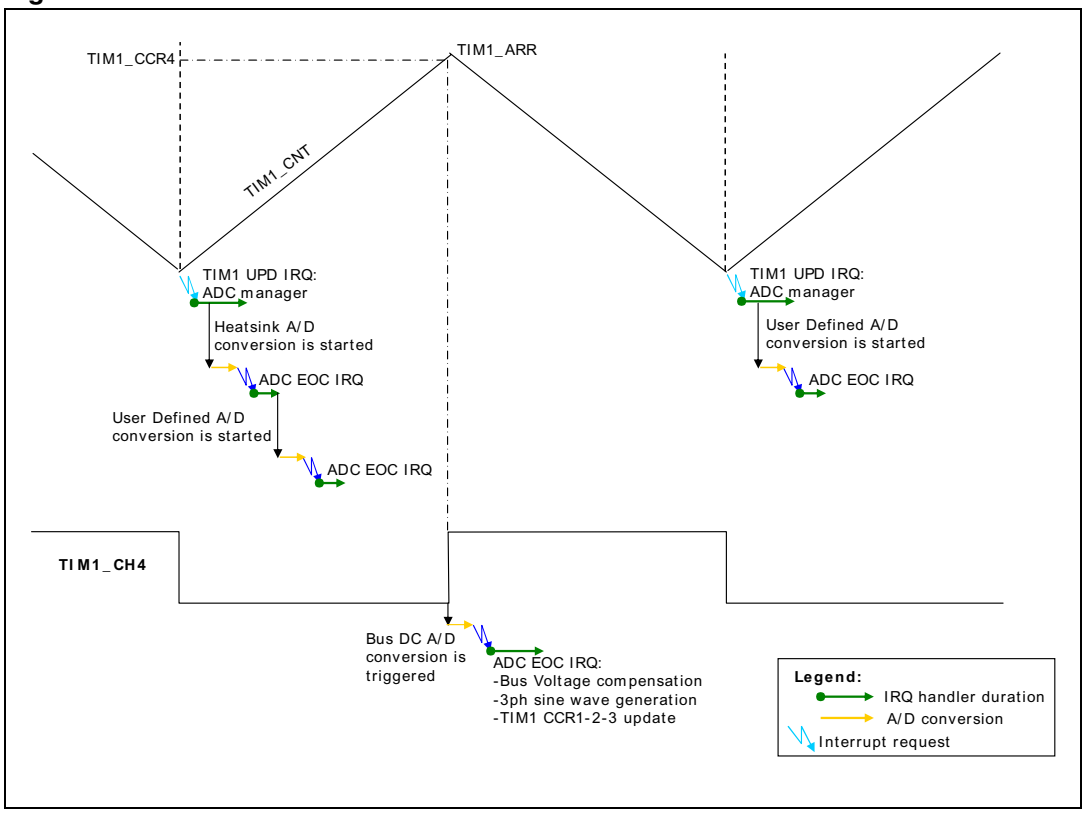
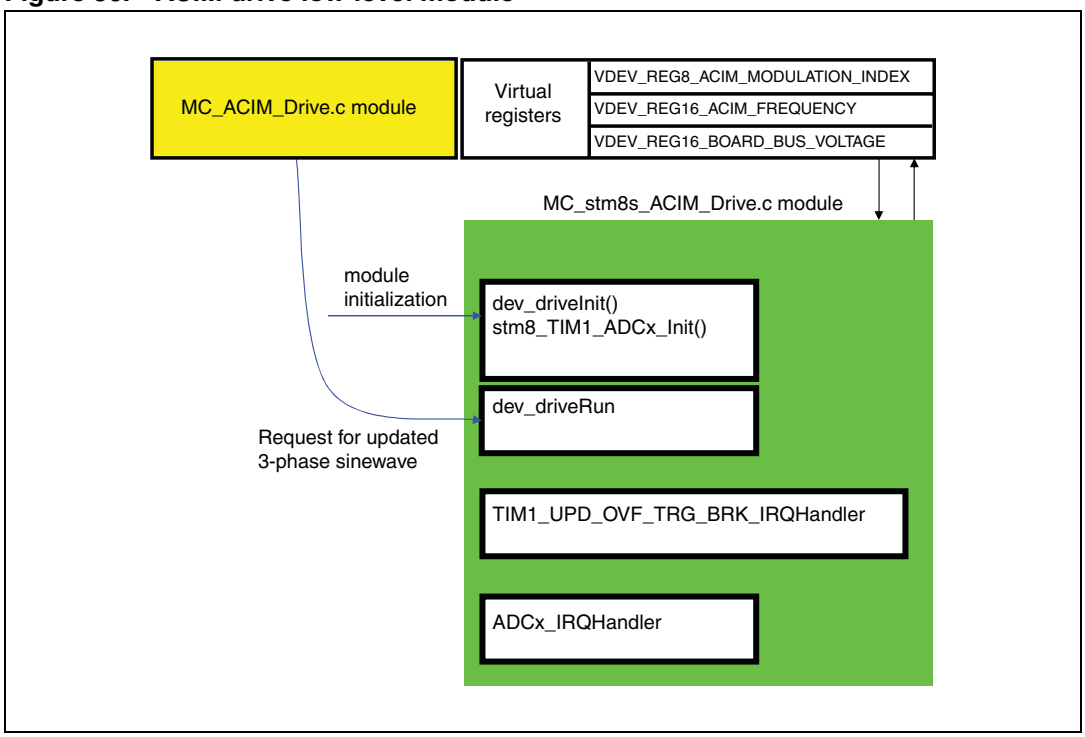


Figure 36. ACIM drive low-level module



4.3.2 Tachogenerator signal reading

The low-level shell of the measurement method is implemented in the ACIM firmware library source file `MC_stm8s_tacho`.c.

Motor speed is read via a frequency measurement of the signal coming from the tachogenerator sensor. TIM1 is used for this purpose. One channel is configured in input capture mode with falling edge detection capability. Note that STM8s timers have both rising and falling edge detection capability, which can be used to enhance the resolution.

Figure 37 shows the measurement method which has been implemented in the current firmware.

Figure 38 shows the most important functions implemented in the current firmware, together with their connections. It also shows the main connections with other modules.

Two registers, register 0 and register 1, are defined which are used alternatively, during each period, to store the captured values of the timer. Each register has three locations which are called 0a/0b/index0 (register 0) or 1a/1b/index1 (register 1). The first location (0a or 1a respectively) is used to store the earliest capture which has occurred after a TIMx update (where x stands for the timer number selected to read the period of the tacho signal). The second location (0b or 1b respectively) is rewritten each time a new capture arrives. The third location (index0 or index1 respectively) counts the number of captures that have occurred in that timer period.

Four basic operations of tachogenerator signal reading

- On a capture event, the corresponding interrupt handler is entered. The captured value of TIMx is written in a register location (see above). The index variable is incremented.
- On a timer update event, the corresponding interrupt handler is entered. Pointing to registers is alternated (context switching). For example, if register 1 was used to store captures in the previous period, register 0 is used in the next period while register 1 is being analyzed (see below).
- The timer prescaler (TIMx_PSCR) is managed and configured to detect a number of captures equal to the user-defined define statement, `TACHO_PULSE_AVERAGED` (see *Section 5.1.4: Tacho sensor define statements: MC_tacho_param.h*), during the next timer period.
- Captures that occurred in the last timer period (register 1, following the example above) are processed to calculate the tacho signal period. Three sets of information are entered into the virtual registers (see *Table 3: Virtual registers*): the number of timer counts between the first and last captures are stored in register VDEV_REG16_TACHO_COUNTS, the number of captures occurred is stored in register VDEV_REG8_TACHO_PULSES, while the timer prescaler used for that particular measurement is stored in VDEV_REG8_TACHO_PRESCALER. On the basis of these data, the high-level section of the tachogenerator signal measurement module calculates the motor speed. It takes into account the tachogenerator pole number and timer clock frequency (see *Section 4.4.4: Tachogenerator signal reading*).

Figure 37. Tachogenerator reading method

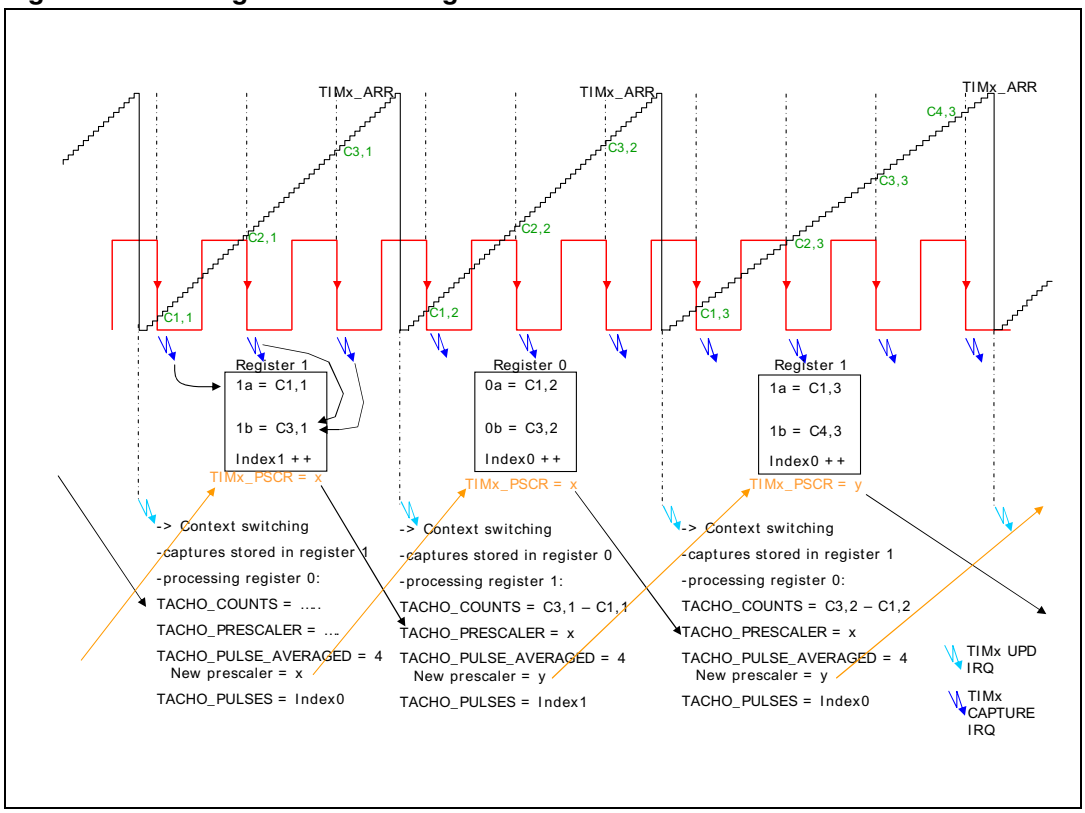
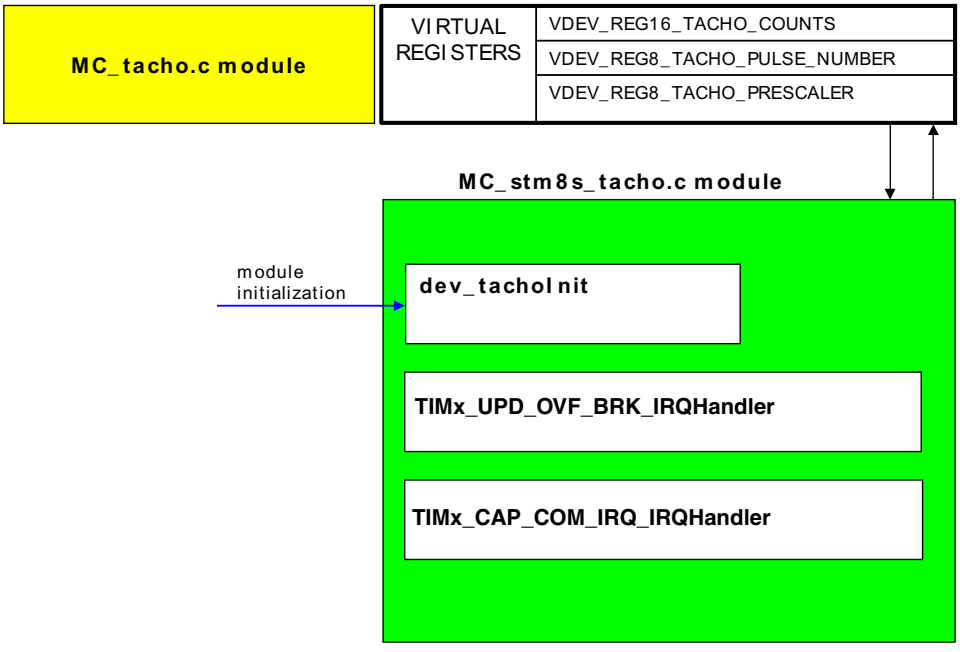


Figure 38. Tachogenerator sensing low-level module



4.3.3 Dissipative brake

The ACIM motor is able to transform kinetic energy into electrical energy just like a dynamo.

This situation occurs when the control tries to decelerate the motor, particularly, when the stator frequency applied is lower than the rotor speed resulting in a negative slip speed being applied.

In this situation, the inverter bulk capacitor is charged unless the power system used has regenerative capabilities. The voltage across the bulk capacitors could increase to a destructive level depending on the amount of energy transferred.

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.

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 generated. Instead the break control pin is driven to turn on the external brake resistor to dissipate the extra energy.

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

If the value falls below a certain threshold, the dissipative brake is stopped. To add a hysteresis between such switching on and off, the turn “off” threshold is reduced compared to the turn “on” threshold.

See *Section 5.1.6: Power stage define statements: MC_PowerStage_Param.h* for details on how to enable or disable this feature. See *Section 5.2: Setting up the system when using a brake resistor* for details on the hardware setup required to use this feature.

4.4 High-level control

This section explains how to implement a high-level ACIM drive algorithm independently from the microcontroller peripheral definitions.

4.4.1 Virtual timers

Virtual timers are high-level hardware-independent general purpose counters. They are used to manage the execution of the code performed at specified time intervals, for example, the speed regulation algorithm that must be performed at fixed time intervals.

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

The virtual timers can be used in two modes:

- Polling mode: The end of counting has to be checked using a specific function call. 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 are implemented inside each “drive firmware”. Each virtual timer is dedicated to specific operations. It is identified by a name, VTIMx, where x is the number of the virtual timer. Virtual timer names can be customized through a define statement. For example, timer number 0 can be named VTIM_KEY by using the define statement:
`#define VTIM_KEY VTIM0`

4.4.2 Using the ACIM virtual timers

The list of virtual timers used by the ACIM drive is given in *Table 9*.

Table 9. ACIM 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_TIME300ms).
VTIM_DISPLAY_REFRESH	Polling	This virtual timer is used to count the LCD refresh frequency (DISPLAY_REFRESH_TIME300ms).
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 as 1 s by the firmware.
V_TIM_ACIMDRIVE	Automatic	This virtual timer is used to call the ACIM_drive function (see *Section 4.4.3: ACIM scalar control*).
V_TIM_ACIMSTARTUP	Automatic	This virtual timer is used to time out the startup procedure
V_TIM_ACIMUPDATEINFO	Polling	This virtual timer defines the refresh rate of the information sent to the LCD and DAC.
V_TIM_ACIMSTARTUPINIT	Polling	This virtual timer is used to define the duration of the high side driver bootstrap capacitor-charging phase, executed before each motor startup

4.4.3 ACIM scalar control

The high-level shell of the ACIM scalar control drive is implemented in the current firmware library source file `MC_ACIM_Drive.c`.

This module implements the control schemes explained in *Section 2.4: Speed closed loop control* and *Section 2.5: Speed open loop control*.

Figure 39 shows the functions implemented in the current firmware, together with their connections. It also shows the main connections with other modules.

The core function is ACIM_Drive() which is awakened automatically by the virtual timer V_TIM_ACIM_DRIVE. The periodicity is fixed by the `CONTROL_LOOP_PERIOD` define statement (see *Section 5.1.3: ACIM drive control define statements: MC_ACIM_Drive_Param.h*).

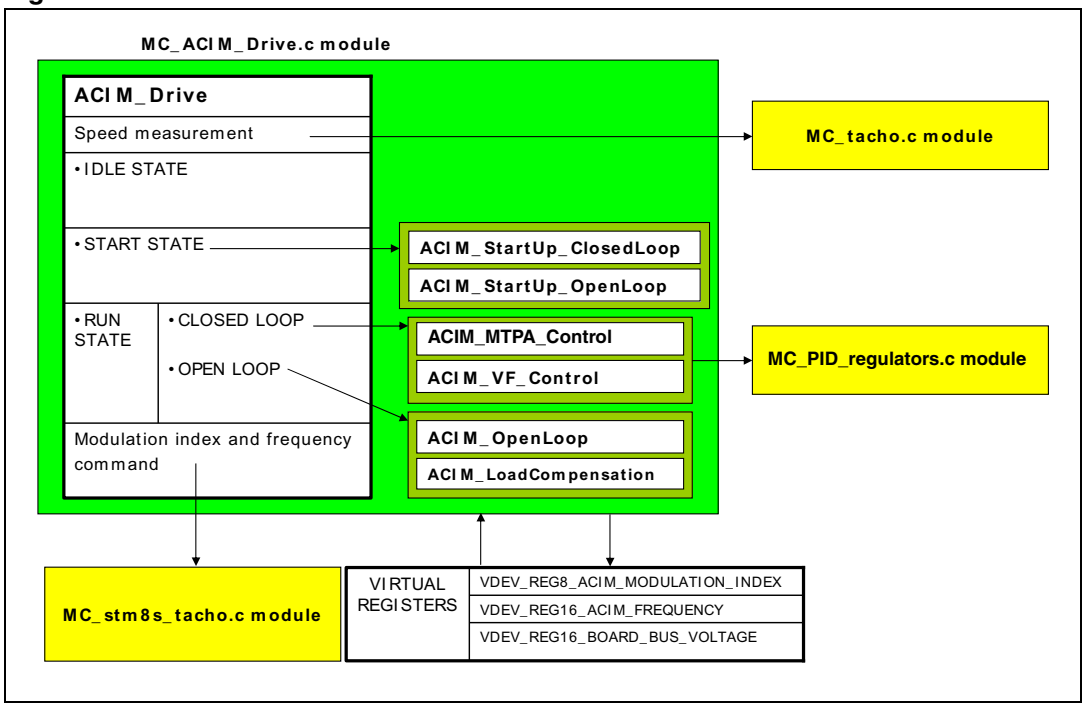
This function can be divided into three successive stages:

1. A request for the rotor speed measurement to be updated: The Tacho_CalcSpeed_HzMec function is called using the `MC_tacho.c` module.
2. The three-phase sine wave voltage and frequency to be calculated in the next control period. This is done according to the current state of the application state machine (if in idle, start, or run state).
   a) If in idle state: No operations
   b) If in start state: ACIM_StartUp_ClosedLoop or ACIM_StartUp_OpenLoop functions are called.
   c) If in run state:

      Closed loop mode: ACIM_MTPA_Control or ACIM_VF_Control functions are called according to the actual drive strategy. For example, ACIM_VF_Control implements the control scheme shown in *Figure 7: V/f and slip regulation control scheme* while ACIM_MTPA_Control implements the scheme shown in *Figure 10: MTPA mode control scheme*. The PID linear regulators are executed by the PID_Regulator function (`MC_pid_regulators.c` module).

      If in open loop mode: The ACIM_OpenLoop function is called. If open loop mode with load compensation functionality is enabled, the ACIM_LoadCompensation function is also called. These functions implement the control scheme shown in *Figure 11: Speed open loop control with load compensation*. The ACIM_LoadCompensation function executes the required look up table.
3. A request to update the three-phase sine wave with the calculated voltage and frequency. The dev_driveRun function is called using the `MC_stm8s_ACIM_drive.c` module.

Figure 39. ACIM scalar control module



4.4.4 Tachogenerator signal reading

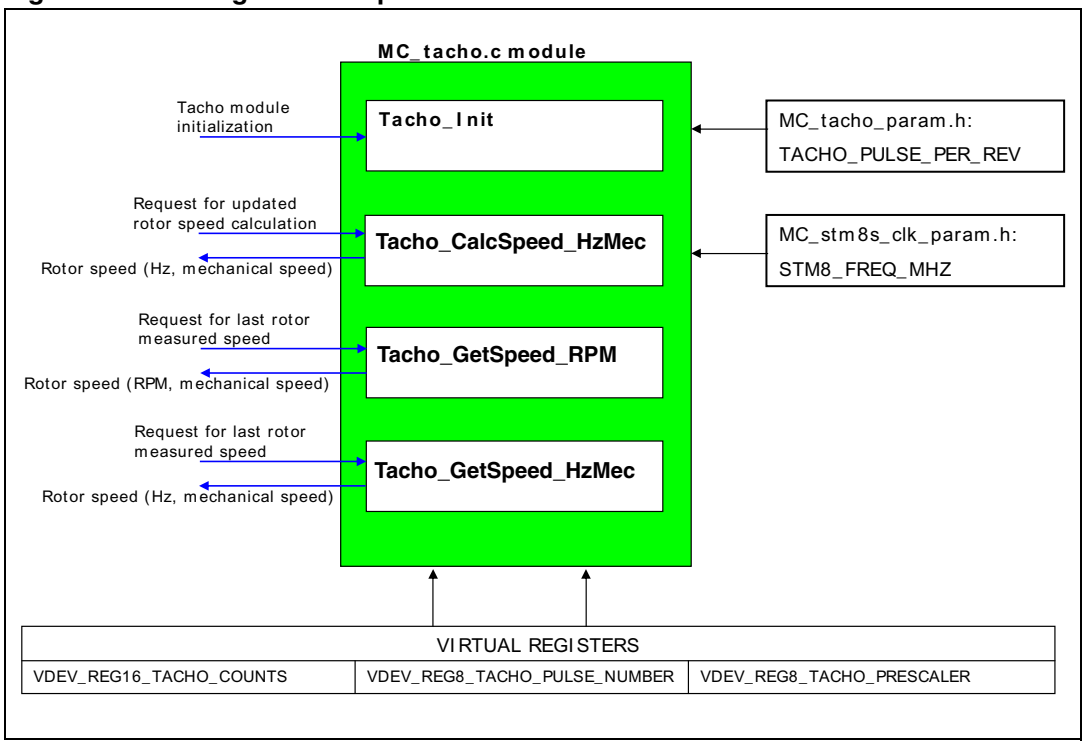
The high-level shell of the tachogenerator signal reading is implemented in the current firmware library source file `MC_tacho.c` (see *Figure 40*).

This module's operations rely on measurements performed by the low-level shell (see *Section 4.3.2: Tachogenerator signal reading*) which are subsequently stored in virtual registers. The operations take account of define statements which are related to the specific tachogenerator sensor in use. They also take into account the timer clock frequency. For example, motor speed is calculated using *Equation 16*.

Equation 16

$$Motorspeed = \frac{STM8\_FREQ \times VDEV\_TACHO\_PULSE\_NUMBER}{VDEV\_TACHO\_COUNTS \times VDEV\_PRESC\ ALER \times TACHO\_PULSE\_PER\_REV}$$

Figure 40. Tachogenerator speed measurement module



5 Designing an application using the ACIM software library

It is relatively easy to set up an operational evaluation platform with a drive system that includes the STM8/128-MCKIT motor control starter kit (featuring the STM8S microcontroller on which the software runs) and a permanent-magnet motor.

This section explains how to quickly configure your system and, if necessary, customize the library accordingly.

Follow the steps below to accomplish this task:

1. Collect all the information needed regarding the hardware in use (motor parameters, power device features, speed/position sensor parameters, current sensors transconductance).
2. Edit, using an integrated development environment (IDE), the following high-level parameter files present in the folder **STM8-MC_KIT\MC_FWLIB_SCALAR\param**:
   - `MC_ACIM_conf.h` (see *Section 5.1.1*)
   - `MC_ACIM_Motor_Param.h` (see *Section 5.1.2*)
   - `MC_ACIM_Drive_Param.h` (see *Section 5.1.3*)
3. If the drive is being performed using a tachogenerator sensor, setup the parameter header file:
   - `MC_tacho_param.h` (see *Section 5.1.4*)
4. If working with different hardware that is compatible with the STM8/128-MCKIT motor control starter kit, edit (using an IDE) the following parameter header files too:
   - `MC_ControlStage_param.h` (see *Section 5.1.5*),
   - `MC_PowerStage_Param.h` (see *Section 5.1.6*)
5. It is also necessary to edit the following low-level parameter files present in the folder **STM8-MC_KIT\STM8_MC_FRAMEWORK\param**:
   - `MC_stm8s_clk_param.h` (see *Section 5.1.7*)
   - `MC_stm8s_ACIM_param.h` (see *Section 5.1.8*)
   - `MC_stm8s_port_param.h` (see *Section 5.1.9*)
   - `MC_stm8s_tacho_param.h` (see *Section 5.1.10*)
6. Re-build the project and download it on the STM8S microcontroller.

Note: These modifications can be performed automatically using the STM8S_MC_Firmware_Library builder.

5.1 Customizing the ACIM software library parameter file

5.1.1 ACIM configuration file: `MC_ACIM_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 the actual speed/position sensor.

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 speed/position sensor choice, depending on requirements, includes speed closed loop, speed open loop, and tacho sensing:

- `#define SPEED_CLOSED_LOOP`

  Uncomment the above statement when a tachogenerator sensor is being used to detect rotor speed.

  Based on this feedback, speed closed loop control is carried out as explained in *Section 2.4*.

  Fill out `MC_tacho_param.h` and `MC_stm8s_tacho_param.h` (see *Section 5.1.4* and *Section 5.1.10*) which concern the define statements related to speed sensing.

  Fill out `MC_ACIM_Drive_Param.h` and `MC_ACIM_Motor_param.h` (see *Section 5.1.3* and *Section 5.1.2* respectively) which concern the define statements related to drive method selection.

- `#define SPEED_OPEN_LOOP`

  Uncomment the above statement when a speed sensor is not being used to control or detect rotor speed.

  Speed open loop control is carried out as explained in *Section 2.5*.

  Fill out `MC_ACIM_Drive_Param.h` and `MC_ACIM_Motor_param.h` (see *Section 5.1.3* and *Section 5.1.2* respectively) which concern the parameters related to drive method selection.

- `#define SPEED_OPEN_LOOP_TACHO_SENSING`

  Uncomment the above statement when a speed sensor is not being used to control rotor speed.

  Speed open loop control is carried out as explained in *Section 2.5*. The tachogenerator signal is processed to check operating conditions or for debug purposes.

  Fill out `MC_tacho_param.h` and `MC_stm8s_tacho_param.h` (see *Section 5.1.4* and *Section 5.1.10*) which concern the define statements related to speed sensing.

  Fill out `MC_ACIM_Drive_Param.h` and `MC_ACIM_Motor_param.h` (see *Section 5.1.3* and *Section 5.1.2* respectively) which concern the define statements related to drive method selection.

5.1.2 ACIM motor define statements: `MC_ACIM_Motor_Param.h`

The `MC_ACIM_Motor_Param.h` header file includes define statements related to the motor. They are:

- `#define MOTOR_POLE_PAIRS`

  Defines the number of motor pole pairs

- `#define MAX_SPEED_RPM`

  Defines the maximum rotor speed, expressed in rpm

- `#define V_F_RATIO`

  This statement is used to set the nominal V/f ratio the drive applies to the motor. It can be expressed as a decimal number which is calculated as a ratio of motor nominal phase voltage (expressed in volts, 0 V to peak) and nominal electrical frequency (expressed in Hertz). This define statement is directly linked to the motor air-gap flux and hence, to the motor magnetizing current (see *Section 2.3: Electromagnetic torque characteristic curve*).

- `#define MAX_V_F_SLIP`

  Defines, in one tenth of a Hertz, the maximum slip frequency (f_{sl}) that can be applied by the drive when accelerating in speed closed loop, V/f mode (see *Section 2.4.1: V/f control and slip regulation*). This define statement is linked, in association with the magnetizing current, to the maximum stator current allowed (see *Section 2.3: Electromagnetic torque characteristic curve*). It must be chosen to avoid running next to the pull-out torque condition.

- `#define MIN_V_F_SLIP`

  Defines, in one tenth of a Hertz, the maximum negative slip frequency (f_{sl}) that can be applied by the drive when decelerating in speed closed loop, V/f mode (see *Section 2.4.1: V/f control and slip regulation*). This parameter is linked, in association with the magnetizing current, to the maximum stator current allowed (see *Section 2.3: Electromagnetic torque characteristic curve*). It must be chosen to avoid running next to the pull-out torque condition.

- `#define MTPA_SLIP`

  Defines, in one tenth of a Hertz, the optimum slip frequency (f_{sl}) that is applied by the drive when in speed closed loop, MTPA mode, fixed slip region (see *Section 2.4.2: Maximum torque per ampere (MTPA) control*).

- `#define STARTUP_V0`

  Defines, in one tenth of a volt, the voltage boost required at startup to compensate for stator voltage drop (see *Section 2.6: Startup strategy*).

5.1.3 ACIM drive control define statements: `MC_ACIM_Drive_Param.h`

The `MC_ACIM_Drive_Param.h` header file includes define statements related to:

- General drive define statements
- Speed closed loop mode and related define statements
- Speed open loop mode and related define statements
- Open loop load compensation mode define statements
- Operation speed checks define statements
- Startup phase related define statements

General drive define statements

- `#define CONTROL_LOOP_PERIOD`

  Defines the speed regulation frequency expressed in milliseconds
- `#define TARGET_ROTOR_SPEED`

  Defines the default mechanical rotor speed set point when in run state. Expressed in rpm.
- `#define PWM_FREQUENCY`

  Defines the PWM switching frequency applied to the power stage. Expressed in Hertz.
- `#define PWM_REFRESH_RATE`

  Defines the repetition rate as a number of full PWM periods, at which point new duty cycle command values are calculated and refreshed in output. The higher the refresh rate the lower the CPU load and the better the output resolution achievable. However, maximum achievable frequency and DC bus ripple compensation are negatively affected (see *Section 4.3.1: Combined utilization of ADC and TIM1 for motor driving*).
- `#define DEAD_TIME_NS`

  Defines the dead time duration, expressed in nanoseconds, to avoid a shoot-through condition.
- `#define DEAD_TIME_COMPENSATION`

  Uncomment this define statement to enable the dead time compensation feature
- `#define BUS_SAMPLING_FREQ`

  Defines the DC bus voltage sampling frequency required. Expressed in Hertz

Speed closed loop mode and related define statements

In conjunction with the configuration selected in `MC_ACIM_conf.h` (*Section 5.1.1*), the following define statements are available if SPEED_CLOSED_LOOP is uncommented:

- `#define CLOSEDLOOP_CONTROLMODE SPEED_CLOSEDLOOP_MTA`

  Uncomment above statement to enable the “most efficient” MTPA control strategy

  Fill the parameters of both the linear regulators for MTPA and V/f control areas (see *Section 2.4.2: Maximum torque per ampere (MTPA) control*).
- `#define CLOSEDLOOP_CONTROLMODE SPEED_CLOSEDLOOP_VF`

  Uncomment above statement to enable the “most dynamic” V/f and slip control strategy.

  Fill the parameters of the V/f control area linear regulator only (*Section 2.4.1: V/f control and slip regulation*).
- `#define CLOSEDLOOP_TUNING`

  Define statement valid only in speed closed loop mode

  Comment above statement to minimize the set of parameters that can be changed, in real-time, by using the LCD user interface. This define statement has a positive effect in reducing the application code size.
- `#define VF_PID_TYPE`

  Define statement valid only in speed closed loop mode

  It is used to configure the type of speed controller used when in the V/f and slip control area. There are two possible settings: *PI* or *PID*. PI sets the proportional integral regulator while PID sets the proportional integral derivative regulator.

- `#define VF_KP`

  Defines the proportional gain of the speed controller (16-bit value, adjustable from 0 to 32767) in the V/f and slip control area.

- `#define VF_KI`

  Defines the integral gain of the speed controller (16-bit value, adjustable from 0 to 32767) in the V/f and slip control area.

- `#define VF_KD`

  Defines the derivative gain of the speed controller (16-bit value, adjustable from 0 to 32767) in the V/f + slip control area. This define statement is used only if the PID controller is selected.

- `#define VF_KP_DIVISOR`

  Defines the scaling factor of the proportional gain of the speed controller (16-bit power-of-two value) in the V/f + slip control area.

- `#define VF_KI_DIVISOR`

  Defines the scaling factor of the integral gain of the speed controller (16-bit power-of-two value) in the V/f + slip control area.

- `#define VF_KD_DIVISOR`

  Defines the scaling factor of the differential gain of the speed controller (16-bit power-of-two value) in the V/f + slip control area. This define statement is used only if the PID controller is selected.

- `#define VF_OUT_MAX`

  This define statement sets the positive saturation value of the speed controller output in the V/f + slip control area. The default value is MAX_V_F_SLIP (see *Section 5.1.2*)

- `#define VF_OUT_MIN`

  This define statement sets the negative saturation value of the speed controller output in the V/f and slip control area. The default value is -MAX_V_F_SLIP (see *Section 5.1.2*). Setting this define statement to zero keeps the controller from applying a negative/braking torque.

- `#define VF_INTERM_MIN`

  This define statement sets the negative saturation value of the speed controller integral action in the V/f + slip control area. The default value is VF_KI_DIVISOR*VF_OUT_MIN.

- `#define VF_INTERM_MAX`

  This define statement sets the positive saturation value of the speed controller integral action in the V/f and slip control area. the default value is VF_KI_DIVISOR*VF_OUT_MIN.

- `#define MTPA_PID_TYPE`

  This define statement is valid only in speed closed loop mode when MTPA control strategy is enabled. It is used to configure the type of speed controller used when in the MTPA control area. There are two possible settings: *PI* or *PID*. PI sets the proportional integral regulator while PID sets the proportional integral derivative regulator.

- `#define MTPA_KP`

  Defines the proportional gain of the speed controller (16-bit value, adjustable from 0 to 32767) in the MTPA control area.

- `#define MTPA_KI`

  Defines the integral gain of the speed controller (16-bit value, adjustable from 0 to 32767) in the MTPA control area.

- `#define MTPA_KD`

  Defines the derivative gain of the speed controller (16-bit value, adjustable from 0 to 32767) in the MTPA control area. This define statement is used only if the PID controller is selected.

- `#define MTPA_KP_DIVISOR`

  Defines the scaling factor of the proportional gain of the speed controller (16-bit power-of-two value) in the MTPA control area.

- `#define MTPA_KI_DIVISOR`

  Defines the scaling factor of the integral gain of the speed controller (16-bit power-of-two value) in the MTPA control area.

- `#define MTPA_KD_DIVISOR`

  Defines the scaling factor of the differential gain of the speed controller (16-bit power-of-two value) in the MTPA control area. This define statement is used only if the PID controller is selected.

- `#define MTPA_OUT_MAX`

  This define statement sets the positive saturation value of the speed controller output in the MTPA control area. The default value is 255 (full DC bus voltage exploitation is allowed).

- `#define MTPA _OUT_MIN`

  This define statement sets the negative saturation value of the speed controller output in the *MTPA* control area. The default value is 0 and only non-negative values are allowed. This is because the controller is unable to apply a negative/braking torque when in the MTPA control area.

- `#define MTPA _INTERM_MIN`

  This define statement sets the negative saturation value of the speed controller integral action in the MTPA control area. The default value is MTPA_KI_DIVISOR*MTPA_OUT_MIN.

- `#define MTPA _INTERM_MAX`

  This define statement sets the positive saturation value of the speed controller integral action in the MTPA control area. The default value is MTPA_KI_DIVISOR*MTPA_OUT_MAX.

Speed open loop mode and related define statements

In conjunction with the configuration selected in `MC_ACIM_conf.h` (*Section 5.1.1*), the following define statements are available if `SPEED_OPEN_LOOP` or `SPEED_OPEN_LOOP_TACHO_SENSING` are uncommented:

- `#define OPENLOOP_CONTROLMODE SPEED_OPENLOOP`

  Uncomment above statement to enable a pure speed open loop V/f control with a real-time adjustable V/f ratio and a foreseen slip frequency (see *Section 2.5*).

- `#define OPENLOOP_CONTROLMODE SPEED_OPENLOOP_LOAD_COMPENSATION`

  Uncomment above statement to enable a speed open loop V/f control with automatic load compensation (see *Section 2.5.1*).

  Note: A look-up table that matches the foreseen load characteristic curve must exist. Such a look up table is calculated automatically by the STM8S_MC_Firmware_Library builder (see *Section A.4*).

- `#define OPEN_LOOP_ACCELERATION_SLOPE`

  Defines the acceleration slope to be imposed during target speed variations, both in start and run state. Expressed in rpm/s.

- `#define OPENLOOP_SLIP`

  Defines, in one tenth of a Hertz, the default slip frequency (f_{sl}) that is applied by the drive during run state if the selected operating mode is speed open loop with no load compensation.

Open loop load compensation mode define statements

The define statements of this section (SEGDIV, ANGC, OFST) are automatically worked-out by the *STM8S_MC_Firmware_Library builder*. Please refer to *Section 2.5.1: Load compensation* for a more detailed explanation of speed open loop, load compensation mode.

Operation speed check define statements

- `#define MIN_RUN_SPEED`

  This define statement is valid in speed closed loop mode, speed open loop mode, and tacho sensing mode.

  It defines the minimum speed below which speed feedback is unrealistic in the application (in run state). Expressed in rpm. This allows a low frequency to be discerned. This value is set to 200 rpm by default and depends on sensor and signal conditioning stage characteristics. Typically, the tacho signal is too weak at very low speeds to trigger input capture on the MCU.

- `#define MAX_SPEED_FEEDBACK`

  This define statement is valid in speed closed loop or in speed open loop with tacho sensing mode. It defines the maximum speed above which speed feedback is unrealistic in the application (in run state). It is expressed in rpm.

- `#define STALL_SPEED`

  This define statement is valid in speed closed loop mode, speed open loop mode, and tacho sensing mode.

  It defines the maximum motor speed allowable to begin the startup procedure (see *Section 2.6*). Expressed in rpm.

Startup phase related define statements

- `#define STARTUP_VAL_SPEED`

  In speed closed loop mode or in speed open loop mode with tacho sensing mode, this define statement configures the minimum measured rotor speed at which the startup phase is validated, so that the control can switch to run state (see *Section 2.6: Startup strategy*).

  In speed open loop mode (without tacho sensing mode), it defines the target speed of the startup phase acceleration. This parameter defines the maximum duration of the startup acceleration in conjunction with the `OPEN_LOOP_ACCELERATION_SLOPE` parameter. It is expressed in rpm.

- `#define STARTUP_DURATION`

  In speed closed-loop mode, this define statement configures the maximum duration of the startup procedure expressed in milliseconds. This is the longest time the controller waits for the exit condition (`STARTUP_VAL_SPEED` define statement) to be verified.

  In speed open loop mode and tacho sensing, it defines the duration of the startup procedure expressed in milliseconds. At the end of this duration the controller checks if the exit condition (`STARTUP_VAL_SPEED` define statement) is verified.

  In speed open loop mode, it defines the duration of the startup procedure expressed in milliseconds. At the end of this duration, unless other errors have occurred, the controller switches from start to run state.

- `#define STARTUP_V_F_RATIO`

  This define statement can be expressed as a decimal number. It is used to set the V/f ratio the drive applies to the motor during the startup phase. STARTUP_V_F_RATIO can be calculated as a ratio of motor phase voltage (expressed in volts, 0 V to peak) and electrical frequency (expressed in Hertz). If the selected operating mode is speed closed loop, this define statement defines maximum applicable V/f ratio. However, if in speed open loop, it defines the fixed V/f ratio to be applied. When the motor and the inverter being used are working in conditions where they can withstand a current overload, this define statement (in association with `#define STARTUP_SLIP`) can help produce the extra starting torque many applications require (see *Section 2.6: Startup strategy*).

- `#define STARTUP_SLIP`

  Defines, in one tenth of a Hertz, the slip frequency (f_{sl}) that is applied by the drive during the startup procedure. When the motor and the inverter being used are working in conditions where they can withstand a current overload, this define statement (in association with `#define STARTUP_V_F_RATIO`) can help produce the extra starting torque many applications require (see *Section 2.6: Startup strategy*).

5.1.4 Tacho sensor define statements: `MC_tacho_param.h`

The `MC_tacho_param.h` header file includes define statements related to the tachogenerator sensor. These settings are used only in sensored configurations (speed closed loop or speed open loop and tacho sensing modes). In conjunction with these define statements, the user should fill the `MC_stm8s_tacho_param.h` header file.

The tacho sensor define statements comprise:

- `#define TACHO_PULSE_PER_REV`

  Defines the number of pulses per revolution given by the tachogenerator

- `#define TACHO_PULSE_AVERAGED`

  Defines the target number of tacho periods to be captured during each speed measurement cycle. This define statement determines the (fractional) number of rotor revolutions (TACHO_PULSE_AVERAGED / TACHO_PULSE_PER_REV) over which the average motor speed is calculated.

5.1.5 Control stage define statements: `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 define statements: `MC_PowerStage_Param.h`

The `MC_PowerStage_param.h` header file includes define statements 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.

The power stage define statements comprise:

- `#define RS_M`

  This define statement is not used in the STM8S ACIM control software library V1.0

- `#define AOP`

  This define statement is not used in the STM8S ACIM control software library V1.0

- `#define DISSIPATIVE_BRAKE`

  Uncomment this define statement to enable the dissipative brake function (see *Section 4.3.3*). 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`

  `#define MIN_BUS_VOLTAGE`

  These two values (expressed in volts) 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 is handled by activating the brake resistor, and the corresponding error message is not issued.

- `#define NTC_THRESHOLD_C`

  `#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`

  `#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 41*).

- `#define BKIN_POLARITY`

  When the firmware runs on a customized hardware, this define statement allows the polarity of the break input to be configured. 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`

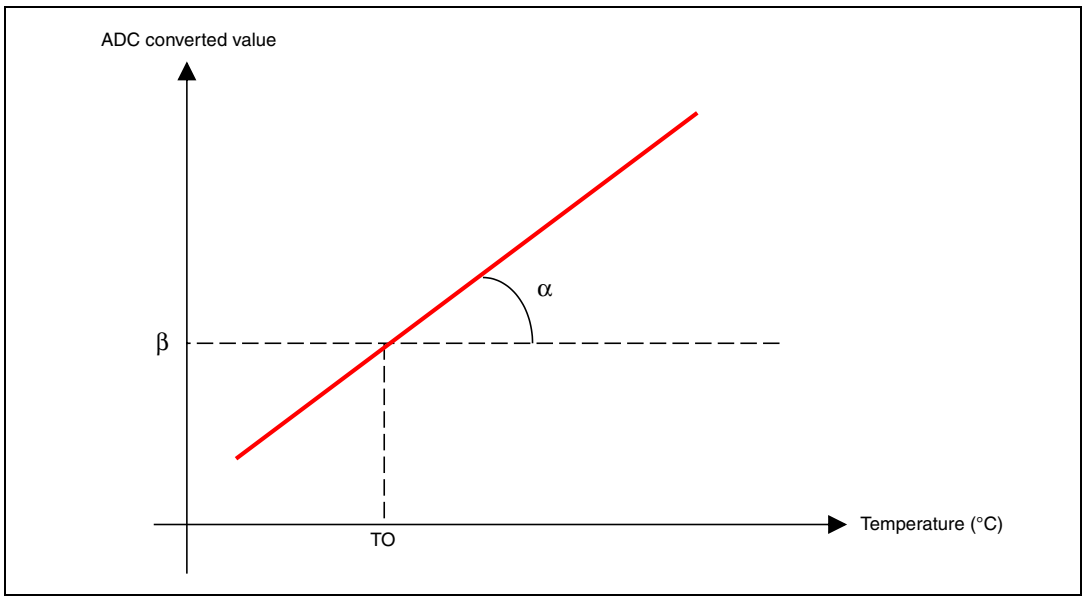
  `#define PWM_W_LOW_SIDE_POLARITY`

  `#define PWM_W_HIGH_SIDE_POLARITY`

  When the firmware runs on a customized hardware, these define statements allow the polarity of the PWM output to be configured. 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 the status of the PWM output during the idle state to be configured. 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 disable the complementary output control of TIM1. In this case, the required deadtime should be managed by the power device drivers.

Figure 41. Transduction curve between the temperature sensor and the ADC converted



This curve represents *Equation 17*, where α is defined using `TEMP_SENS_ALPHA`, β is defined using `TEMP_SENS_BETA` and T0 is defined using `TEMP_T0`.

Equation 17 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 following define statement strictly related to the microcontroller and its peripherals. For this reason the name of the microcontroller is present in the name of the file.

- `#define STM8_FREQ_MHZ`

  This define statement is used to set the CPU frequency in mega Hertz. Either 16 MHz or 24 MHz can be chosen.

  Note: If 24Mhz is chosen, the firmware is configured to use the external oscillator with 1 wait state for the Flash latency. Otherwise the internal oscillator with 0 wait state is used.

5.1.8 Microcontroller specific ACIM drive define statements: `MC_stm8s_ACIM_param.h`

The *MC*`_stm8s_ACIM_param.h` header file contains the following define statements related to the ACIM drive.

- `#define BUS_ADC_CHANNEL`

  `#define BUS_ADC_PORT`

  `#define BUS_ADC_PIN`

  The ADC channel to be used for the bus DC sampling and its related GPIO port and pin can be configured using the above define statements if the firmware runs on customized hardware.

  Note: The PIN description can be read from the appropriate STM8S datasheets. The pins should not be used for other purpose inside the firmware and should never be configured as outputs.

- `#define HEATSINK_ADC_CHANNEL`

  `#define HEATSINK _ADC_PORT`

  `#define HEATSINK _ADC_PIN`

  The ADC channel to be used for the heatsink temperature sampling and its related GPIO port and pin can be configured using the above define statements if the firmware runs on customized hardware.

  Note: The PIN description can be read from the appropriate STM8S datasheets. The pins should not be used for other purpose inside the firmware and should never be configured as outputs.

- `#define USER1_ADC_CHANNEL`

  `#define USER1_ADC_PORT`

  `#define USER1_ADC_PIN`

  The ADC channel to be used for a user defined A/D conversion and its related GPIO port and pin can be configured using the above define statements if the firmware runs on customized hardware.

  Note: The PIN description can be read from the appropriate STM8S datasheets. The pins should not be used for other purpose inside the firmware and should never be configured as outputs.

- `#define SINE3RDHARM`

  This define statement is the sine wave reference look-up table that is stored in the Flash memory. It contains third harmonic by default which allows approximately 15% more voltage to be obtained on a motor from a given DC bus compared to pure sine (see *Section 2.7: Three-phase PWM sine wave and third harmonic generation*).

- `#define BUSVOLT_BUFFER_SIZE`

  Defines the buffer size utilized for averaging bus voltage measurement. Maximum buffer size is 255.

  Note: The DC bus voltage is sampled at a frequency defined by BUS_SAMPLING_FREQ (`MC_ACIM_Drive_Param.h`).

- `#define HEATSINK_SAMPLING_FREQ`

  Defines the heatsink temperature sampling frequency required. Expressed in Hertz.

- `#define USERADC_SAMPLING_FREQ`

  Defines the user-defined AD conversion sampling frequency required. Expressed in Hertz.

- `#define STARTUP_ANGLE`

  Defines the initial angle of the three-phase voltage system. This angle is expressed in unsigned 16-bit format, where 0-65535 corresponds to 0-2 π radians.

5.1.9 Port pin definition define statements: 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

  #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 correct 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 correct 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 4.3.3*).

  In the first define statement, set the port by replacing the x character in the GPIOx string with the correct 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 correct number. For instance set GPIO_PIN_1 if pin 1 is used.

5.1.10 Tacho param microcontroller interfaces: `MC_stm8s_tacho_param.h`

The `MC_stm8s_tacho_param.h` header file contains the following define statements related to low-level operations of the tachogenerator signal processing:

- `#define TACHO_TIMERx_CHANNELy`

  The default choice is TACHO_TIMER2_CHANNEL2 (the tacho signal is routed to Tim2, input capture 2 in the STM8/128-MCKIT motor control starter kit). If the firmware runs on customized hardware, the correct statement, which reflects the actual timer/input capture selection, should be uncommented.

- `#define TACHO_IC_PORT`
  `#define TACHO_IC_PIN`

  The corresponding GPIO port and pin must be declared for the timer/input capture selected for tacho signal processing (see *Section 5.1.9*).

  Note: The PIN description can be read from the appropriate STM8S datasheets. The pins should not be used for other purpose inside the firmware and should never be configured as outputs. GPIO alternate function remapping of TIM2/TIM3 channels should also be considered.

- `#define IC_FILTER_DURATION`

  Defines the length of the digital filter to be applied at the input stage of the selected timer channel.

  The digital filter contains an event counter in which N events are needed to validate a transition of the output. Valid filter values are summarized in *Table 10*.

Table 10. Tacho input capture filter duration

#define IC_FILTER_DURATION	Sampling frequency	Events N	µsec filter F_CPU 16 Mhz	µsec filter F_CPU 24 Mhz
0x00	No filter	No filter	0.0625	0.0417
0x01	F_CPU	2	0.1250	0.0833
0x02	F_CPU	4	0.2500	0.1667
0x03	F_CPU	8	0.5000	0.3333
0x04	F_CPU/2	6	0.7500	0.5000
0x05	F_CPU/2	8	1.0000	0.6667
0x06	F_CPU/4	6	1.5000	1.0000
0x07	F_CPU/4	8	2.0000	1.3333
0x08	F_CPU/8	6	3.0000	2.0000
0x09	F_CPU/8	8	4.0000	2.6667
0x0a	F_CPU/16	5	5.0000	3.3333
0x0b	F_CPU/16	6	6.0000	4.0000
0x0c	F_CPU/16	8	8.0000	5.3333
0x0d	F_CPU/32	5	10.0000	6.6667
0x0e	F_CPU/32	6	12.0000	8.0000
0x0f	F_CPU/32	8	16.0000	10.67

- `#define TACHO_TIMER_ARR`

  Defines the reload register value for the timer that handles the tachogenerator signal. This define statement is involved in the automatic prescaler adaptation to the motor speed according to *Equation 18* approximated to the next greatest power of two.

- `#define MAX_ERROR_NUMBER`

  Defines the maximum number of times the timer (which is reading the tacho signal period) can reach the overflow without having captured a number of tacho pulses equal to TACHO_PULSE_AVERAGED (`MC_tacho_param.h`). At this point, a fault message is issued (SPEED FEEDBACK).

Equation 18

$$\text{Prescaler} = \frac{\text{cpu\_frequency} \times \text{TACHO\_PULSE\_AVERAGED} \times 60}{\text{TACHO\_TIMER\_ARR} \times \text{motor\_speed(RPM)} \times \text{TACHO\_PULSE\_PER\_REV}}$$

The refresh rate of the low-level section of the speed measurement process can also be calculated, using *Equation 19* below and considering the formula above with a given motor speed.

Equation 19

$$\text{Refresh rate (s)} = \frac{\text{Prescaler}}{\text{cpu\_frequency}} \times \text{TACHO\_TIMER\_ARR}$$

- `#define MAX_PRESCALER`

  Defines the maximum prescaler value for the timer that handles the tachogenerator signal.

  Effectively, this defines the lowest speed that can be measured, according to *Equation 20*.

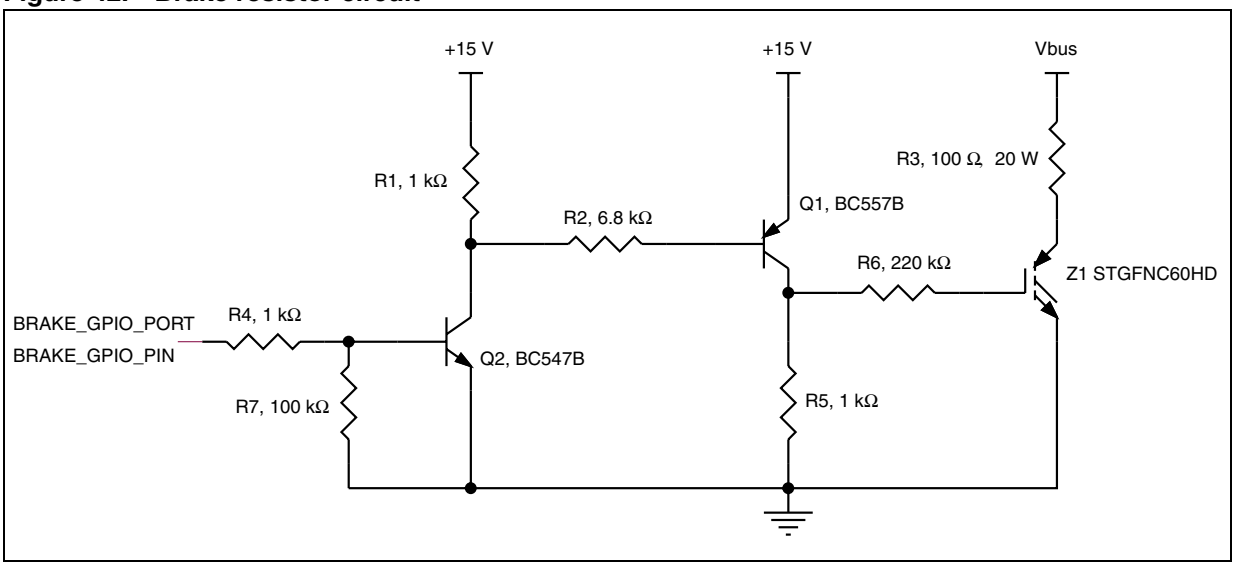
Equation 20

$$\text{Min speed (RPM)} = \frac{\text{cpu\_frequency(Hz)} \times 60 \times \text{TACHO\_PULSE\_AVERAGED}}{\text{MAX\_PRESCALER} \times \text{TACHO\_TIMER\_ARR} \times \text{TACHO\_PULSE\_PER\_REV}}$$

5.2 Setting up the system when using a brake resistor

To make the STM8/128-MCKIT motor control starter kit board suitable for the management of a brake resistor, some additional components must be soldered onto its wrapping area. *Figure 42* gives an example of the circuit to be used for hardware implementation of the brake.

Figure 42. Brake resistor circuit



1. The size of the resistor in terms of both resistance and sustainable power should be carefully dimensioned.
2. In the STM8/128-MCKIT motor control starter kit, pin 23 of the MC connector (J7) that carries the signal for brake implementation is positioned close to the wrapping area.

6 Module description

The current firmware is comprised of a set of modules that are subdivided logically into three groups:

- High-level MC modules
- Low-level MC modules
- Standard library

The last two groups contain functions related to the microcontroller (see *Section 4.3: Low-level control*) while the first group contains hardware independent functions as described in *Section 4.4: High-level control*.

6.1 High-level modules

The high-level MC modules are stored in the **MC_FWLIB_SCALAR** folder and include the following:

- `MC_ACIM_Drive`: This module contains all functions related to the electrical drive and engine control (see *Section 4.4.3: ACIM scalar control*).
- `MC_tacho.c`: This module contains all functions related to rotor speed measurement via a tachogenerator sensor (see *Section 4.4.4: Tachogenerator signal reading*).
- `MC_ACIM_Motor`: This module is the holder of the drive structure. It contains all functions used to interact (get or set parameters) with the ACIM drive structure or to get a reference of that structure for other modules.
- `MC_ACIM_User_Interface`: This module is the holder of the user interface specific for the ACIM drive. It contains a function used to provide the reference of that structure for other modules. Interaction with the structure is managed by `MC_User_Interface.c`.
- `MC_User_Interface`: This module is used to manage the user interface (see *Section 3.1: ACIM user interface*) which interacts with `MC_keys` and `MC_display` modules. The user interface is specifically defined according to the drive of the `MC_ACIM_User_Interface` module.
- `MC_dev`: This module is an interface between the high-level and low-level modules. For example, it is used to initialize the low-level modules by calling each specific initialization function (dev_clkInit, dev_portInit, ...).
- `MC_display`: This module is used to manage user display information. In the firmware the display is implemented using an LCD screen of 15 rows x 2 lines.

  Note: This module is developed over the low-level virtual I/O functions (see Section 4.2: Library architecture).
- `MC_Keys`:

  This module manages the button and joystick (4 directions plus a center button). It is developed over the low level virtual I/Os functions (see *Section 4.2: Library architecture*).

- `MC_pid_regulators`: This module is used to manage all the regulators needed by the application. They can be proportional integral derivative (PID) or proportional integral regulators (PIR). The `MC_pid_regulator` module is used to instance a regulator structure and to execute it.
- `MC_StateMachine`: This module is used to manage the main application state machine. The only interface with it, is 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 4.4.1: Virtual timers*.
- `Main`: This is the main application firmware module. In the actual implementation it is used to execute the state machine in an infinite loop.

6.2 Low-level modules

The low-level MC modules are stored in the folder **STM8_MC_FRAMEWORK** and include the following:

- `MC_stm8s_ACIM_drive`: This module contains all functions related to the low-level electrical drive and engine control.
- `MC_stm8s_tacho.c`: This module contains all functions related to the low-level shell of the rotor speed measurement method via a tachogenerator sensor (see *Section 4.3.2: Tachogenerator signal reading*).
- `MC_stm8s_ACIM_it`: This module contains all the interrupt service routines defined inside the interrupt vector that are NOT used by the firmware. The interrupt service routine used by the firmware is defined inside each module that uses it.
- `MC_stm8s_clk`: This module manages the function responsible for setting the microcontroller clock.
- `MC_stm8s_DAC`: This module manages the digital to analog function implemented for debugging purposes. It use TIM3 as described in *Section A.1*.
- `MC_stm8s_display`: This module contains the low-level functions that interact with the LCD display. It has been developed on top of the `mono_lcd` module. Exported functions include dev_displayInit (used to configure the hardware for the LCD visualization), dev_displayClear (used to clear the LCD screen), dev_displayFlush (displays the data that have already been formatted by the `MC_display` module), dev_displayPrintch (used to refresh the cursor.)
- `MC_stm8s_keys`: This module is used to initialize the hardware (dev_keysInit)
- `MC_stm8s_port`: This module is used to initialize the GPIOs of the microcontroller required for configuration.
- `MC_stm8s_vtimer`: This module is used to implement management of the low-level virtual timer so it is used only for initialization of the hardware (dev_vtimerInit) and contains the interrupt service routine of TIM4.
- `vdev_ios`: This module is used to manage the low-level input output functionality (see *Table 3: Virtual registers*).

Appendix A Additional information

A.1 DAC configuration

In the current firmware library, the DAC functionality is implemented using two output compare channels (PD2 and PD0 pins) of TIM3 and modulating the duty cycle of the generated 62.5 kHz PWM signal. To filter the generated signals without introducing significant delays on the waveforms, use a first order low-pass filter (for example, with a 1 kΩ resistor and a 33 nF capacitor).

In the ACIM drive firmware, both DAC outputs are used. The first, monitors the stator voltage amplitude and the second monitors the slip frequency controller outputs.

It is also possible to use the DAC outputs to monitor two user defined variables (example, user_var1 and user_var2) by modifying the statements below that are present in the `MC_ACIM_Drive.c` file:

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

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

Note: It is not possible to use the DAC functionality with the dissipative brake function.

See *Section 5.1.5: Control stage define statements: MC_ControlStage_param.h* for details on how to enable he DAC functionality.

A.2 Motor control related CPU load

Table 11 gives the estimated execution time for a set of ACIM motor control library functions, while *Table 12* give an estimate of the CPU load during ACIM motor control software execution.

Table 11. Example of ACIM motor control function execution time

Function	Source file	Estimated execution time (µs)	Priority level
ACIM_Drive()	`MC_ACIM_Drive.c`	250	1
ADCx_IRQHandler()	`MC_stm8s_ACIM_drive.c`	13.7	3
TIM1_UPD_OVF_TRG_BRK_IRQHandler()	`MC_stm8s_ACIM_drive.c`	4.4	3
TIMx_UPD_OVF_TRG_BRK_IRQHandler()(1)	`MC_stm8s_tacho.c`	3.4	1
TIMx_CAP_COM_IRQHandler()(1)	`MC_stm8s_tacho.c`	2.4	2

1. See *Figure 38: Tachogenerator sensing low-level module*

Table 12. CPU load resulting from motor control

Parameter	Value	CPU load (in %)
Speed loop control frequency	200 Hz	5
PWM and sine waves control frequency	8 KHz	14.5
Motor speed	3000 RPM	0.1
Number of tachogenerator pole pairs	8	
TACHO_PULSE_AVERAGED (see *Section 5.1.4: Tacho sensor define statements: MC_tacho_param.h*)	3	
Total estimated CPU load		**19.6**

A.3 References

1. Cacciato M, Consoli A, Scarcella G, Scelba G, and Testa A (2006) Efficiency optimization techniques via constant optimal slip control of induction motor drives, *Proceedings of SPEEDAM*.
2. Mohan N, Undeland TM, and Robbins WP (1995) Power electronics: Converters, Applications and Design. *Wiley, second edition*.
3. Fitzgerald AE, Kingsley Jr. C, and Umans SD (1990) Electric Machinery, 5th edition, *McGraw-Hill,* New York.
4. Chapman SJ (1999) Electric Machinery Fundamentals, 3rd edition. *McGraw-Hill*, NewYork.
5. Krause PC, Wasynczuk O, and Sudhoff SD (2002) Analysis of Electric Machinery and Drive Systems. *Wiley-IEEE Press*.

A.4 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 13. Document revision history

Date	Revision	Changes
01-Jul-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