PMSMLPC55S36EVK

MCUXpresso SDK Field-Oriented Control (FOC) of 3-Phase PMSM and BLDC Motors

Rev. 2 — 31 July 2023 User guide

Document information

Information	Content
Keywords	PMSMLPC55S36EVK , PMSM, FOC, MCAT, MID, Motor control, Sensorless control, Speed control, Servo control, Position control
Abstract	This user guide describes the implementation of the motor-control software for 3-phase Permanent Magnet Synchronous Motors.

1 Introduction

SDK motor control example user guide describes the implementation of the motor-control software for 3-phase Permanent Magnet Synchronous Motors (PMSM) using following NXP platforms:

- LPCXpresso55S36 (LPCXpresso55S36)
- Freedom Development Platform for Low-Voltage, 3-Phase PMSM Motor Control (FRDM-MC-LVPMSM)

The document is divided into several parts. Hardware setup, processor features, and peripheral settings are described at the beginning of the document. The next part contains the PMSM project description and motor control peripheral initialization. The last part describes user interface and additional example features.

Available motor control examples types with supported motors, and possible control methods are listed in Table 1.

Table 1. Available example type, supported motors and control methods

Example type	Supported motor	Possible control methods in SDK example				
		Scalar and Voltage	Current FOC (Torque)	Sensorless Speed FOC	Sensored Speed FOC	Sensored Position FOC
pmsm_enc	Linix 45ZWN24-40 (default motor)	✓	✓	✓	N/A	N/A
	Teknic M-2310P (with ENC)	✓	✓	✓	✓	✓
pmsm_enc_iopamp	Linix 45ZWN24-40 (default motor)	✓	✓	✓	N/A	N/A
	Teknic M-2310P (with ENC)	✓	✓	✓	✓	✓
pmsm_enc_dual	Linix 45ZWN24-40 - motor M2	✓	✓	✓	N/A	N/A
	Teknic M-2310P (with ENC) - motor M1	✓	✓	✓	✓	✓
pmsm_periph_framework	Linix 45ZWN24-40 - motor M2	N/A	N/A	N/A	N/A	N/A
	Teknic M-2310P (with ENC) - motor M1	N/A	N/A	N/A	N/A	N/A

SDK motor control example description:

- **pmsm_enc** - pmsm example uses float arithmetic, the example contains sensored and also sensorless field oriented vector control (FOC). This example can be used for sensor and sensorless motor control application both. Default motor configuration is tuned for the Linix 45ZWN24-40 motor.
- **pmsm_enc_iopamp** - pmsm example uses float arithmetic, the example contains sensored and also sensorless field oriented vector control (FOC). This example can be used for sensor and sensorless motor control application both. Default motor configuration is tuned for the Linix 45ZWN24-40 motor. Application use internal operational amplifier (IOPAMP) on the development board. FRDM-MC-LVPMSM board must be modified to properly application functionality (see Figure 7). The current scale depends on the setting IOPAMPs.
- **pmsm_enc_dual** - pmsm example uses float arithmetic, the example contains sensored and also sensorless field oriented vector control (FOC). This example can be used for sensor and sensorless dual motor control application both. Default motor configuration is tuned for the Linix 45ZWN24-40 motor.

- **pmsm_periph_framework** - this example contains pins setting (e.g. PWM outputs, ADC inputs) and peripherals setting (e.g. ADC, timers, comparator). Example does not contain algorithms or state machine for motor control application, so spinning with the motor is not possible in the default example.

The SDK motor control example contains several additional features:

- **FreeMASTER** pmsm_float_enc.pmpx project provides a simple and user-friendly way for algorithm tuning, software control, debugging, and diagnostics.
- **MCAT** - Motor Control Application Tuning page based on the FreeMASTER runtime debugging tool.
- **MID** - Motor parameter identification.

The control software and the PMSM control theory, in general, are described in *Sensorless PMSM Field-Oriented Control (FOC)* (document DRM148).

2 Hardware setup

2.1 Linix 45ZWN24-40 motor

The Linix 45ZWN24-40 motor is a low-voltage 3-phase permanent-magnet motor with hall sensor used in PMSM applications. The motor parameters are listed in Table 2.

Table 2. Linix 45ZWN24-40 motor parameters

Characteristic	Symbol	Value	Units
Rated voltage	Vt	24	V
Rated speed	-	4000	RPM
Rated torque	T	0.0924	Nm
Rated power	P	40	W
Continuous current	Ics	2.34	A
Number of pole-pairs	pp	2	-

Figure 1. Linix 45ZWN24-40 permanent magnet synchronous motor

The motor has two types of connectors (cables). The first cable has three wires and is designated to power the motor. The second cable has five wires and is designated for the hall sensors' signal sensing. For the PMSM sensorless application, only the power input wires are needed.

2.2 Teknic M-2310P motor

The Teknic M-2310P-LN-04K motor is a low-voltage 3-phase permanent-magnet motor used in PMSM applications. The motor has two feedback sensors (hall and encoder). For information on the wiring of feedback sensors, see the data sheet on the manufacturer webpage. The motor parameters are listed in Table 3.

Table 3. Teknic M-2310P motor parameters

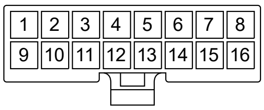
Characteristic	Symbol	Value	Units
Rated voltage	Vt	40	V
Rated speed	-	6000	RPM
Rated torque	T	0.247	Nm

Table 3. Teknic M-2310P motor parameters...*continued*

Characteristic	Symbol	Value	Units
Rated power	P	170	W
Continuous current	Ics	7.1	A
Number of pole-pairs	pp	4	-

Figure 2. Teknic M-2310P permanent magnet synchronous motor

For the sensorless control mode, you only need the power input wires. If used with the hall or encoder sensors, connect the sensor wires to the NXP Freedom power stage.



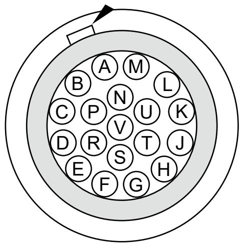
(Wire entry view)

Motor phases

Pin	Color	Signal	Pin	Color	Signal
1	DRAIN x3	P DRAIN	9	16AWG BLK	PHASE R
2	N/A	N/A	10	16AWG RED	PHASE S
3	GRN	COMM S-T	11	16AWG WHT	PHASE T
4	GRN/WHT	COMM R-S	12	RED	+5VDC IN
5	GRY/WHT	COMM T-R	13	BRN	ENC 1
6	DRAIN x1	E DRAIN	14	ORN	ENC B
7	BLK	GND	15	BLU	ENC A
8*	BLU/WHT	ENC A-	16*	ORN/WHT	ENC B-

Encoder wires

Figure 3. Teknic motor connector type 1



Motor phases (Mating face shown) Encoder wires

Pin	Color	Signal	Pin	Color	Signal
R	DRAIN x3	P DRAIN	L	GRY/WHT	COMM T-R
C	16AWG RED	PHASE S	U	BRN	ENC I
D	16AWG WHT	PHASE T	G	GRN	COMM S-T
B	16AWG BLK	PHASE R	T	RED	+5VDC IN
J	BLU	ENC A	F*	ORN/WHT	ENC B~
K*	BLU/WHT	ENC A~	V	ORN	ENC B
H	GRN/WHT	COMM R-S	M	DRAIN x1	E DRAIN
S	BLK	GND			

Figure 4. Teknic motor connector type 2

2.3 FRDM-MC-LVPMSM

In a shield form factor, this evaluation board effectively turns an NXP Freedom development board or an evaluation board into a complete motor-control reference design. It is compatible with existing NXP Freedom development boards and evaluation boards. The Freedom motor-control headers are compatible with the Arduino R3 pin layout.

The FRDM-MC-LVPMSM low-voltage, 3-phase Permanent Magnet Synchronous Motor (PMSM) Freedom development platform board has a power supply input voltage of 24 VDC to 48 VDC with reverse polarity protection circuitry. The auxiliary power supply of 5.5 VDC is created to supply the FRDM MCU boards. The output current is up to 5 A RMS. The inverter itself is realized by a 3-phase bridge inverter (six MOSFETs) and a 3-phase MOSFET gate driver. The analog quantities (such as the 3-phase motor currents, DC-bus voltage, and DC-bus current) are sensed on this board. There is also an interface for speed and position sensors (encoder, hall). The block diagram of this complete NXP motor-control development kit is shown in Figure 1.

Figure 5. Motor-control development platform block diagram

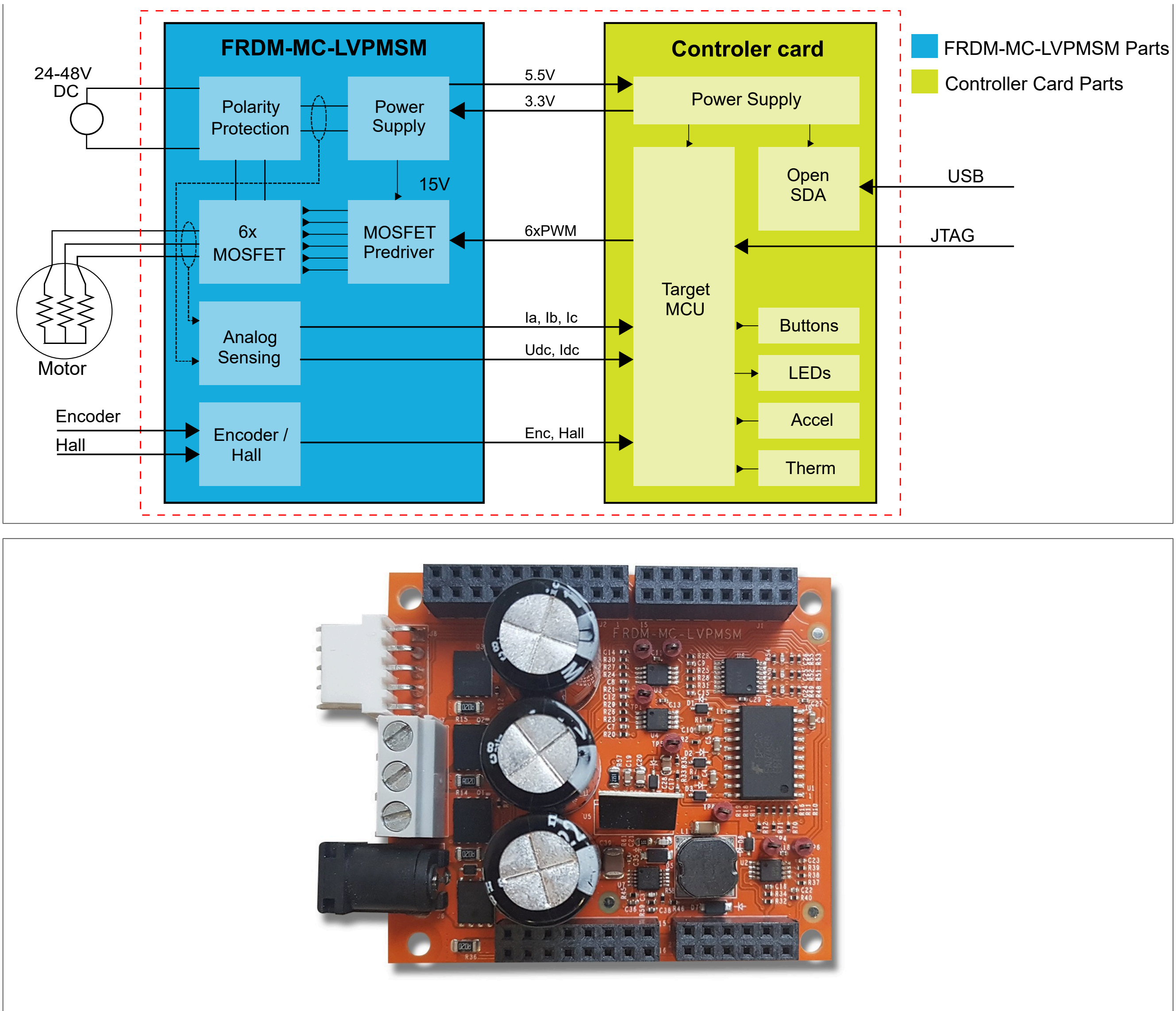


Figure 6. FRDM-MC-LVPMSM

The FRDM-MC-LVPMSM board does not require a complicated setup. For more information about the Freedom development platform, see www.nxp.com.

Note:

There might be a wrong FRDM-MC-LVPMSM series in the market (series VV19520XXX). This series is populated with 10 mOhm shunt resistors and noisy operational amplifiers which affect phase current measurement. The `mc_pmsm` example is tuned for original FRDM-MC-LVPMSM board with 20 mOhm shunt resistors.

2.3.1 Using FRDM with IOPAMPs

If the example with internal operational amplifier (IOPAMP) is intended to be used, Freedom development board must be modified to properly application functionality. The current scale can be different from original FRDM-MC-LVPMSM board. For a correct connection, remove from the FRDM-MC-LVPMSM board resistors R23, R24, R25, R26, R27 and R28. Furthermore connect signals on this board as follow:

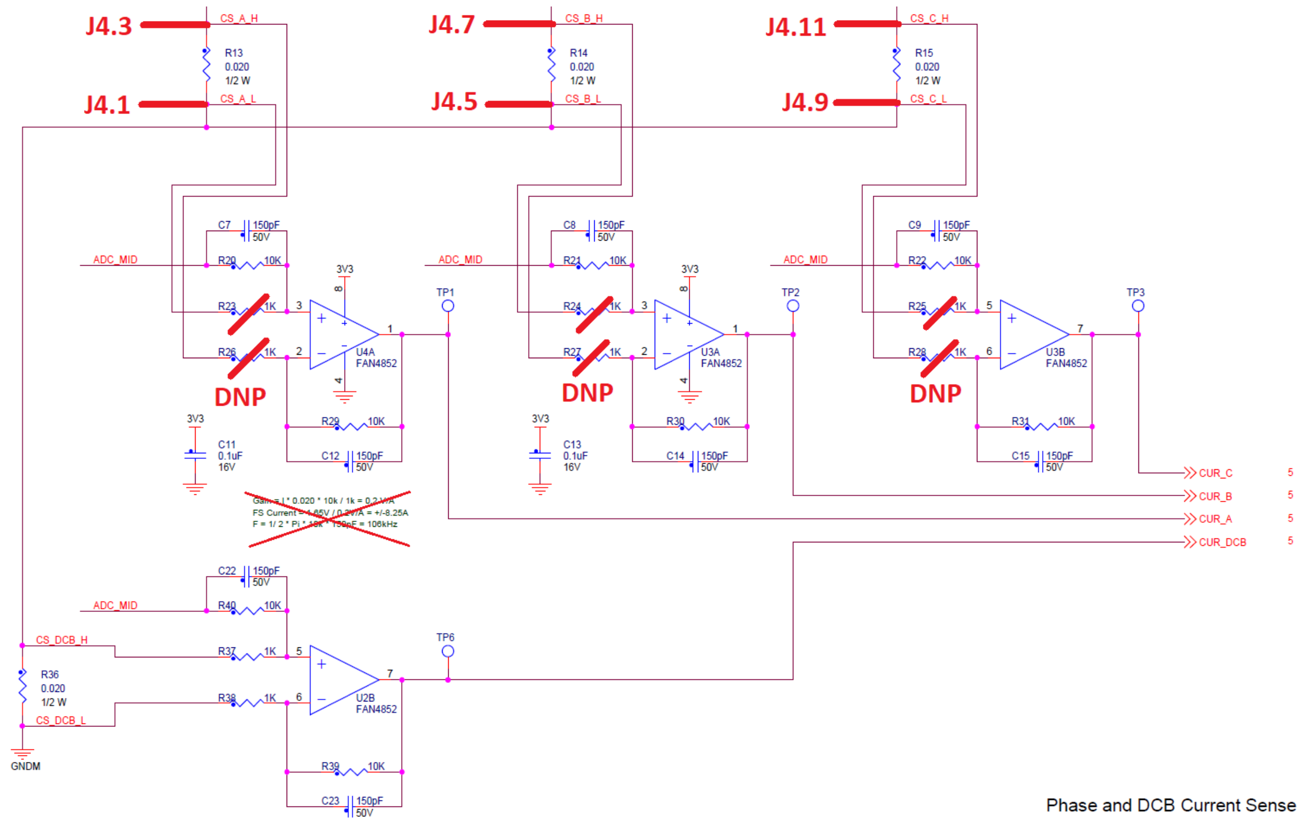


Figure 7. FRDM-MC-LVPMSM modification for IOPAMP application

2.4 LPC55S36-EVK

The LPCXpresso55S36 development board is an ideal platform for evaluation and development with the LPC55S36 MCU based on the Arm Cortex-M33 architecture. The Arm Cortex-M33 core operates at up to 150 MHz. The board includes the high-performance on-board debug probe, audio subsystem, and accelerometer, with a possibility to add off-the-shelf add-on boards for networking, sensors, displays, and other interfaces. For the motor-control application can be used Motor 1 or Motor 2 connector. Configure the jumpers to the default settings as described in the schematic document for LPC55S36-EVK board revision D, which can be found in LPC5500 Series. For more information about LPC55S36-EVK board navigate to the user manual `LPC55S36-EVKUM`.

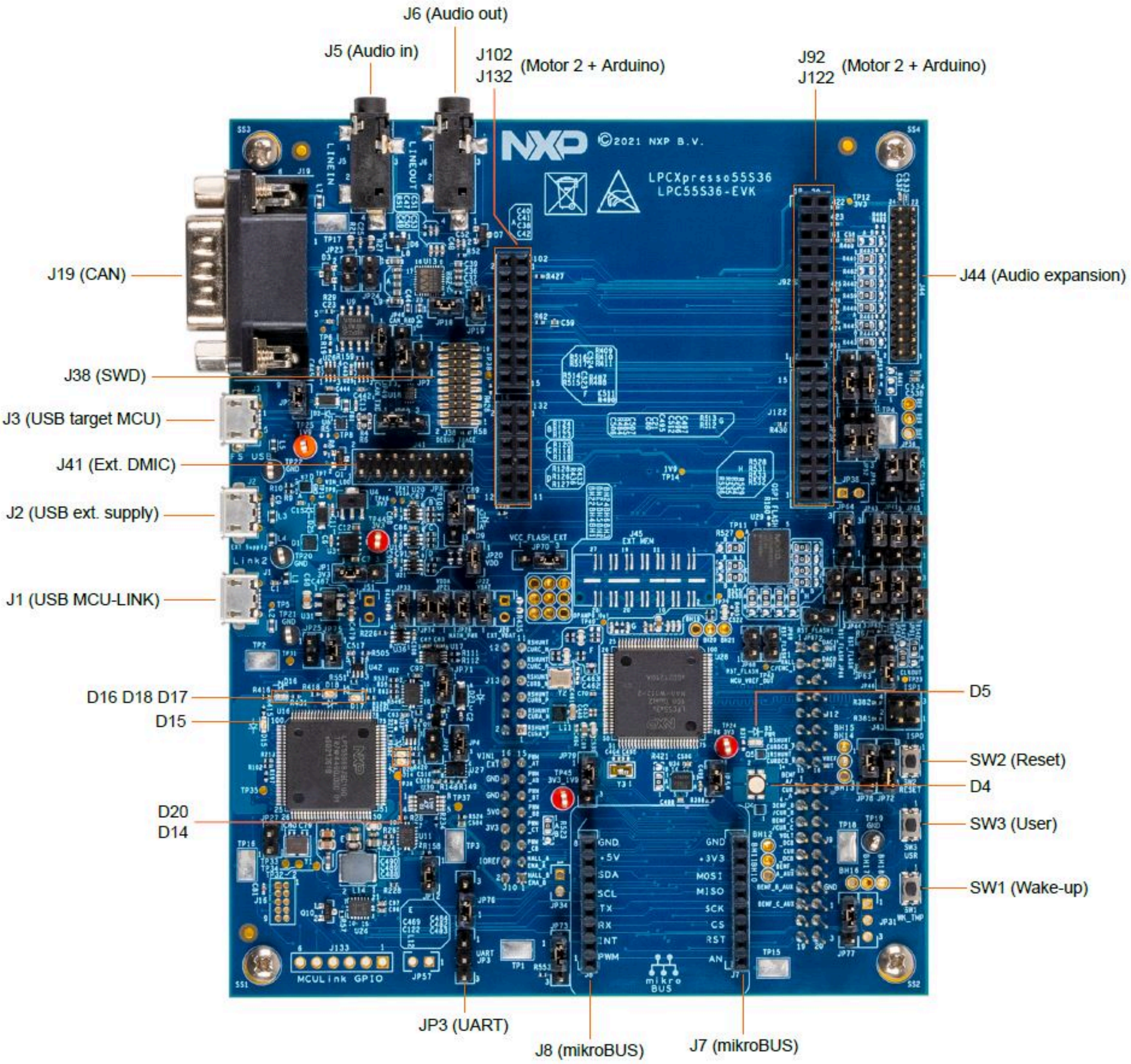


Figure 8. LPC55S36-EVK board (top-side view)

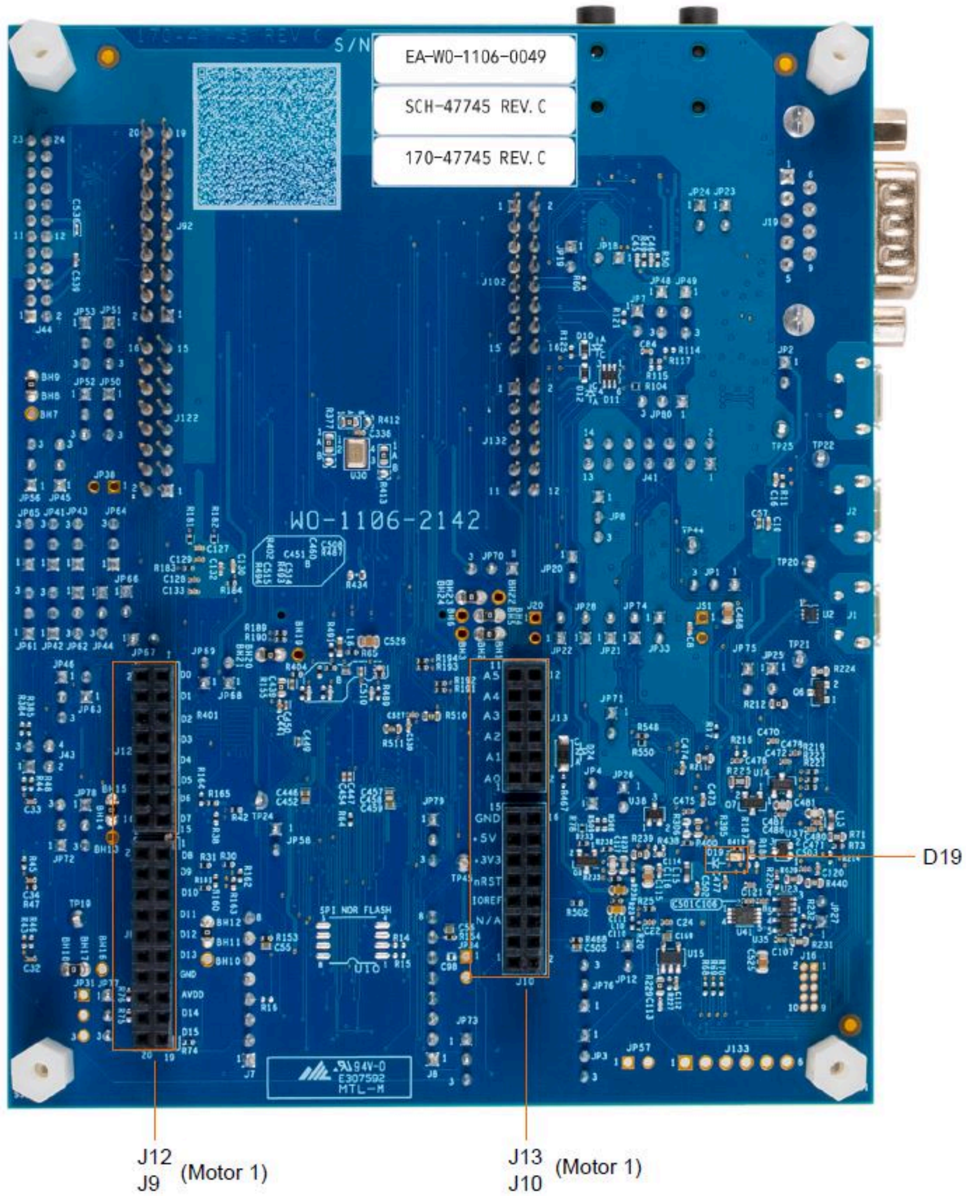


Figure 9. LPC55S36-EVK board (bottom-side view)

2.4.1 Hardware assembling

1. Connect the FRDM-MC-LVPMSM shield to the Motor 1 or Motor 2 connector of the LPC55S36-EVK board.
2. Connect the 3-phase motor wires to the screw terminals (J7) on the Freedom PMSM power stage.
3. Plug the USB cable from the USB host to the MCULink micro USB connector (J1) on the EVK board.
4. Plug the 24-V DC power supply to the DC power connector on the Freedom PMSM power stage.

2.4.2 Additional information

- In the `pmsm_enc` example, user can choose by definition `M1_CONNECTOR_ID`, which motor connector will be used. Set this definition to `M1_CONNECTOR_ID_MC1` for using motor connector 1 or to `M1_CONNECTOR_ID_MC2` for using motor connector 2.

- `pmsm_enc_iopamp` example is available only on motor connector 1.
- In the `pmsm_enc_dual` example, MCAT interface is not available. Motor identification is implemented only on motor connector 1.
- `pmsm_enc_dual` example use internal operational amplifier (IOPAMP) on motor connector 1 as default. External operational amplifier on motor connector 1 can be used as well. In this case, set define `USE_INTERNAL_OPAMPS` to false.
- Definitions mentioned above can be found in `mc_periph_init.h` file.

3 Processors features and peripheral settings

This chapter describes the peripheral settings and application timing.

3.1 LPC-55S36

The LPC55S36 MCU family is built upon Cortex-M33-based MCU introduced with the LPC5500 series. This high-efficiency family leverages the new Armv8-M architecture to introduce new levels of performance and advanced security capabilities, including TrustZone-M and co-processor extensions. The LPC55S36 family enables these co-processors extensions and leverages them to bring significant signal processing efficiency gains from a proprietary DSP accelerator offering a 10x clock cycle reduction. An optional second Cortex-M33 core offers flexibility to balance high performance and power efficiency.

In addition, the LPC55S36 MCU family provides benefits, such as the 40-nm NVM-based process technology cost advantages, broad scalable packages, and memory options, as well as a robust enablement including the MCUXpresso Software and Tools ecosystem and low-cost development boards.

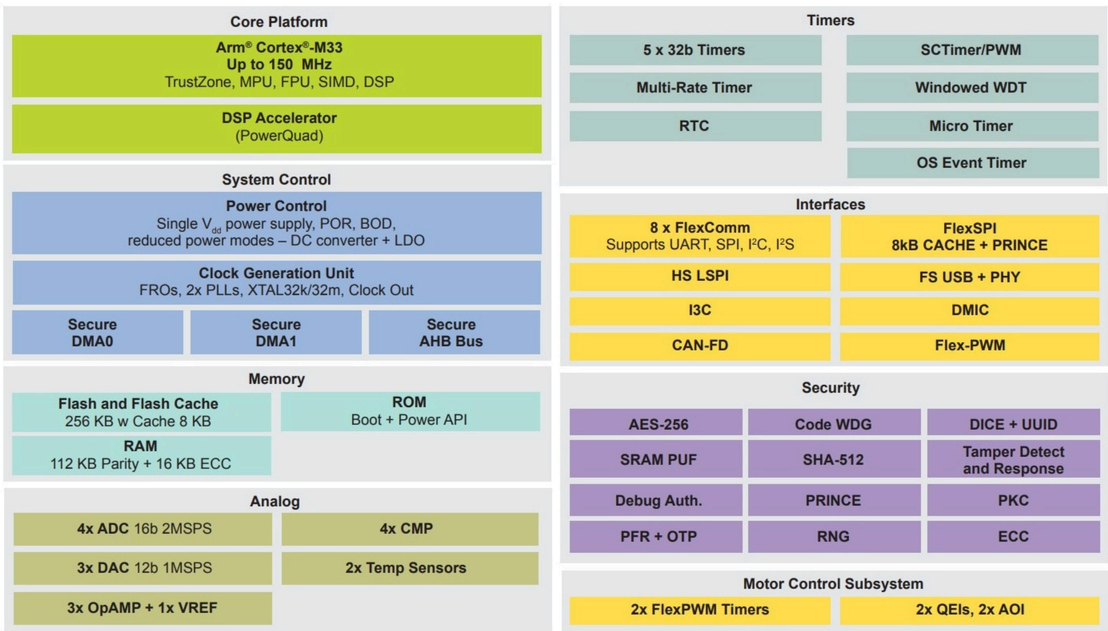


Figure 10. LPC55S3x block diagram

3.1.1 LPC55S36 hardware timing and synchronization

Correct and precise timing is crucial for motor-control applications. Therefore, the motor-control-dedicated peripherals take care of the timing and synchronization on the hardware layer. In addition, you can set the PWM frequency as a multiple of the ADC interrupt (ADC ISR) frequency where the FOC algorithm is calculated. In this case, the PWM frequency is equal to the FOC frequency.

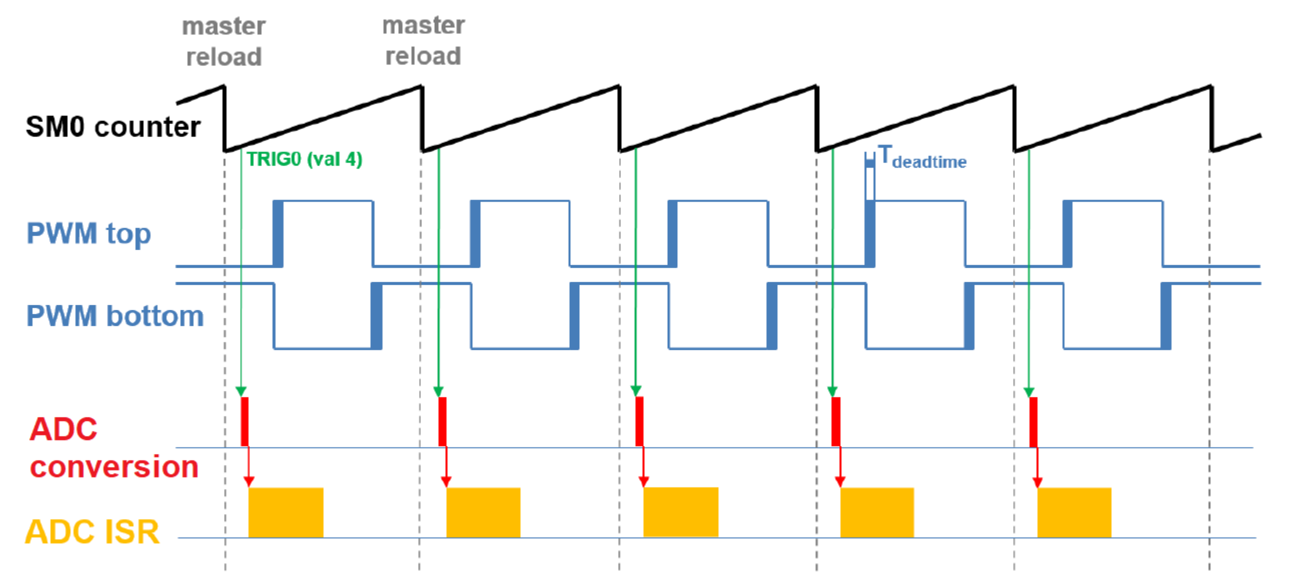


Figure 11. Hardware timing and synchronization on LPC55S36

- The top signal shows the eFlexPWM counter (SM0 counter). The dead time is emphasized at the PWM top and PWM bottom signals. The SM0 submodule generates the master reload at every opportunity.
- The SM0 generates trigger 0 (when the counter counts to a value equal to the VAL4 value) for the ADC with a delay of approximately $T_{deatime}/2$. This delay ensures correct current sampling at the duty cycles close to 100 %.
- ADC starts the ADC conversion.
- When the ADC conversion is completed, the ADC ISR (ADC interrupt) is entered. The FOC calculation is done in this interrupt.

3.2 CPU load and memory usage

The following information applies to the application built using one of the following IDE: MCUXpresso IDE, IAR, or Keil MDK. The memory usage is calculated from the *.map linker file, including FreeMASTER recorder buffer allocated in RAM. In the MCUXpresso IDE, the memory usage can be also seen after project build in the Console window. Table 4 shows the maximum CPU load of the supported examples. The CPU load is measured using the SYSTICK timer. The CPU load is dependent on the fast-loop (FOC calculation) and slow-loop (speed loop) frequencies. The total CPU load is calculated using the following equations:

$$CPU_{fast} = cycles_{fast} \frac{f_{fast}}{f_{CPU}} 100 \left[\%\right] \tag{1}$$

$$CPU_{slow} = cycles_{slow} \frac{f_{slow}}{f_{CPU}} 100 \left[\%\right] \tag{2}$$

$$CPU_{total} = cycles_{fast} + CPU_{slow} \left[\%\right] \tag{3}$$

Where:

CPU_{fast} = the CPU load taken by the fast loop

$cycles_{fast}$ = the number of cycles consumed by the fast loop

f_{fast} = the frequency of the fast-loop calculation

f_{CPU} = CPU frequency

CPU_{slow} = the CPU load taken by the slow loop

$cycles_{slow}$ = the number of cycles consumed by the slow loop

f_{slow} = the frequency of the slow-loop calculation

CPU_{total} = the total CPU load consumed by the motor control

Table 4. Maximum CPU load (fast loop)

	CPU load (debug configuration)		Memory usage (debug configuration)	
Application	Fast loop	Slow loop	Program flash	SRAM
pmsm_enc	39.6%	4.3%	91 892 B	16 192 B
pmsm_enc_iopamp	37.2%	4.5%	91 760 B	16 192 B
pmsm_enc_dual	77.8%	5.0%	109 348 B	17 264 B
pmsm_periph_ framework	N/A	N/A	36 100 B	11 428 B

Note: *Memory usage and maximum CPU load can differ depending on the used IDEs and settings.*

4 Project file and IDE workspace structure

All the necessary files are included in one package, which simplifies the distribution and decreases the size of the final package. The directory structure of this package is simple, easy to use, and organized logically. The folder structure used in the IDE differs from the structure of the PMSM package installation, but it uses the same files. The different organization is chosen due to better manipulation of folders and files in workplaces and the possibility of adding or removing files and directories. The pack_motor_<board_name> project includes all the available functions and routines. This project serves for development and testing purposes.

4.1 PMSM project structure

The directory tree of the PMSM project is shown in below.

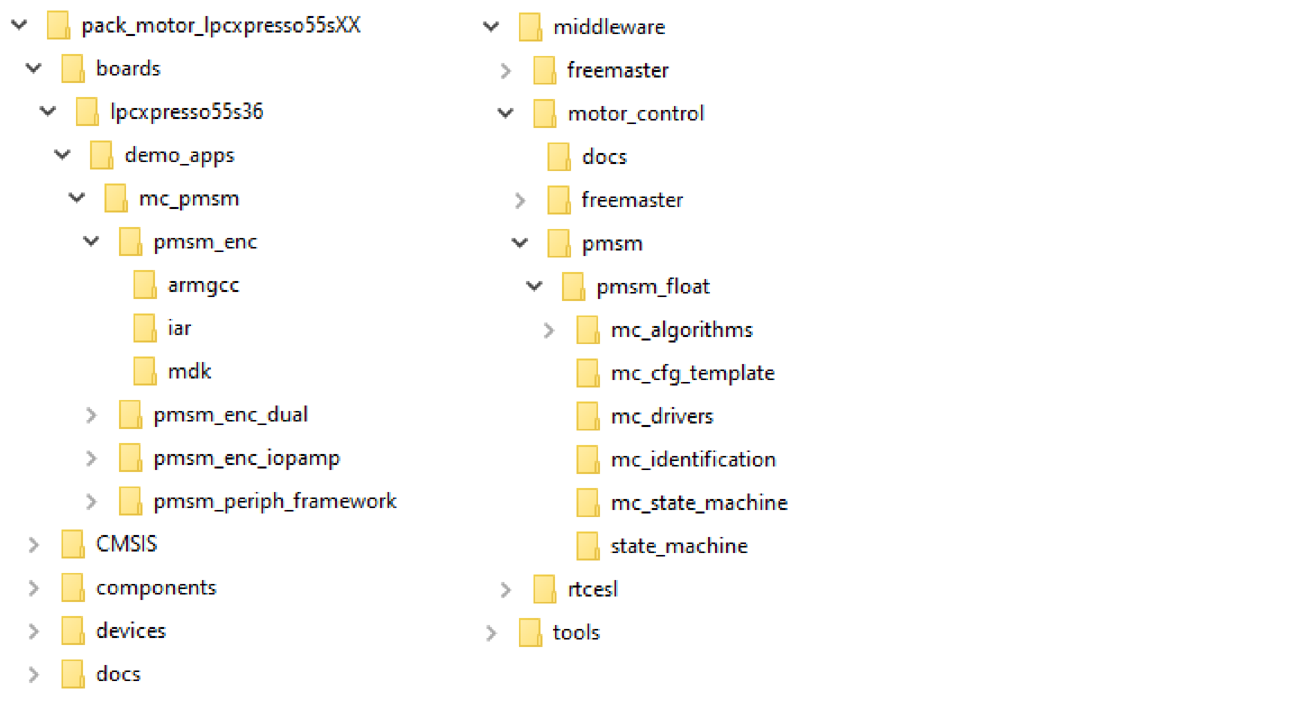


Figure 12. Directory tree

The main project folder pack_motor_lpcxx\boards\lpcxpressoxx\demo_apps\mc_pmsm\pmsm_enc contains the following folders and files:

- iar—for the IAR Embedded Workbench IDE.
- armgcc—for the GNU Arm IDE.
- mdk—for the uVision Keil IDE.
- m1_pmsm_appconfig.h—contains the definitions of constants for the application control processes, parameters of the motor and regulators, and the constants for other vector control-related algorithms. When you tailor the application for a different motor using the Motor Control Application Tuning (MCAT) tool, the tool generates this file at the end of the tuning process.
- main.c—contains the basic application initialization (enabling interrupts), subroutines for accessing the MCU peripherals, and interrupt service routines. The FreeMASTER communication is performed in the background infinite loop.
- board.c—contains the functions for the UART, GPIO, and SysTick initialization.

- `board.h`—contains the definitions of the board LEDs, buttons, UART instance used for FreeMASTER, and so on.
- `clock_config.c and .h`—contains the CPU clock setup functions. These files are going to be generated by the clock tool in the future.
- `mc_periph_init.c`—contains the motor-control driver peripherals initialization functions that are specific for the board and MCU used.
- `mc_periph_init.h`—header file for `mc_periph_init.c`. This file contains the macros for changing the PWM period and the ADC channels assigned to the phase currents and board voltage.
- `freemaster_cfg.h`—the FreeMASTER configuration file containing the FreeMASTER communication and features setup.
- `pin_mux.c and .h`—port configuration files. It is recommended to generate these files in the pin tool.
- `peripherals.c and .h`—MCUXpresso Config Tool configuration files.

The main motor-control folder `pack_motor_lpcxx\middleware\motor_control\` contains these subfolders:

- `pmsm`—contains main pmsm motor-control functions
- `freemaster`—contains the FreeMASTER project files `pmsm_float_enc.pmpx, pmsm_float_enc_dual.pmpx, pmsm_periph_framework.pmpx`. Open these files in the FreeMASTER tool and use it to control the application. The folder also contains the auxiliary files for the MCAT tool.

The `pack_motor_lpcxx\middleware\motor_control\pmsm\pmsm_float` folder contains the following subfolders common to the other motor-control projects:

- `mc_algorithms`—contains the main control algorithms used to control the FOC and speed control loop. Folder also contains MCAA library.
- `mc_cfg_template`—contains templates for MCUXpresso Config Tool components.
- `mc_drivers`—contains the source and header files used to initialize and run motor-control applications.
- `mc_identification`—contains the source code for the automated parameter-identification routines of the motor.
- `mc_state_machine`—contains the software routines that are executed when the application is in a particular state or state transition.
- `state_machine`—contains the state machine functions for the FAULT, INITIALIZATION, STOP, and RUN states.

5 Motor-control peripheral initialization

The motor-control peripherals are initialized by calling the `MCDRV_Init_M1()` function during MCU startup and before the peripherals are used. All initialization functions are in the `mc_periph_init.c` source file and the `mc_periph_init.h` header file. The definitions specified by the user are also in these files. The features provided by the functions are the 3-phase PWM generation and 3-phase current measurement, as well as the DC-bus voltage and auxiliary quantity measurement. The principles of both the 3-phase current measurement and the PWM generation using the Space Vector Modulation (SVM) technique are described in *Sensorless PMSM Field-Oriented Control* (document DRM148).

The `mc_periph_init.h` header file provides the following macros defined by the user:

- `M1_MCDRV_ADC_PERIPH_INIT`: this macro calls ADC peripheral initialization.
- `M1_MCDRV_PWM_PERIPH_INIT`: this macro calls PWM peripheral initialization.
- `M1_MCDRV_QD_ENC`: this macro calls QD peripheral initialization.
- `M1_PWM_FREQ`: the value of this definition sets the PWM frequency.
- `M1_FOC_FREQ_VS_PWM_FREQ`: enables you to call the fast-loop interrupt at every first, second, third, or n^{th} PWM reload. This is convenient when the PWM frequency must be higher than the maximal fast-loop interrupt.
- `M1_SPEED_LOOP_FREQ`: the value of this definition sets the speed loop frequency (TMR1 interrupt).
- `M1_PWM_DEADTIME`: the value of the PWM dead time in nanoseconds.
- `M1_PWM_PAIR_PH[A..C]`: these macros enable a simple assignment of the physical motor phases to the PWM periphery channels (or submodules). You can change the order of the motor phases this way.
- `M1_ADC[1,2]_PH_[A..C]`: these macros assign the ADC channels for the phase current measurement. The general rule is that at least one-phase current must be measurable on both ADC converters, and the two remaining phase currents must be measurable on different ADC converters. The reason for this is that the selection of the phase current pair to measure depends on the current SVM sector. If this rule is broken, a preprocessor error is issued. For more information about the 3-phase current measurement, see *Sensorless PMSM Field-Oriented Control* (document DRM148).
- `M1_ADC[1,2]_UDCB`: this define is used to select the ADC channel for the measurement of the DC-bus voltage.

In the motor-control software, the following API-serving ADC and PWM peripherals are available:

- The available APIs for the ADC are:
  - `mcdrv_adc_t`: MCDRV ADC structure data type.
  - `void M1_MCDRV_ADC_PERIPH_INIT()`: this function is by default called during the ADC peripheral initialization procedure invoked by the `MCDRV_Init_M1()` function and should not be called again after the peripheral initialization is done.
  - `void M1_MCDRV_CURR_3PH_CHAN_ASSIGN(mcdrv_adc_t*)`: calling this function assigns proper ADC channels for the next 3-phase current measurement based on the SVM sector.
  - `void M1_MCDRV_CURR_3PH_CALIB_INIT(mcdrv_adc_t*)`: this function initializes the phase-current channel-offset measurement.
  - `void M1_MCDRV_CURR_3PH_CALIB(mcdrv_adc_t*)`: this function reads the current information from the unpowered phases of a stand-still motor and filters them using moving average filters. The goal is to obtain the value of the measurement offset. The length of the window for moving the average filters is set to eight samples by default.
  - `void M1_MCDRV_CURR_3PH_CALIB_SET(mcdrv_adc_t*)`: this function asserts the phase-current measurement offset values to the internal registers. Call this function after a sufficient number of `M1_MCDRV_CURR_3PH_CALIB()` calls.

  - `void M1_MCDRV_ADC_GET(mcdrv_adc_t*)`: this function reads and calculates the actual values of the 3-phase currents, DC-bus voltage, and auxiliary quantity.
- The available APIs for the PWM are:
  - `mcdrv_pwma_pwm3ph_t`: MCDRV PWM structure data type.
  - `void M1_MCDRV_PWM_PERIPH_INIT`: this function is by default called during the PWM periphery initialization procedure invoked by the `MCDRV_Init_M1()` function.
  - `void M1_MCDRV_PWM3PH_SET(mcdrv_pwma_pwm3ph_t*)`: this function updates the PWM phase duty cycles.
  - `void M1_MCDRV_PWM3PH_EN(mcdrv_pwma_pwm3ph_t*)`: this function enables all PWM channels.
  - `void M1_MCDRV_PWM3PH_DIS(mcdrv_pwma_pwm3ph_t*)`: this function disables all PWM channels.
  - `bool_t M1_MCDRV_PWM3PH_FLT_GET(mcdrv_pwma_pwm3ph_t*)`: this function returns the state of the overcurrent fault flags and automatically clears the flags (if set). This function returns true when an overcurrent event occurs. Otherwise, it returns false.
- The available APIs for the quadrature encoder are:
  - `mcdrv_qd_enc_t`: MCDRV QD structure data type.
  - `void M1_MCDRV_QD_PERIPH_INIT()`: this function is by default called during the QD periphery initialization procedure invoked by the `MCDRV_Init_M1()` function.
  - `void M1_MCDRV_QD_GET(mcdrv_qd_enc_t*)`: this function returns the actual position and speed.
  - `void M1_MCDRV_QD_SET_DIRECTION(mcdrv_qd_enc_t*)`: this function sets the direction of the quadrature encoder.
  - `void M1_MCDRV_QD_CLEAR(mcdrv_qd_enc_t*)`: this function clears the internal variables and decoder counter.

6 User interface

The application contains the demo mode to demonstrate motor rotation. You can operate it either using the user button, or using FreeMASTER. The NXP development boards include a user button associated with a port interrupt (generated whenever one of the buttons is pressed). At the beginning of the ISR, a simple logic executes and the interrupt flag clears. When you press the button, the demo mode starts. When you press the same button again, the application stops and transitions back to the STOP state.

The other way to interact with the demo mode is to use the FreeMASTER tool. The FreeMASTER application consists of two parts: the PC application used for variable visualization and the set of software drivers running in the embedded application. The serial interface transfers data between the PC and the embedded application. This interface is provided by the debugger included in the boards.

The application can be controlled using the following two interfaces:

- The user button on the development board (controlling the demo mode):
  - LCPXpresso55S36-EVK - SW3
- Remote control using FreeMASTER (Following chapter):
  - Setting a variable in the FreeMASTER Variable Watch. See chapture Section 7.4

Identify all motor parameters if you are using your own motor (different from the default motors). The automated parameter identification is described in the following sections.

7 Remote control using FreeMASTER

This section provides information about the tools and recommended procedures to control the sensor/sensorless PMSM Field-Oriented Control (FOC) application using FreeMASTER. The application contains the embedded-side driver of the FreeMASTER real-time debug monitor and data visualization tool for communication with the PC. It supports non-intrusive monitoring, as well as the modification of target variables in real time, which is very useful for the algorithm tuning. Besides the target-side driver, the FreeMASTER tool requires the installation of the PC application as well. You can download FreeMASTER 3.0 at www.nxp.com/freemaster. To run the FreeMASTER application including the MCAT tool, double-click the `pmsm_float_enc.pmpx` file located in the `middleware\motor_control\freemaster` folder. The FreeMASTER application starts and the environment is created automatically, as defined in the *.pmpx file.

Note: *In MCUXpresso, the FreeMASTER application can run directly from IDE in `motor_control/freemaster` folder.*

7.1 Establishing FreeMASTER communication

The remote operation is provided by FreeMASTER via the USB interface. To control a PMSM motor using FreeMASTER, perform the steps below:

1. Download the project from your chosen IDE to the MCU and run it.
2. Open the FreeMASTER project `pmsm_float_enc.pmpx` . The PMSM project uses the TSA by default, so it is not necessary to select a symbol file for FreeMASTER.
3. To establish the communication, click the communication button (the green "GO" button in the top left-hand corner).

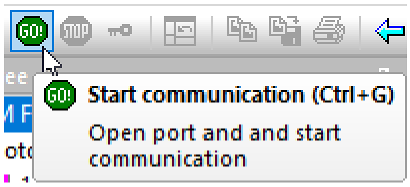


Figure 13. Green "GO" button placed in top left-hand corner

4. If the communication is established successfully, the FreeMASTER communication status in the bottom right-hand corner changes from "Not connected" to "RS-232 UART Communication; COMxx; speed=115200". Otherwise, the FreeMASTER warning pop-up window appears.

Figure 14. FreeMASTER—communication is established successfully

5. To reload the MCAT HTML page and check the App ID, press F5.
6. Control the PMSM motor by writing to a control variable in a variable watch.
7. If you rebuild and download the new code to the target, turn the FreeMASTER application off and on.

If the communication is not established successfully, perform the following steps:

1. Go to the **Project** > **Options** > **Comm** tab and make sure that the correct COM port is selected and the communication speed is set to 115200 bps.

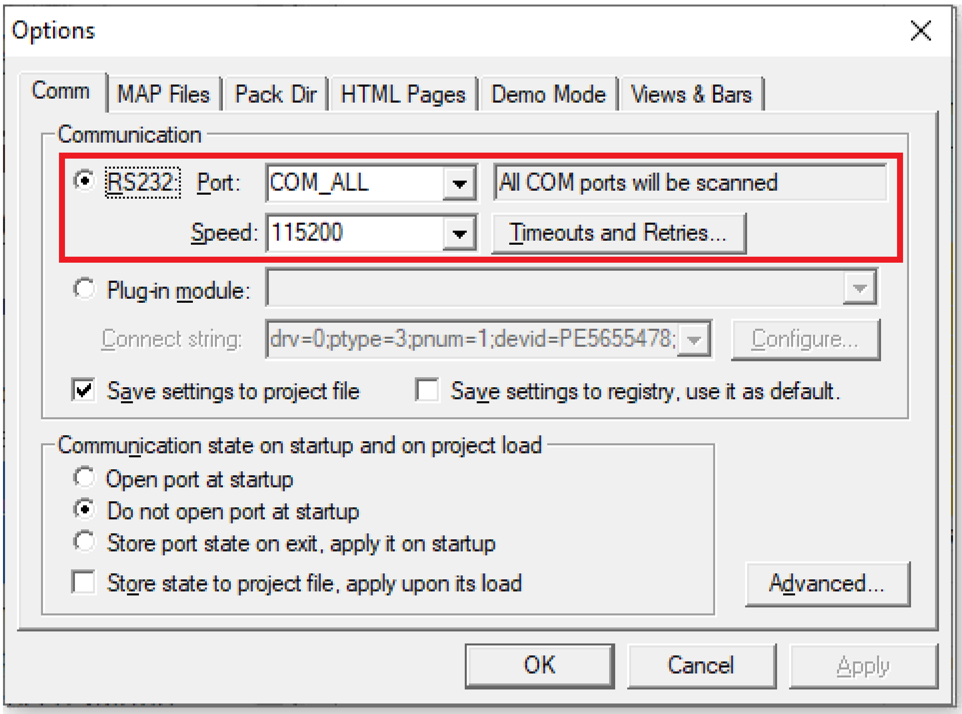


Figure 15. FreeMASTER communication setup window

2. Ensure, that your computer is communicating with the plugged board. Unplug and then plug in the USB cable and reopen the FreeMASTER project.

7.2 TSA replacement with ELF file

The FreeMASTER project for motor control example uses Target-Side Addressing (TSA) information about variable objects and types to be retrieved from the target application by default. With the TSA feature, you can describe the data types and variables directly in the application source code and make this information available to the FreeMASTER tool. The tool can then use this information instead of reading symbol data from the application's ELF/Dwarf executable file.

FreeMASTER reads the TSA tables and uses the information automatically when an MCU board is connected. A great benefit of using the TSA is no issues with the correct path to ELF/Dwarf file. The variables described by TSA tables may be read-only, so even if FreeMASTER attempts to write the variable, the target MCU side denies the value. The variables not described by any TSA tables may also become invisible and protected even for read-only access.

The use of TSA means more memory requirements for the target. If you do not want to use the TSA feature, you must modify the example code and FreeMASTER project.

To modify the example code, follow the steps below:

1. Open motor control project and rewrite macro `FMSTR_USE_TSA` from 1 to 0 in `freemaster_cfg.h` file.
2. Build, download, and run motor control project.
3. Open FreeMASTER project and click to **Project** > **Options** (or use shortcut Ctrl+T).
4. Click to **MAP Files** tab and find Default symbol file (ELF/Dwarf executable file) located in IDE output folder.

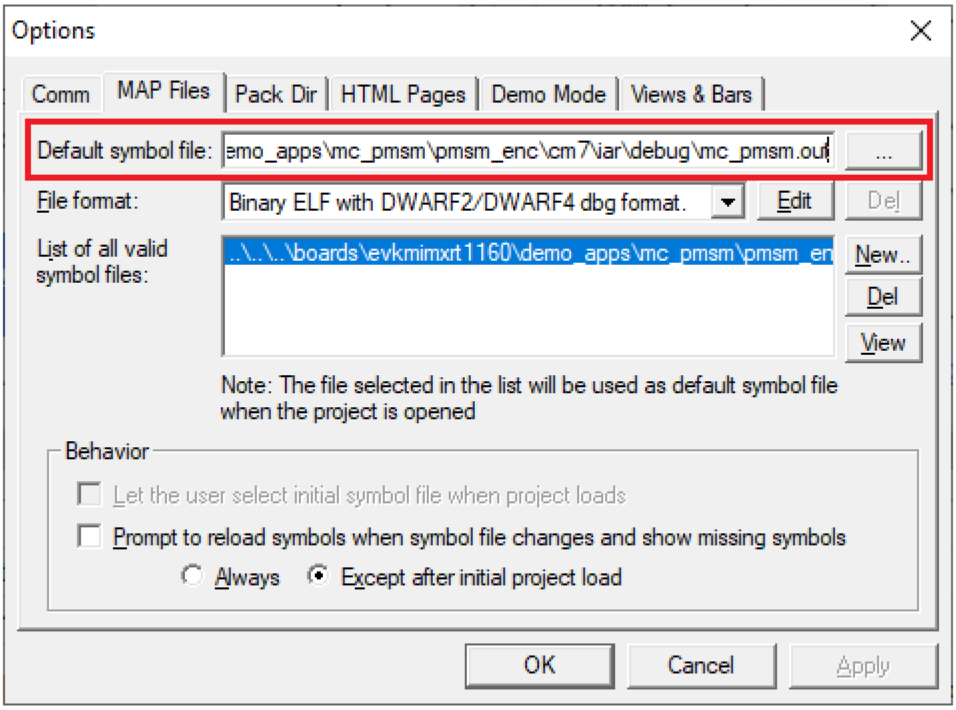


Figure 16. Default symbol file

5. Click **OK** and restart the FreeMASTER communication.

For more information, check FreeMASTER User Guide.

7.3 Motor Control Aplication Tuning interface (MCAT)

The PMSM sensor/sensorless FOC application can be easily controlled and tuned using the Motor Control Application Tuning (MCAT) plug-in for PMSM. The MCAT for PMSM is a user-friendly page, which runs within the FreeMASTER. The tool consists of the tab menu and workspace as shown in Figure 17. Each tab from the tab menu (4) represents one submodule which enables tuning or controlling different application aspects. Besides the MCAT page for PMSM, several scopes, recorders, and variables in the project tree (5) are predefined in the FreeMASTER project file to further the motor parameter tuning and debugging simplify.

When the FreeMASTER is not connected to the target, the "Board found" line (2) shows "Board ID not found". When the communication with the target MCU is established, the "Board found" line is read from Board ID variable watch and displayed. If the connection is established and the board ID is not shown, press F5 to reload the MCAT HTML page.

There are three action buttons in MCAT (3):

- **Load data** - MCAT input fields (for example, motor parameters) are loaded from `mX_pmsm_appconfig.h` file (JSON formatted comments). Only `existing mX_pmsm_appconfig.h` files can be selected for loading. Loaded `mX_pmsm_appcofig.h` file is displayed in grey field (7).
- **Save data** - MCAT input fields (JSON formatted comments) and output macros are saved to `mX_pmsm_appconfig.h` file. Up to 9 files (`m1-9_pmsm_appconfig.h`) can be selected. A pop-up window with the user motor ID and description appears when a different `mX_pmsm_appcofig.h` file is selected. The motor ID and description are also saved in `mX_pmsm_appcofig.h` as a JSON comment. The embedded code includes `m1_pmsm_appcofig.h` only at single motor control application. Therefore, saving to higher indexed `mX_pmsm_appconfig.h` files has no effect at the compilation stage.
- **Update target** - writes the MCAT calculated tuning parameters to FreeMASTER Variables, which effectively updates the values on target MCU. These tuning parameters are updated in MCU's RAM. To write these

tuning parameters to MCU's flash memory, `m1_pmsm_appcofig.h` must be saved, code recompiled, and downloaded to MCU.

Note: *Path to `mX_pmsm_appcofig.h` file also composed from Board ID value. Therefore, FreeMASTER must be connected to the target, and Board ID value read prior using Save/Load buttons.*

Note: *Only **Update target** button updates values on the target in real time. Load/Save buttons operate with `mX_pmsm_appcofig.h` file only.*

Note: *MCAT may require Internet connection. If no Internet connection is available, CSS and icons may not be properly loaded.*

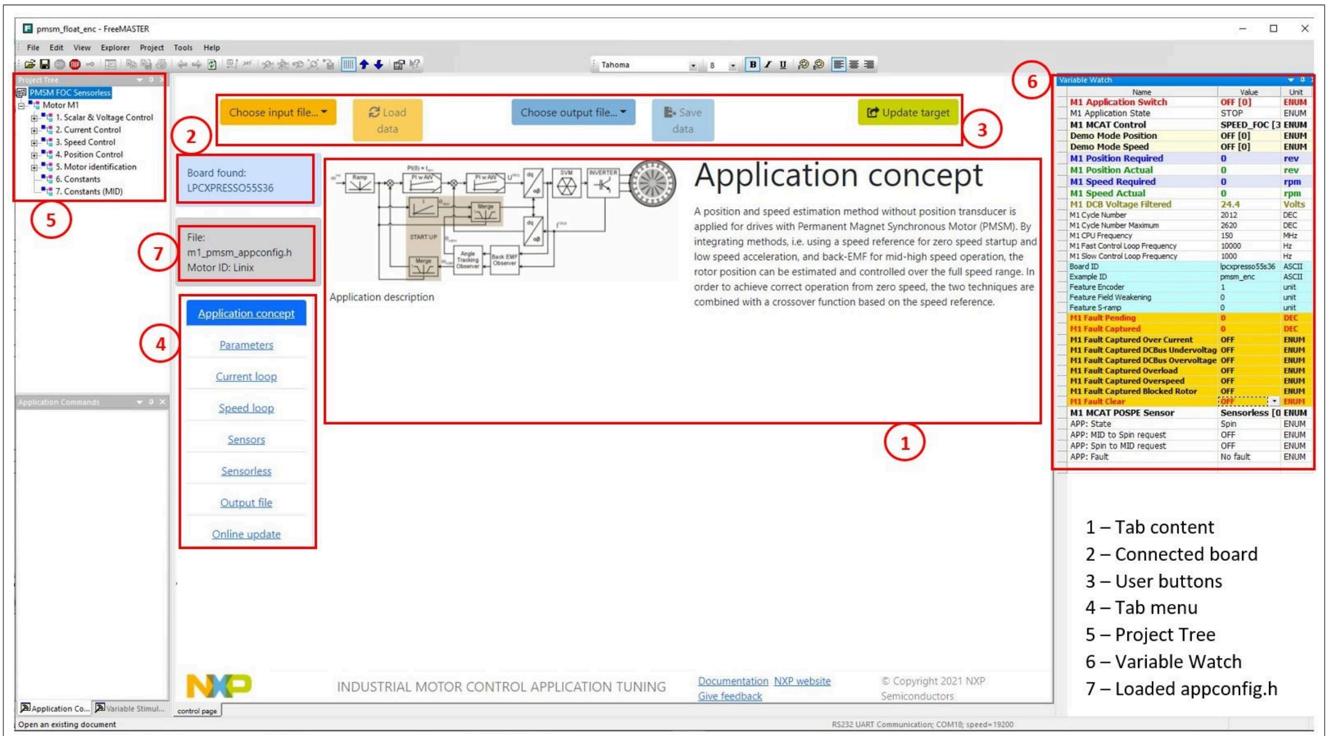


Figure 17. FreeMASTER + MCAT layout

1. Tab content
2. Connected board
3. User buttons
4. Tab menu
5. Project tree
6. Variable watch
7. Loaded configuration

In the default configuration, the following tabs (4) are available:

- Application concept: welcome page with the PMSM sensor/sensorless FOC diagram and a short application description.
- Parameters: this page enables you to modify the motor parameters, hardware and application scales specification, alignment, and fault limits.
- Current loop: current loop PI controller gains and output limits.
- Speed loop: this tab contains fields for the specification of the speed controller proportional and integral gains, as well as the output limits and parameters of the speed ramp. The position proportional controller constant is also set here.

- Sensors: this page contains the encoder parameters and position observer parameters.
- Sensorless: this page enables you to tune the parameters of the BEMF observer, tracking observer, and open-loop startup.
- Output file: this tab shows all the calculated constants that are required by the PMSM sensor/sensorless FOC application. It is also possible to generate the `m1_pmsm_appconfig.h` file, which is then used to preset all application parameters permanently at the project rebuild.
- Online update : this tab shows actual values of variables on target and new calculated values, which can be used to update the target variables.

Every sublock in FreeMASTER project tree (5) has defined several variables in variable watch (6).

The following sections provide simple instructions on how to identify the parameters of a connected PMSM motor and how to tune the application appropriately.

7.3.1 MCAT tabs description

This chapter describes MCAT input parameters and equations used to calculate MCAT output (generated) parameters. In the default configuration, the below described tabs are available. Some tabs may be missing if not supported in the embedded code. There are general constants used at MCAT calculations listed in the following table:

Table 5. Constants used in equations

Constant	Value	Unit
UmaxCoeff	1.73205	-
DiscMethodFactor	1	-
k_factor	100	-
pi	3.1416	-

7.3.1.1 Application concept

This tab is a welcome page with the PMSM sensor/sensorless FOC diagram and a short description of the application.

7.3.1.2 Parameters

This tab enables modification of motor parameters, specification of hardware and application scales, alignment, and fault limits. All inputs are described in the following table. MCAT group and MCAT name help to locate the parameter in MCAT layout. Equation name represents the input parameter in equations below.

Table 6. Parameters tab inputs

MCAT group	MCAT name	Equation name	Description	Unit
Motor parameters	PP	Pp	Motor number of pole-pairs. Obtain from motor manufacturer or use the pole-pair assistant to determine and then fill manually.	-
	Rs	Rs	Stator phase resistance. Obtain from motor manufacturer or use the electrical parameters identification and then fill manually.	[Ω]
	Ld	Ld	Stator direct inductance. Obtain from motor manufacturer or	[H]

Table 6. Parameters tab inputs...*continued*

MCAT group	MCAT name	Equation name	Description	Unit
			use the electrical parameters identification and then fill manually.	
	Lq	Lq	Stator quadrature inductance. Obtain from motor manufacturer or use the electrical parameters identification and then fill manually.	[H]
	Ke	Ke	Motor electrical constant. Obtain from motor manufacturer or use the Ke identification and then fill manually.	[V.sec/rad]
	J	J	Drive inertia (motor + plant). Use the mechanical identification and then fill manually.	[kg.m2]
	Iph nom	IphNom	Nominal motor current. Obtain from motor manufacturer.	[A]
	Uph nom	UphNom	Nominal motor voltage. Obtain from motor manufacturer.	[V]
	N nom	Nnom	Nominal motor speed. Obtain from motor manufacturer.	[rpm]
Hardware scales	I max	Imax	Current sensing HW scale. Keep as-is in case of standard NXP HW or recalculate according to own schematic.	[A]
	U DCB max	UdcbMax	DCBus voltage sensing HW scale. Keep as-is in case of standard NXP HW or recalculate according to own schematic.	[V]
Fault limits	U DCB trip	UdcbTrip	DCBus braking resistor threshold. Braking resistor's transitor is turned on when DCbus voltage exceeds this threshold.	[V]
	U DCB under	UdcbUnder	DCBus under voltage fault threshold	[V]
	U DCB over	UdcbOver	DCBus over voltage fault threshold	[V]
	N over	Nover	Over speed fault threshold	[rpm]
	N min	Nmin	Minimal closed loop speed. When the required speed ramps down under this threshold, the motor control state machine goes to freewheel state where top and bottom transistors are turned off and motor speeds down freely. Applies only for sensorless operation.	[rpm]

Table 6. Parameters tab inputs*...continued*

MCAT group	MCAT name	Equation name	Description	Unit
	E block	Eblock	Blocked rotor detection. When BEMF voltage drops under *E block* threshold for more than *E block per* (fast loop ticks), the blocked rotor fault is detected.	[V]
	E block per	EblockPer		-
Application scales	N max	Nmax	Application speed scale. Keep about 10 % margin above *N over*.	[rpm]
	U DCB IIR F0	UdcbIIRf0	Cut-off frequency of DCBus IIR filter	[Hz]
	Calibration duration	CalibDuration	ADC (phase current offset) calibration duration. Done every time transitioning from STOP to RUN.	[sec]
	Fault duration	FaultDuration	After fault condition disappears, wait defined time to clear pending faults bitfield and transition to STOP state.	[sec]
	Freewheel duration	FreewheelDuration	Free-wheel state duration. Freewheel state in entered when ramped speed drops under *N min*.	[sec]
	Scalar Uq min	ScalarUqMin	Scalar control voltage minimal value.	[V]
Alignment	Align voltage	AlignVoltage	Motor alignment voltage.	[V]
	Align duration	AlignDuration	Motor alignment duration.	[sec]

Output equations (applies for saving to `mX_pmsm_appcofig.h` and also for updating a corresponding FreeMASTER variable):

- `M1_U_MAX` = UdcbMax / UmaxCoeff
- `M1_FREQ_MAX` = Nmax / 60 * Pp
- `M1_ALIGN_DURATION` = AlignDuration / speedLoopSampleTime
- `M1_CALIB_DURATION` = CalibDuration / speedLoopSampleTime
- `M1_FAULT_DURATION` = FaultDuration / speedLoopSampleTime
- `M1_FREEWHEEL_DURATION` = FreewheelDuration / speedLoopSampleTime
- `M1_E_BLOCK_PER` = EblockPer
- `M1_SPEED_ANGULAR_SCALE` = 60 / (Pp * 2 * pi)
- `M1_N_MIN` = Nmin / 60 * (Pp * 2 * pi)
- `M1_N_MAX` = Nmax / 60 * (Pp * 2 * pi)
- `M1_N_ANGULAR_MAX` = (60 / (Pp * 2 * pi))
- `M1_N_NOM` = Nnom / 60 * (Pp * 2 * pi)
- `M1_N_OVERSPEED` = Nover / 60 * (Pp * 2 * pi)
- `M1_UDCB_IIR_B0` = (2 * pi * UdcbIIRf0 * currentLoopSampleTime) / (2 + (2 * pi * UdcbIIRf0 * currentLoopSampleTime))
- `M1_UDCB_IIR_B1` = (2 * pi * UdcbIIRf0 * currentLoopSampleTime) / (2 + (2 * pi * UdcbIIRf0 * currentLoopSampleTime))

- `M1_UDCB_IIR_A1` = -(2 * pi * UdcbIIRf0 * currentLoopSampleTime - 2) / (2 + (2 * pi * UdcbIIRf0 * currentLoopSampleTime))
- `M1_SCALAR_VHZ_FACTOR_GAIN` = UphNom*k_factor/100/(Nnom*Pp/60)
- `M1_SCALAR_INTEG_GAIN` = 2*pi*Pp*Nmax/60*currentLoopSampleTime/pi
- `M1_SCALAR_RAMP_UP` = speedLoopIncUp*currentLoopSampleTime/60*Pp
- `M1_SCALAR_RAMP_DOWN` = speedLoopIncDown*currentLoopSampleTime/60*Pp

7.3.1.3 Current loop

This tab enables current loop PI controller gains and output limits tuning. All inputs are described in the following table. MCAT group and MCAT name help to locate the parameter in MCAT layout. Equation name represents the input parameter in equations bellow.

Table 7. Current loop tab input

MCAT group	MCAT name	Equation name	Description	Unit
Loop parameters	Sample time	currentLoopSampleTime	Fast control loop period. This disabled value is read from target via FreeMASTER because application timing is set in embedded code by peripherals setting. This value is accessible only if target is not connected and value cannot be obtained from target.	[sec]
	F0	currentLoopF0	Current controller's bandwidth	[Hz]
	ξ	currentLoopKsi	Current controller's attenuation	-
Current PI controller limits	Output limit	currentLoopOutputLimit	Current controllers' output voltage limit = Duty cycle limit. Be careful setting this limit above 95 % because it affects current sensing (Some minimal bottom transistors on time is required).	[%]

Output equations (applies for saving to `mX_pmsm_appcofig.h` and also for updating a corresponding FreeMASTER variable):

- `M1_CLOOP_LIMIT` = currentLoopOutputLimit / UmaxCoeff / 100
- `M1_D_KP_GAIN` = (2 * currentLoopKsi * 2 * pi * currentLoopF0 * Ld) - Rs
- `M1_D_KI_GAIN` = (2 * pi * currentLoopF0)^2 * Ld * currentLoopSampleTime / DiscMethodFactor
- `M1_Q_KP_GAIN` = (2 * currentLoopKsi * 2 * pi * currentLoopF0 * Lq) - Rs
- `M1_Q_KI_GAIN` = (2 * pi * currentLoopF0)^2 * Lq * currentLoopSampleTime / DiscMethodFactor

7.3.1.4 Speed loop

This tab enables speed loop PI controller gains and output limits tuning, required speed ramp parameters, feedback speed filter tuning, and position P controller gain tuning (available at sensored/encoder applications only). MCAT group and MCAT name help to locate the parameter in MCAT layout. Equation name represents the input parameter in equations bellow.

Table 8. Speed loop tab input

MCAT group	MCAT name	Equation name	Description	Unit
Loop parameters	Sample time	speedLoopSampleTime	Slow control loop period. This disabled value is read from target via FreeMASTER because application timing is set in embedded code by peripherals setting. This value is accessible only if target is not connected and value cannot be obtained from target.	[sec]
	F0	speedLoopF0	Speed controller's bandwidth	[Hz]
	ξ	speedLoopKsi	Speed controller's attenuation	-
Speed ramp	Inc up	speedLoopIncUp	Required speed maximal acceleration	[rpm/sec]
	Inc down	speedLoopIncDown	Required speed maximal acceleration	[rpm/sec]
Actual speed filter	Cut-off freq	speedLoopCutOffFreq	Speed feedback (before entering PI subtraction) filter bandwidth.	[Hz]
Speed PI controller limits	Upper limit	speedLoopUpperLimit	Maximal required Q-axis current (Speed controller's output). Q-axis current limitation equals to motor torque limitation.	[A]
	Lower limit	speedLoopLowerLimit	Minimal required Q-axis current (Speed controller's output). Q-axis current limitation equals to motor torque limitation.	[A]
Position P controller constants	PL_Kp	speedLoopPLKp	Position controller proportional constant in time domain.	

Output equations (applies for saving to `mX_pmsm_appcofig.h` and also for updating a corresponding FreeMASTER variable):

- varKt = 3 * Ke / (sqrt(3))
- `M1_SPEED_PI_PROP_GAIN` = (2 * pi / 60 * (4 * speedLoopKsi * pi * speedLoopF0) * J / varKt)
- `M1_SPEED_PI_INTEG_GAIN` = (2 * pi / 60 * ((2 * pi * speedLoopF0) * (2 * pi * speedLoopF0) * J) / (varKt * 10) * speedLoopSampleTime)
- `M1_SPEED_RAMP_UP` = (speedLoopIncUp * speedLoopSampleTime / (60 / (Pp * 2 * pi)))
- `M1_SPEED_RAMP_DOWN` = (speedLoopIncDown * speedLoopSampleTime / (60 / (Pp * 2 * pi)))
- `M1_SPEED_IIR_B0`= (2 * pi * speedLoopCutOffFreq * currentLoopSampleTime) / (2 + (2 * pi * speedLoopCutOffFreq * currentLoopSampleTime))
- `M1_SPEED_IIR_B1` = (2 * pi * speedLoopCutOffFreq * currentLoopSampleTime) / (2 + (2 * pi * speedLoopCutOffFreq * currentLoopSampleTime))
- `M1_SPEED_IIR_A1` = -(2 * pi * speedLoopCutOffFreq * currentLoopSampleTime - 2) / (2 + (2 * pi * speedLoopCutOffFreq * currentLoopSampleTime))

7.3.1.5 Sensors

Available at sensored (encoder) applications only. This tab enables setting the encoder properties and tuning encoder's tracking observer. MCAT group and MCAT name help to locate the parameter in MCAT layout. Equation name represents the input parameter in equations bellow.

Table 9. Sensors tab input

MCAT group	MCAT name	Equation name	Description	Unit
Quadrature encoder	Pulse number	sensorEncPulseNumber	Number of quadrature encoder pulses. Obtain this value from encoder manufacturer OR estimate based on speed/ position comparison of Scalar controlled application with encoder processing running on background.	[pulses]
	Direction	sensorEncDir	Encoder direction / Phase A&B order.	-
	Minimal speed	sensorEncNmin	Encoder minimal speed.	[rpm]
Position observer parameters	Sample time	sensorObsrvParSampleTime	Current control loop sampling period. This disabled value is read from target via Free MASTER because application timing is set in embedded code by peripherals setting. This value is accessible only if target is not connected and value cannot be obtained from target.	[sec]
	F0	sensorObsrvParF0	Position observer bandwidth	[Hz]
	ξ	sensorObsrvParKsi	Position observer attenuation	-

Output equations (applies for saving to `mX_pmsm_appcofig.h` and also for updating a corresponding FreeMASTER variable):

- `M1_POSPE_KP_GAIN` = (4.0 * pi * sensorObsrvParKsi * sensorObsrvParF0)
- `M1_POSPE_KI_GAIN` = ((2*pi*sensorObsrvParF0)^2 * sensorObsrvParSampleTime)
- `M1_POSPE_INTEG_GAIN` = (sensorObsrvParSampleTime / pi / DiscMethodFactor)
- `M1_POSPE_ENC_N_MIN` = sensorEncNmin
- `M1_POSPE_MECH_POS_GAIN` = (32768/((sensorEncPulseNumber*4)/2))

7.3.1.6 Sensorless

This tab enables BEMF observer and Tracking observer parameters tuning and open-loop startup tuning. MCAT group and MCAT name help to locate the parameter in MCAT layout. Equation name represents the input parameter in equations bellow.

Table 10. Sensorless tab input

MCAT group	MCAT name	Equation name	Description	Unit
BEMF observer parameters	F0	sensorlessBemfObsrvF0	BEMF observer bandwidth	[Hz]
	ξ	sensorlessBemfObsrvKsi	BEMF observer attenuation	-

Table 10. Sensorless tab input*...continued*

MCAT group	MCAT name	Equation name	Description	Unit
Tracking observer parameters	F0	sensorlessTrackObsrvF0	Tracking observer bandwidth	[Hz]
	ξ	sensorlessTrackObsrvKsi	Tracking observer attenuation	-
Open loop startup parameters	Startup ramp	sensorlessStartupRamp	Open loop startup ramp	[rpm/sec]
	Startup current	sensorlessStartupCurrent	Open loop startup current	[A]
	Merging Speed	sensorlessMergingSpeed	Merging speed	[rpm]
	Merging Coefficient	sensorlessMergingCoeff	Merging coefficient (100 % = merging is done within one electrical revolution)	[%]

Output equations (applies for saving to `mX_pmsm_appcofig.h` and also for updating a corresponding FreeMASTER variable):

- `M1_I_SCALE` = (Ld / (Ld + currentLoopSampleTime * Rs))
- `M1_U_SCALE` = (currentLoopSampleTime / (Ld + currentLoopSampleTime * Rs))
- `M1_E_SCALE` = (currentLoopSampleTime / (Ld + currentLoopSampleTime * Rs))
- `M1_WI_SCALE` = (Lq * currentLoopSampleTime / (Ld + currentLoopSampleTime * Rs))
- `M1_BEMF_DQ_KP_GAIN` = ((2 * sensorlessBemfObsrvKsi * 2 * pi * sensorlessBemfObsrvF0 * Ld - Rs))
- `M1_BEMF_DQ_KI_GAIN` = (Ld * (2 * pi * sensorlessBemfObsrvF0)^ 2 * currentLoopSampleTime)
- `M1_TO_KP_GAIN` = 2 * sensorlessTrackObsrvKsi * 2 * pi * sensorlessTrackObsrvF0
- `M1_TO_KI_GAIN` = ((2 * pi * sensorlessTrackObsrvF0)^ 2) * currentLoopSampleTime
- `M1_TO_THETA_GAIN` = (currentLoopSampleTime / pi)
- `M1_OL_START_RAMP_INC` = (sensorlessStartupRamp * currentLoopSampleTime / (60 / (Pp * 2 * pi)))
- `M1_MERG_SPEED_TRH` = (sensorlessMergingSpeed / (60 / (Pp * 2 * pi)))
- `M1_MERG_COEFF` = ((sensorlessMergingCoeff / 100) * sensorlessMergingSpeed * Pp * currentLoopSampleTime) / 60
- TO_IIR_cutoff_freq = 1 / (2 * speedLoopSampleTime) * 0.8
- `M1_TO_SPEED_IIR_B0` = (2 * pi * TO_IIR_cutoff_freq * currentLoopSampleTime) / (2 + (2 * pi * TO_IIR_cutoff_freq * currentLoopSampleTime))
- `M1_TO_SPEED_IIR_B1` = (2 * pi * TO_IIR_cutoff_freq * currentLoopSampleTime) / (2 + (2 * pi * TO_IIR_cutoff_freq * currentLoopSampleTime))
- `M1_TO_SPEED_IIR_A1` = -(2 * pi * TO_IIR_cutoff_freq * currentLoopSampleTime - 2) / (2 + (2 * pi * TO_IIR_cutoff_freq * currentLoopSampleTime))

7.4 Motor Control Modes - How to run motor

In the "Project Tree", you can choose between the scalar and FOC control using the appropriate FreeMASTER tabs. The FreeMASTER variables can control the application, corresponding to the control structure selected in the FreeMASTER project tree. This is useful for application tuning and debugging. The required control structure must be selected in the "M1 MCAT Control" variable. To turn on or off the application, use "M1 Application Switch" variable. Set/clear "M1 Application Switch" variable also enables/disables all PWM channels.

Before motor starts, several conditios have to be completed:

1. Connected power supply to the inverter with the right voltage value.
2. If you want to use sensored control (encoder feedback), connect the encoder to the inverter.

3. No pending fault. Check variable "M1 Fault Pending" in "Motor M1" project tree subblock. If there is some value, first remove the cause of the fault, or disable fault checking. (for example in variable "M1 Fault Enable Blocked Rotor")

7.4.1 Scalar control

The scalar control diagram is shown in figure below. It is the simplest type of motor-control techniques. The ratio between the magnitude of the stator voltage and the frequency must be kept at the nominal value. Therefore, the control method is sometimes called Volt per Hertz (or V/Hz). The position estimation BEMF observer and tracking observer algorithms run in the background, even if the estimated position information is not directly used. This is useful for the BEMF observer tuning. For more information, see the *Sensorless PMSM Field-Oriented Control* (document DRM148).

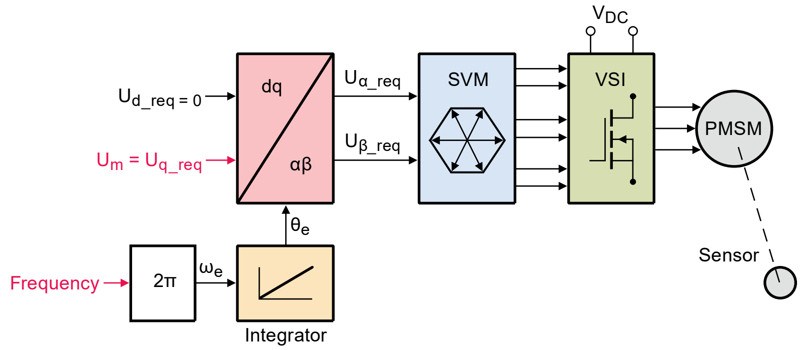


Figure 18. Scalar control mode

For run motor in scalar control, follow these steps:

1. Switch project tree subblock on "Scalar & Voltage Control".
2. Switch variable "M1 MCAT Control" on "SCALAR_CONTROL".
3. In variable "M1 Scalar Freq Required" set required frequency. (i.e. 20Hz)
4. Set variable "M1 Application Switch" to "1". Motor start spinning.
5. Observe motor speed, position, phase currents and other graphs predefined in subblock scopes and recorders.

7.4.2 Open loop control mode

Open loop mode (its diagram is shown in figure below) is similar in function to the Scalar control mode. However, it provides more flexibility in specifying required parameters. This mode allows you to set specific angle and frequency, according to the following equation:

$$\theta_{el} = \theta_{init} + \int_{t_0}^{t} 2\pi f \ dt \tag{4}$$

Besides setting voltage in DQ axis, when using this mode you can also enable current controllers and specify required currents in D and Q axis. Therefore, this function can be utilized for current controller parameter tuning. Please, bear in mind that current controllers cannot be enabled/disabled in SPIN state (user must turn the Application Switch OFF before enabling/disabling current controllers).

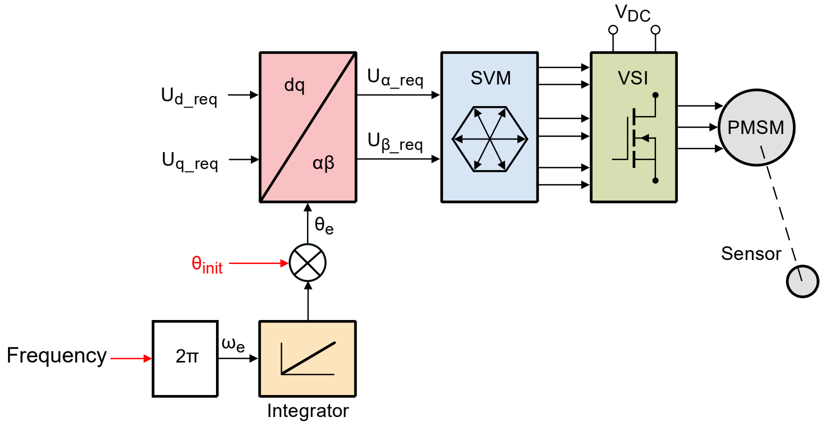


Figure 19. Voltage - Open loop control

For run motor in Voltage - Open loop control, follow these steps:

1. Switch project tree subblock on "Openloop Control".
2. Switch variable "M1 MCAT Control" on "OPEN_LOOP".
3. In variable "M1 Openloop Required Ud" and "M1 Openloop Required Uq" set required values.
4. In variable "M1 Openloop Theta Electrical" set required initial position.
5. In variable "M1 Openloop Required Frequency Electrical" set required frequency.
6. Set variable "M1 Application Switch" to "1". Motor start spinning.
7. Observe motor speed, position, phase currents and other graphs predefined in subblock scopes and recorders.

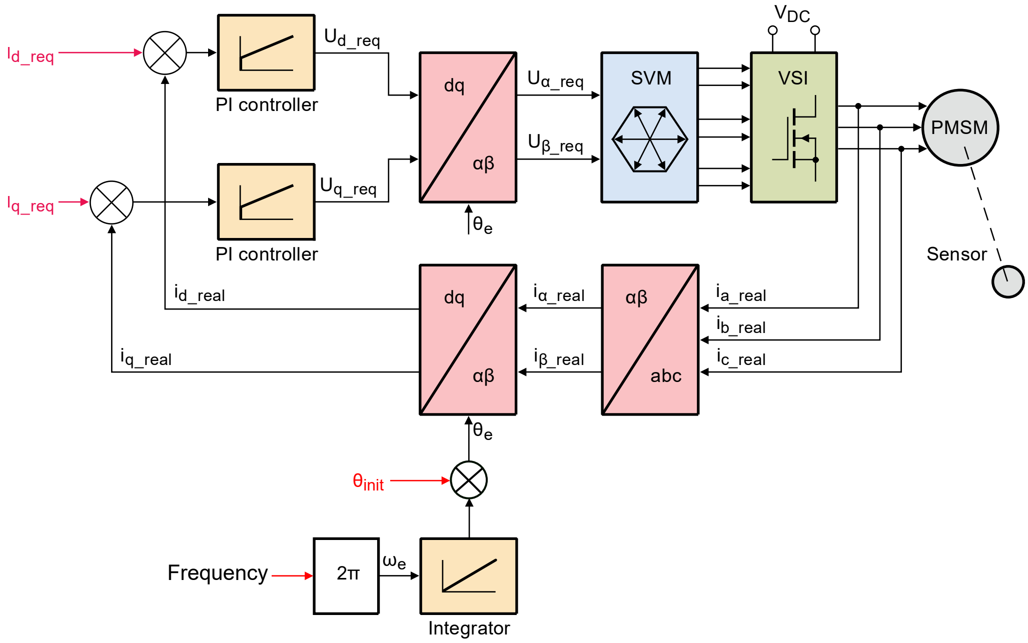


Figure 20. Current - Open loop control

For run motor in Current - Open loop control, follow these steps:

1. Switch project tree subblock on "Openloop Control".
2. Switch variable "M1 MCAT Control" on "OPEN_LOOP".
3. Set variable "M1 Openloop Use I Control" to "1".

4. In variable "M1 Openloop Required Id" and "M1 Openloop Required Iq" set required values.
5. In variable "M1 Openloop Theta Electrical" set required initial position.
6. In variable "M1 Openloop Required Frequency Electrical" set required frequency.
7. Set variable "M1 Application Switch" to "1". Motor start spinning.
8. Observe motor speed, position, phase currents and other graphs predefined in subblock scopes and recorders.

7.4.3 Voltage control

The block diagram of the voltage FOC is shown in Figure 21. Unlike the scalar control, the position feedback is closed using the BEMF observer and the stator voltage magnitude is not dependent on the motor speed. Both the d-axis and q-axis stator voltages can be specified in the "M1 MCAT Ud Required" and "M1 MCAT Uq Required" fields. This control method is useful for the BEMF observer functionality check.

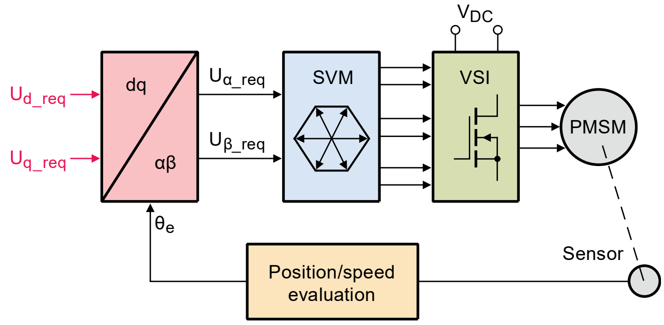


Figure 21. Voltage FOC control mode

For run motor in voltage control, follow these steps:

1. Switch project tree subblock on "Scalar & Voltage Control".
2. Switch variable "M1 MCAT Control" on "VOLTAGE_FOC".
3. In variable "M1 MCAT Uq Required" and "M1 MCAT Ud Required" set required voltages.
4. Set variable "M1 Application Switch" to "1". Motor start spinning.
5. Observe motor speed, position, phase currents and other graphs predefined in subblock scopes and recorders.

7.4.4 Current (torque) control

The current FOC (or torque) control requires the rotor position feedback and the currents transformed into a d-q reference frame. There are two reference variables ("M1 MCAT Id Required" and "M1 MCAT Iq Required") available for the motor control, as shown in Figure 22. The d-axis current component "M1 MCAT Id Required" controls the rotor flux. The q-axis current component of the current "M1 MCAT Iq Required" generates torque and, by its application, the motor starts running. By changing the polarity of the current "M1 MCAT Iq Required", the motor changes the direction of rotation. Supposing the BEMF observer is tuned correctly, the current PI controllers can be tuned using the current FOC control structure.

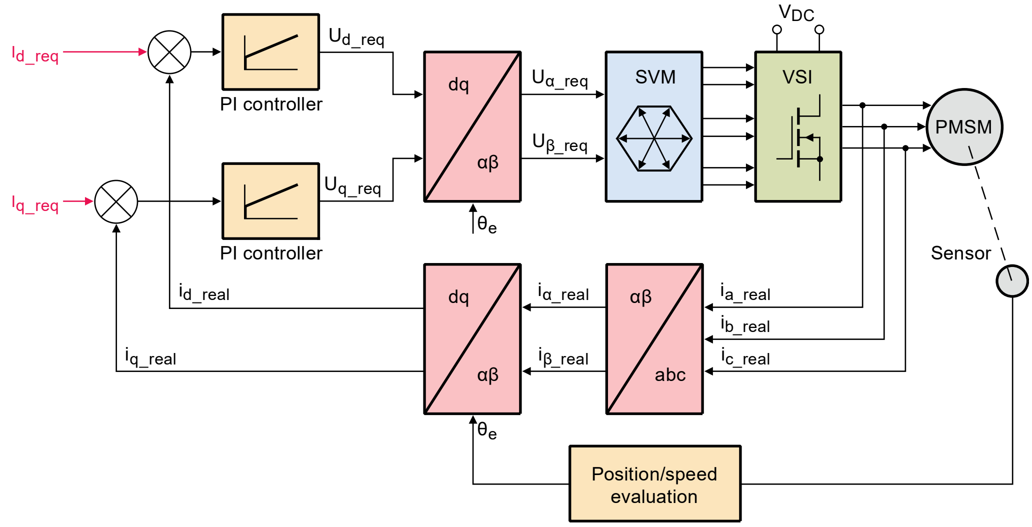


Figure 22. Current (torque) control mode

For run motor in current control, follow these steps:

1. Switch project tree subblock on "Current Control".
2. Switch variable "M1 MCAT Control" on "CURRENT_FOC".
3. In variable "M1 MCAT Iq Required" and "M1 MCAT Id Required" set required currents.
4. Set variable "M1 Application Switch" to "1". Motor start spinning.
5. Observe motor speed, position, phase currents and other graphs predefined in subblock scopes and recorders.

7.4.5 Speed FOC control

As shown in Figure 23, the speed PMSM sensor/sensorless FOC is activated by enabling the speed FOC control structure. Enter the required speed into the "M1 Speed Required" field. The d-axis current reference is held at 0 during the entire FOC operation.

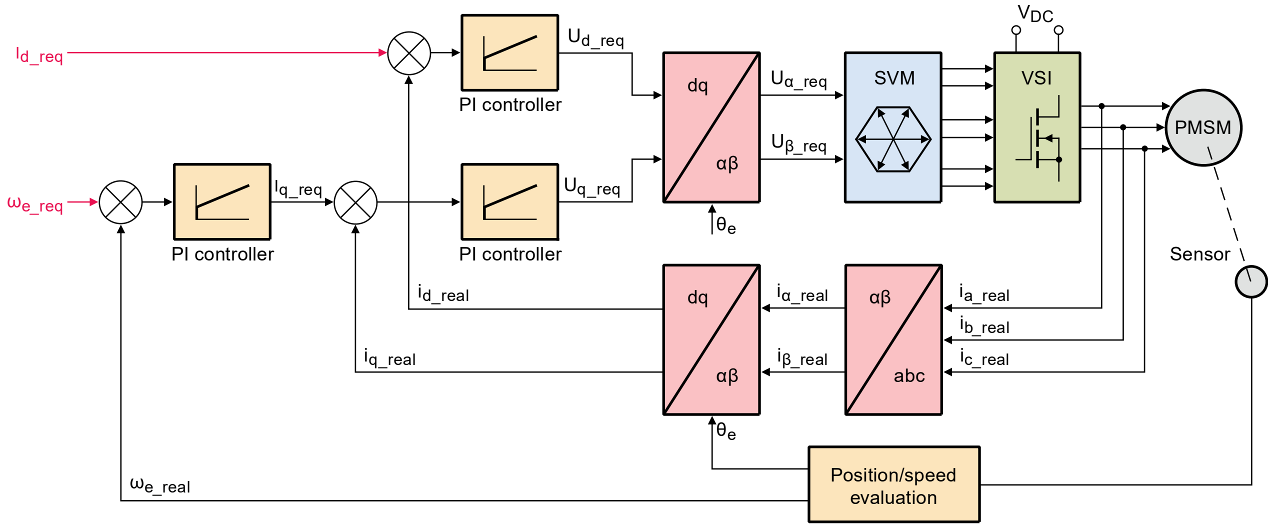


Figure 23. Speed FOC control mode

For run motor in speed FOC control, follow these steps:

1. Switch project tree subblock on "Speed Control".

2. Switch variable "M1 MCAT Control" on "SPEED_FOC".
3. Choose between sensored and sensorless control in variable "M1 MCAT POSPE Sensor".
4. In variable "M1 Speed Required" set the required speed. (i.e. 1000rpm). The motor automatically starts spinning.
5. Observe motor speed, position, phase currents and other graphs predefined in subblock scopes and recorders.

7.4.6 Position (servo) control

The position of PMSM sensor FOC is shown in Figure 24 (available for sensored/encoder based applications only). The position control using the P controller can be tuned in the "Speed loop" menu tab. An encoder sensor is required for the feedback. Without the sensor, the position control does not work. A braking resistor is missing on the FRDM-MC-LVPMSM board. Therefore, it is necessary to set a soft speed ramp (in the "Speed loop" menu tab) because the voltage on the DC-bus can rise when braking the quickly spinning shaft. It may cause an overvoltage fault.

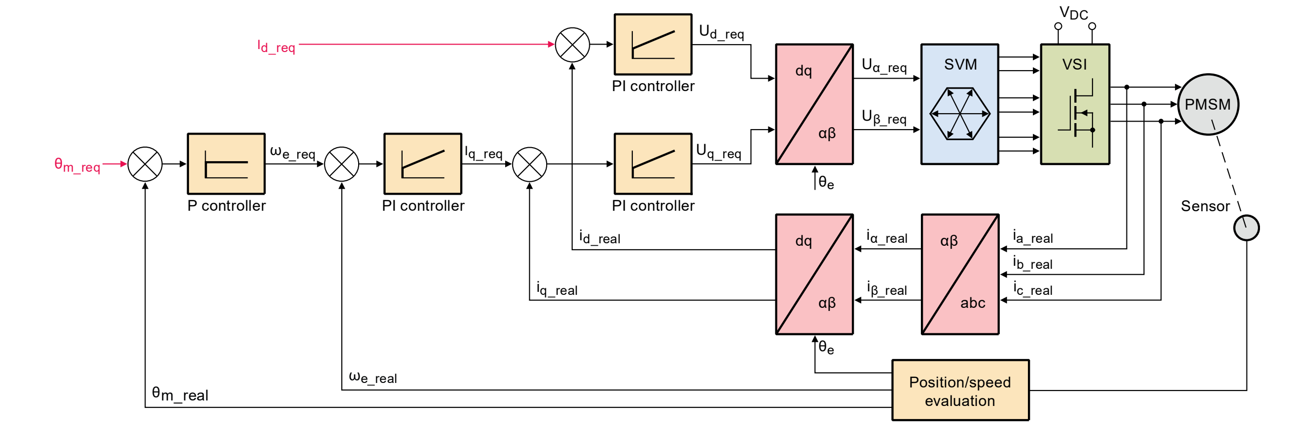


Figure 24. Position control mode

For run motor in position (servo) control, follow these steps:

1. Switch project tree subblock on "Position Control".
2. Switch variable "M1 MCAT Control" on "POSITION_CNTRL".
3. Swich variable "M1 MCAT POSPE Sensor" to "Encoder [1]".
4. In variable "M1 Position Required" set the required psition. (i.e. 10 revs).
5. Set variable "M1 Application Switch" to "1". The motor starts and automatically stops in the required position.
6. Observe motor speed, position, phase currents and other graphs predefined in subblock scopes and recorders.

7.5 Faults explanation

When the motor is running or during the tuning process, there may be several fault conditions. Therefore, the motor-control example has an integrated fault indication located in the variable watch of the "Motor M1" FreeMASTER subblock. If a fault is indicated, state machine enters the FAULT state.

Variable Watch

Name	Value	Unit
M1 Fault Pending	b# 0000 0000	BIN
M1 Fault Captured	b# 0000 0000	BIN
M1 Fault Captured Over Current	Not captured	ENUM
M1 Fault Captured DCBus Undervoltage	Not captured	ENUM
M1 Fault Captured DCBus Overvoltage	Not captured	ENUM
M1 Fault Captured Overload	Not captured	ENUM
M1 Fault Captured Overspeed	Not captured	ENUM
M1 Fault Captured Blocked Rotor	Not captured	ENUM
M1 Fault Clear	No request	ENUM
M1 Fault Enable DCBus Undervoltage	Enabled	ENUM
M1 Fault Enable DCBus Overvoltage	Enabled	ENUM
M1 Fault Enable Overload	Enabled	ENUM
M1 Fault Enable Overspeed	Enabled	ENUM
M1 Fault Enable Blocked Rotor	Enabled	ENUM

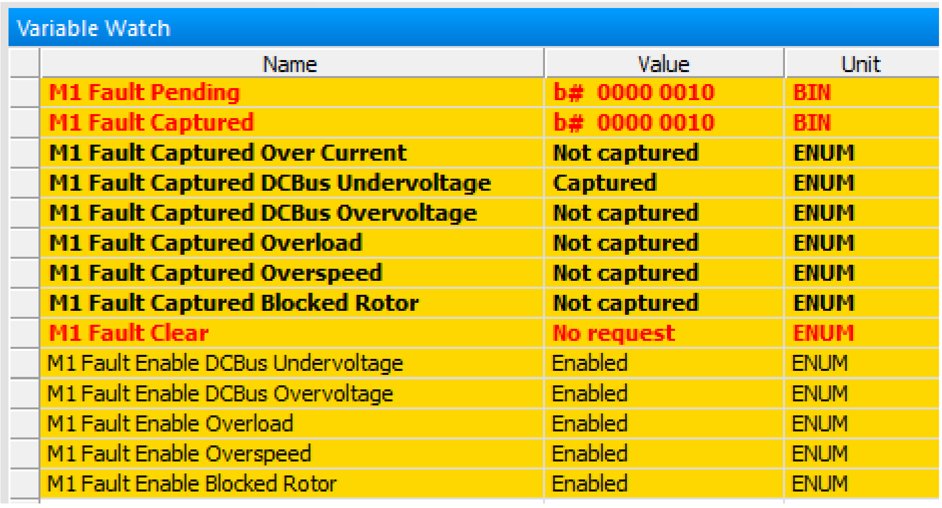
Figure 25. Faults in variable watch located in "Motor M1" subblock

7.5.1 Variable "M1 Fault Pending"

It shows actually persisting faults, which means that the fault indicated during fault conditions is accomplished. For example, if the source voltage is still under the undervoltage fault threshold, the undervoltage pending fault is shown. If the fault condition disappears, the fault pending is cleared automatically. "M1 Fault Pending" is shown in a binary format in the FreeMASTER variable watch. Each place in the variable denotes a different fault condition.

- b 0000 0001 - the overcurrent fault is indicated. If the overcurrent fault is present, the PWMs are automatically disabled. The fault occurs when the DC-Bus current exceeds the **Imax** value (current-sensing HW scale).
- b 0000 0010 - the undervoltage fault is indicated. The undervoltage fault occurs when the UDCBus voltage (source voltage) is lower than the **U DCB under** threshold.
- b 0000 0100 - the overvoltage fault is indicated. The overvoltage fault occurs when the UDCBus voltage (source voltage) is higher than the **U DCB over** threshold.
- b 0000 1000 - the overload fault is indicated. The overload fault occurs when the rotor is overloaded.
- b 0001 0000 - the overspeed fault is indicated. The overspeed fault occurs when the rotor speed exceeds the **N over** threshold.
- b 0010 0000 - the block rotor fault is indicated. The block rotor fault occurs when the back-EMF voltage is lower than the **E block** threshold and the duration of the drop is longer than **E block per**.

Figure 26. Undervoltage fault is indicated (pending)

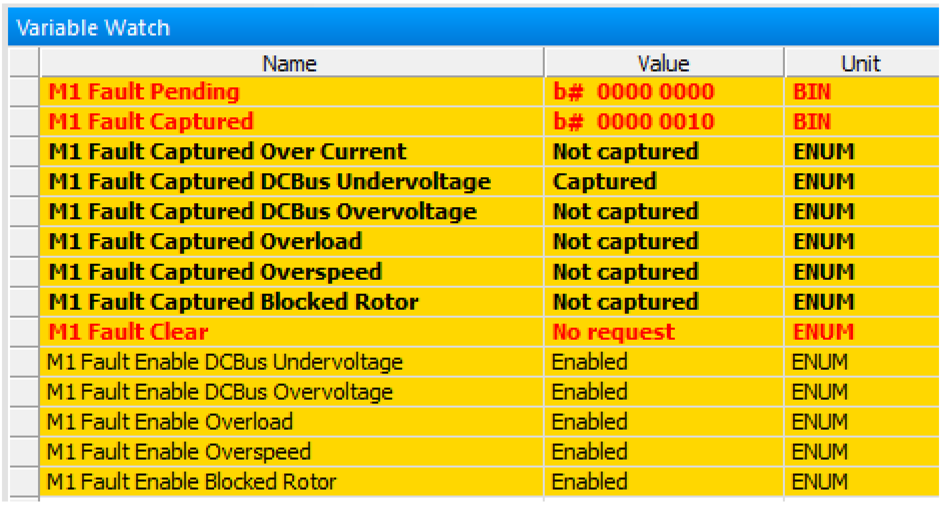


Variable Watch

Name	Value	Unit
M1 Fault Pending	b# 0000 0010	BIN
M1 Fault Captured	b# 0000 0010	BIN
M1 Fault Captured Over Current	Not captured	ENUM
M1 Fault Captured DCBus Undervoltage	Captured	ENUM
M1 Fault Captured DCBus Overvoltage	Not captured	ENUM
M1 Fault Captured Overload	Not captured	ENUM
M1 Fault Captured Overspeed	Not captured	ENUM
M1 Fault Captured Blocked Rotor	Not captured	ENUM
M1 Fault Clear	No request	ENUM
M1 Fault Enable DCBus Undervoltage	Enabled	ENUM
M1 Fault Enable DCBus Overvoltage	Enabled	ENUM
M1 Fault Enable Overload	Enabled	ENUM
M1 Fault Enable Overspeed	Enabled	ENUM
M1 Fault Enable Blocked Rotor	Enabled	ENUM

7.5.2 Variable "M1 Fault Captured"

If any fault condition appears, the fault captured is indicated. Similar to fault pending, fault captured is shown in the BIN format, but every fault type has its own variable ("M1 Fault Captured Over Curent" and others). For example, if the undervoltage fault condition is accomplished, fault captured is indicated. Fault captured is also indicated after the undervoltage fault condition disappears. The captured faults are cleared manually by writing "Clear [1]" to "M1 Fault Clear".



Variable Watch

Name	Value	Unit
M1 Fault Pending	b# 0000 0000	BIN
M1 Fault Captured	b# 0000 0010	BIN
M1 Fault Captured Over Current	Not captured	ENUM
M1 Fault Captured DCBus Undervoltage	Captured	ENUM
M1 Fault Captured DCBus Overvoltage	Not captured	ENUM
M1 Fault Captured Overload	Not captured	ENUM
M1 Fault Captured Overspeed	Not captured	ENUM
M1 Fault Captured Blocked Rotor	Not captured	ENUM
M1 Fault Clear	No request	ENUM
M1 Fault Enable DCBus Undervoltage	Enabled	ENUM
M1 Fault Enable DCBus Overvoltage	Enabled	ENUM
M1 Fault Enable Overload	Enabled	ENUM
M1 Fault Enable Overspeed	Enabled	ENUM
M1 Fault Enable Blocked Rotor	Enabled	ENUM

Figure 27. Undervoltage fault is captured

7.5.3 Variable "M1 Fault Enable"

The fault indication can be undwanted during the tuning process. Therefore, the fault indication can be disabled by writing "Disabled [0]" to the "M1 Fault Enable" variables.

Note: *The overcurrent fault cannot be disabled.*

Note: *Fault thresholds are located in the "MCAT parameters" tab.*

7.6 Initial motor parameters and harware configuration

Motor control examples contain two or more configuration files: `m1_pmsm_appconfig.h`, `m2_pmsm_appconfig.h`, and so on. Each contains constants tuned for the selected motor (Linix 45ZWN24-40 or Teknic M-2310P for the Freedom development platform and Mige 60CST-MO1330 for the High-voltage platform). The initial motor parameters and the hardware configuration (inverter) are to MCAT loaded from `m1_pmsm_appconfig.h` configuration file. There are tree ways to change motor configuration corresponding to the connected motor.

1. The first way is rename the configuration file:
   - In the project example folder, find configuration file to be used.
   - Rename this configuration file to `m1_pmsm_appconfig.h`.
   - Rebuild project and load the code to the MCU.
2. The second way is to change motor configuration, as described in Section 7.3.
3. The last way is change motor and hardware parameters manually:
   - Open the PMSM control application FreeMASTER project containing the dedicated MCAT plug-in module.
   - Select the "Parameters" tab.
   - Specify the parameters manually. The motor parameters can be obtained from the motor data sheet or using the PMSM parameters measurement procedure described in *PMSM Electrical Parameters Measurement* (document AN4680). All parameters provided in Table 11 are accessible. The motor inertia J expresses the overall system inertia and can be obtained using a mechanical measurement. The J parameter is used to calculate the speed controller constant. However, the manual controller tuning can also be used to calculate this constant.

Table 11. MCAT motor parameters

Parameter	Units	Description	Typical range
pp	[-]	Motor pole pairs	1-10
Rs	[Ω]	1-phase stator resistance	0.3-50
Ld	[H]	1-phase direct inductance	0.00001-0.1
Lq	[H]	1-phase quadrature inductance	0.00001-0.1
Ke	[V.sec/rad]	BEMF constant	0.001-1
J	[$kg.m^2$]	System inertia	0.00001-0.1
Iph nom	[A]	Motor nominal phase current	0.5-8
Uph nom	[V]	Motor nominal phase voltage	10-300
N nom	[rpm]	Motor nominal speed	1000-2000

   - Set the hardware scales—the modification of these two fields is not required when a reference to the standard power stage board is used. These scales express the maximum measurable current and voltage analog quantities.
   - Check the fault limits—these fields are calculated using the motor parameters and hardware scales (see Table 12).

Table 12. Fault limits

Parameter	Units	Description	Typical range
U DCB trip	[V]	Voltage value at which the external braking resistor switch turns on	U DCB Over ~ U DCB max

Table 12. Fault limits...*continued*

Parameter	Units	Description	Typical range
U DCB under	[V]	Trigger value at which the undervoltage fault is detected	0 ~ U DCB Over
U DCB over	[V]	Trigger value at which the overvoltage fault is detected	U DCB Under ~ U max
N over	[rpm]	Trigger value at which the overspeed fault is detected	N nom ~ N max
N min	[rpm]	Minimal actual speed value for the sensorless control	(0.05~0.2) *N max

- Check the application scales—these fields are calculated using the motor parameters and hardware scales (see Table 13).

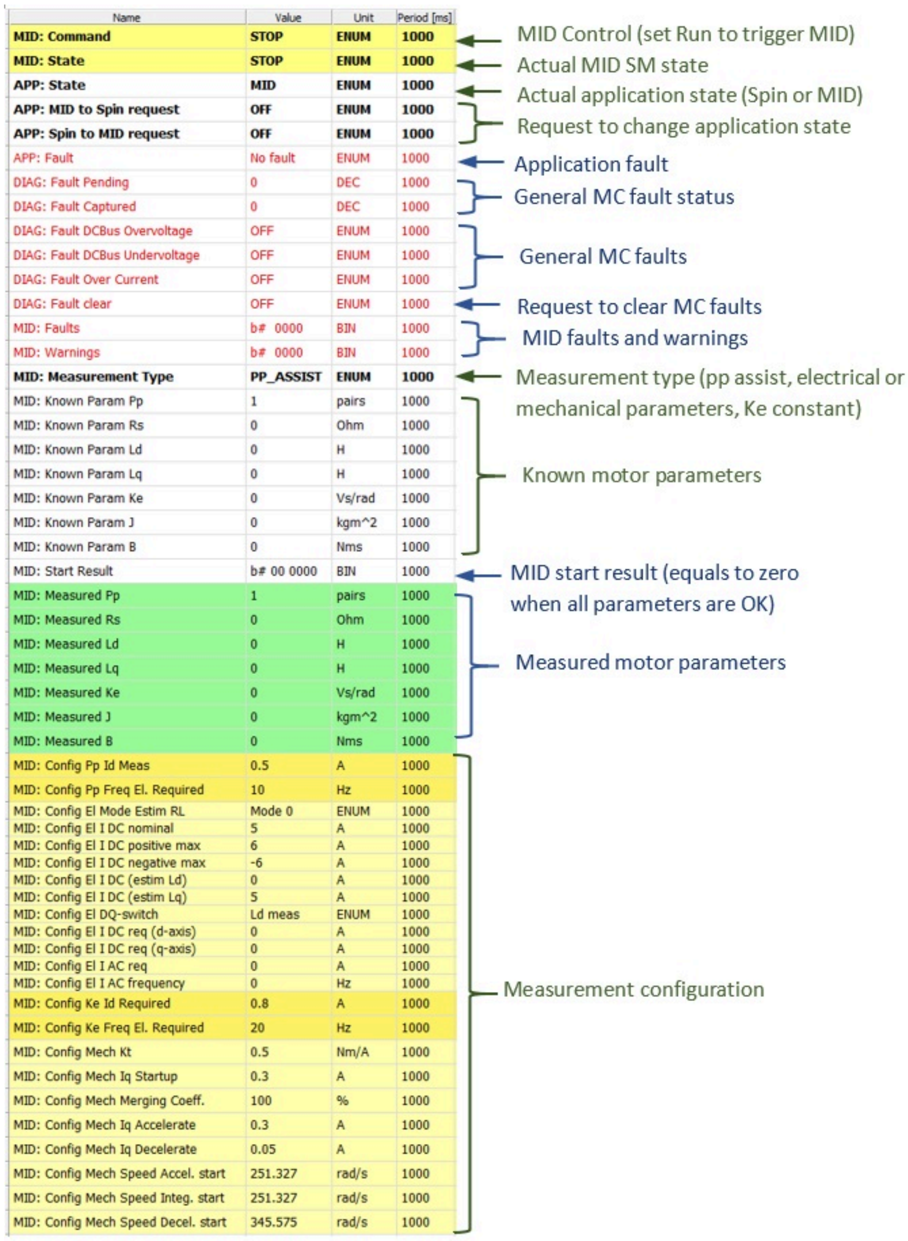
Table 13. Application scales

Parameter	Units	Description	Typical range
N max	[rpm]	Speed scale	>1.1 * N nom
E block	[V]	BEMF scale	ke* Nmax
kt	[Nm/A]	Motor torque constant	-

- Check the alignment parameters—these fields are calculated using the motor parameters and hardware scales. The parameters express the required voltage value applied to the motor during the rotor alignment and its duration.
- To save the modified parameters into the inner file, click the "Store data" button.

7.7 Identifying parameters of user motor

Because the model-based control methods of the PMSM drives provide high performance (for example, dynamic response, efficiency), obtaining an accurate model of a motor is an important part of the drive design and control. For the implemented FOC algorithms, it is necessary to know the value of the stator resistance R_s, direct inductance L_d, quadrature inductance L_q, and BEMF constant K_e. Unless the default PMSM motor described above is used, the motor parameter identification is the first step in the application tuning. This section shows how to identify user motor parameters using MID. MID is written in floating-point arithmetics. Each MID algorithm is detailed in Section 7.8. MID is controlled via the FreeMASTER "Motor Identification" page shown in Figure 28.



Name	Value	Unit	Period [ms]
MID: Command	STOP	ENUM	1000
MID: State	STOP	ENUM	1000
APP: State	MID	ENUM	1000
APP: MID to Spin request	OFF	ENUM	1000
APP: Spin to MID request	OFF	ENUM	1000
APP: Fault	No fault	ENUM	1000
DIAG: Fault Pending	0	DEC	1000
DIAG: Fault Captured	0	DEC	1000
DIAG: Fault DCBus Overvoltage	OFF	ENUM	1000
DIAG: Fault DCBus Undervoltage	OFF	ENUM	1000
DIAG: Fault Over Current	OFF	ENUM	1000
DIAG: Fault clear	OFF	ENUM	1000
MID: Faults	b# 0000	BIN	1000
MID: Warnings	b# 0000	BIN	1000
MID: Measurement Type	PP_ASSIST	ENUM	1000
MID: Known Param Pp	1	pairs	1000
MID: Known Param Rs	0	Ohm	1000
MID: Known Param Ld	0	H	1000
MID: Known Param Lq	0	H	1000
MID: Known Param Ke	0	Vs/rad	1000
MID: Known Param J	0	kgm^2	1000
MID: Known Param B	0	Nms	1000
MID: Start Result	b# 00 0000	BIN	1000
MID: Measured Pp	1	pairs	1000
MID: Measured Rs	0	Ohm	1000
MID: Measured Ld	0	H	1000
MID: Measured Lq	0	H	1000
MID: Measured Ke	0	Vs/rad	1000
MID: Measured J	0	kgm^2	1000
MID: Measured B	0	Nms	1000
MID: Config Pp Id Meas	0.5	A	1000
MID: Config Pp Freq El. Required	10	Hz	1000
MID: Config El Mode Estim RL	Mode 0	ENUM	1000
MID: Config El I DC nominal	5	A	1000
MID: Config El I DC positive max	6	A	1000
MID: Config El I DC negative max	-6	A	1000
MID: Config El I DC (estim Ld)	0	A	1000
MID: Config El I DC (estim Lq)	5	A	1000
MID: Config El DQ-switch	Ld meas	ENUM	1000
MID: Config El I DC req (d-axis)	0	A	1000
MID: Config El I DC req (q-axis)	0	A	1000
MID: Config El I AC req	0	A	1000
MID: Config El I AC frequency	0	Hz	1000
MID: Config Ke Id Required	0.8	A	1000
MID: Config Ke Freq El. Required	20	Hz	1000
MID: Config Mech Kt	0.5	Nm/A	1000
MID: Config Mech Iq Startup	0.3	A	1000
MID: Config Mech Merging Coeff.	100	%	1000
MID: Config Mech Iq Accelerate	0.3	A	1000
MID: Config Mech Iq Decelerate	0.05	A	1000
MID: Config Mech Speed Accel. start	251.327	rad/s	1000
MID: Config Mech Speed Integ. start	251.327	rad/s	1000
MID: Config Mech Speed Decel. start	345.575	rad/s	1000

Figure 28. MID FreeMASTER control

7.7.1 Switch between Spin and MID

Users can switch between two modes of application: *Spin* and *MID* (Motor Identification). *Spin* mode is used for control PMSM (see Section 7.3). *MID* mode is used for motor parameters identification (see Section 7.7.2). The actual mode of application is shown in *APP: State* variable. The mode is changed by writing one to *APP: MID to Spin request* or *APP: Spin to MID request* variables. The transition between Spin and MID can be done only if the actual mode is in a defined stop state (for example, MID not in progress or motor stopped). The result of the change mode request is shown in *APP: Fault* variable. MID fault occurs when parameters identification still runs, or the MID state machine is in the fault state. A spin fault occurs when *M1 Application switch* variable watch is ON, or *M1 Application state* variable watch is not STOP.

7.7.2 Motor parameter identification using MID

The whole MID is controlled via the FreeMASTER "Variable Watch". The Motor Identification (MID) subblock is shown in Figure 28. Following is the motor parameter identification workflow:

1. Set the *MID: Command* variable to STOP.
2. Select the measurement type that you want to perform via the *MID: Measurement Type* variable:
   - `PP_ASSIST` - Pole-pair identification assistant
   - `EL_PARAMS` - Electrical parameters measurement
   - `Ke – BEMF` constant measurement
   - `MECH_PARAMS` - Mechanical parameters measurement
3. Insert the known motor parameters via the *MID: Known Param* set of variables. All parameters with a non-zero known value are used instead of measured parameters (if necessary).
4. Set the measurement configuration parameters in the *MID: Config* set of variables.
5. Start the measurement by setting *MID: Command* to RUN.
6. Observe the *MID Start Result* variable for the MID measurement plan validity (see Table 16) and the actual *MID: State*, *MID: Faults* (see Table 14), and *MID: Warnings* (see Table 15) variables.
7. If the measurement finishes successfully, the measured motor parameters are shown in the *MID: Measured* set of variables and *MID: State* goes to STOP.

7.7.3 MID faults and warnings

The MID faults and warnings are saved in the format of masks in the *MID: Faults* and *MID: Warnings* variables. Faults and warnings are cleared automatically when starting a new measurement. If a MID fault appears, the measurement process immediately stops and brings the MID state machine safely to the STOP state. If a MID warning appears, the measurement process continues. Warnings report minor issues during the measurement process. For more details on individual faults and warnings, see Table 14 and Table 15.

Table 14. Measurement faults

Fault mask	Fault description	Fault reason	Troubleshooting
b#0001	Electrical parameters measurement fault	Some required value cannot be reached or wrong measurement configuration	Check whether measurement configuration is valid
b#0010	Mechanical measurement timeout	Some part of the mechanical measurement (acceleration, deceleration) took too long and exceeded 10 seconds	Raise the MID: Config Mech Iq Accelerate or lower the MID: Config Mech Iq Decelerate

Table 15. Measurement warnings

Warning mask	Warning description	Warning reason	Troubleshooting
b#0001	K_e is out of range	The measured K_e is negative	Visually check whether the motor was spinning properly during the K_e measurement

The MID measurement plan is checked after starting the measurement process. If a necessary parameter is not scheduled for the measurement and not set manually, the MID is not started and an error is reported via the *MID: Start Result* variable.

Table 16. MID Start Result variable

MID Start Result mask	Description	Troubleshooting
b#00 0001	Error during initialization electrical parameters measurement	Check whether inputs to the MCAA_ EstimRLInit_FLT are valid
b#00 0010	The R_s value is missing	Schedule electrical measurement or enter R_s value manually
b#00 0100	The L_d value is missing	Schedule electrical measurement or enter L_d value manually
b#00 1000	The L_q value is missing	Schedule electrical measurement or enter L_q value manually
b#01 0000	The K_e value is missing	Schedule K_e for measurement or enter its value manually
b#10 0000	The *Pp* value is missing	Enter the *Pp* value manually

7.8 MID algorithms

This section describes how each MID algorithm works.

7.8.1 Stator resistance measurement

The stator resistance R_s is averaged from the DC steps generated by the algorithm. The DC step levels are automatically derived from the currents inserted by the user. For more details, refer to the documentation of AMCLIB_EstimRL function from AMMCLib.

7.8.2 Stator inductances measurement

Injection of the AC-DC currents is used for the inductances (L_d *and* L_q) estimation. Injected AC-DC currents are automatically derived from the currents inserted by the user. The default AC current frequency is 500 Hz. For more detail, refer to the documentation of AMCLIB_EstimRL function from AMMCLib.

7.8.3 BEMF constant measurement

Before the actual BEMF constant K_e measurement, the BEMF and Tracking observers parameters are recalculated from the previously measured or manually set R_s, L_d, and L_q parameters. To measure K_e, the motor must spin. During the measurement, the motor is open-loop driven at the user-defined frequency *MID: Config Ke Freq El. Required* with the user-defined current *MID: Config Ke Id Required* value. When the motor reaches the required speed, the BEMF voltages obtained by the BEMF observer are filtered and K_e is calculated:

$$K_e = \frac{U_{BEMF}}{\omega_{el}} \left[\Omega\right] \tag{5}$$

When K_e is being measured, you must visually check whether the motor is spinning properly. If the motor is not spinning properly, perform the steps below:

- Ensure that the number of *pp* is correct. The required speed for the K_e measurement is also calculated from *pp*. Therefore, inaccuracy in *pp* causes inaccuracy in the resulting K_e.
- Increase *MID: Config Ke Id Required* variable to produce higher torque when spinning during the open loop.
- Decrease *MID: Config Ke Freq El. Required* variable to decrease the required speed for the K_e measurement.

7.8.4 Number of pole-pair assistant

The number of pole-pairs can only be measured with a position sensor. However, there is a simple assistant to determine the number of pole-pairs (`PP_ASSIST`). The number of the pp assistant performs one electrical revolution, stops for a few seconds, and then repeats. Because the pp value is the ratio between the electrical and mechanical speeds, it can be determined as the number of stops per one mechanical revolution. It is recommended to refrain from counting the stops during the first mechanical revolution because the alignment occurs during the first revolution and affects the number of stops. During the `PP_ASSIST` measurement, the current loop is enabled, and the I_d current is controlled to *MID: Config Pp Id Meas*. The electrical position is generated by integrating the open-loop frequency *MID: Config Pp Freq El. Required*. If the rotor does not move after the start of `PP_ASSIST` assistant, stop the assistant, increase *MID: Config Pp Id Meas*, and restart the assistant.

7.8.5 Mechanical parameters measurement

The moment of inertia *J* and the viscous friction *B* can be identified using a test with the known generated torque *T* and the loading torque T_{load}.

$$\frac{d\omega_m}{dt} = \frac{1}{J}\left(T - T_{load} - B_{\omega_n}\right)\ \left[rad/s^2\right] \tag{6}$$

The ω_m character in the equation is the mechanical speed. The mechanical parameter identification software uses the torque profile. The loading torque is (for simplicity reasons) said to be 0 during the whole measurement. Only the friction and the motor-generated torque are considered. During the measurement phase, the constant torque T_{meas} is applied and the motor accelerates to 50 % of its nominal speed in time t_1. These integrals are calculated during the period from t_0 (the speed estimation is accurate enough) to t_1:

$$T_{int} = \int_{t_0}^{t_1} T dt\ \left[Nms\right] \tag{7}$$

$$\omega_{int} = \int_{t_0}^{t_1} \omega_m dt\ \left[rad/s\right] \tag{8}$$

During the second phase, the rotor decelerates freely with no generated torque, only by friction. This enables you to measure the mechanical time constant $\tau_m = J/B$ as the time the rotor decelerates from its original value by 63 %.

The final mechanical parameter estimation can be calculated by integrating:

$$\omega_m(t_1) = \frac{1}{J} T_{int} - \frac{B}{J}\omega_{int} + \omega_m(t_0)\ \left[rad/s\right] \tag{9}$$

The moment of inertia is:

$$J = \frac{\tau_m T_{int}}{\tau_m\left[\omega_m(t_1) - \omega_m(t_0)\right] + \omega_{int}}\ \left[kgm^2\right] \tag{10}$$

The viscous friction is then derived from the relation between the mechanical time constant and the moment of inertia. To use the mechanical parameters measurement, the current control loop bandwidth $f_{0,Current}$, the speed control loop bandwidth $f_{0,Speed}$, and the mechanical parameters measurement torque Trq_m must be set.

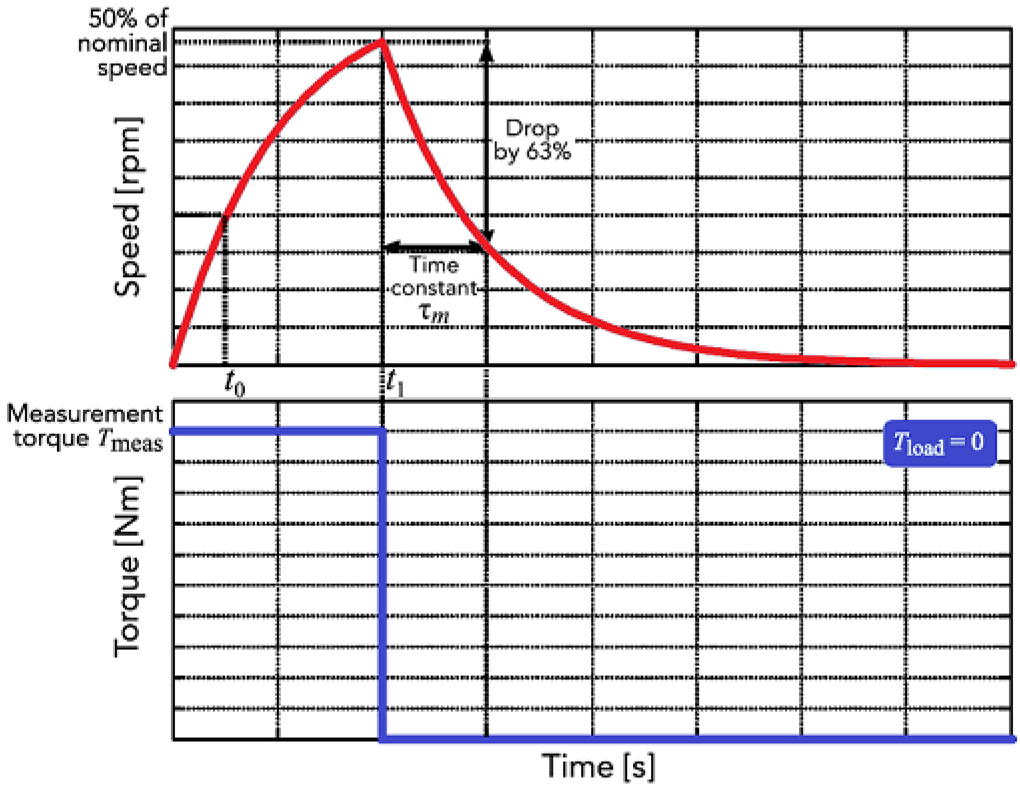


Figure 29. PMSM identification tab

7.9 Electrical parameters measurement control

This section describes how to control electrical parameters measurement, which contains measuring stator resistance R_s, direct inductance L_d, and quadrature inductance L_q. There are available 4 modes of measurement which `MID: Config El Mode Estim RL` variable can select.

Function `MCAA_EstimRLInit_FLT` must be called before the first use of `MCAA_EstimRL_FLT`. Function `MCAA_EstimRL_FLT` must be called periodically with sampling period `F_SAMPLING`, which can be defined by the user. Maximum sampling frequency `F_SAMPLING` is 10 kHz. In the scopes under "Motor identification", FreeMASTER subblock can be observed in measured currents, estimated parameters, and so on.

7.9.1 Mode 0

This mode is automatic. Inductances are measured at a single operating point. The rotor is not fixed. The user has to specify nominal current (`MID: Config El I DC nominal` variable). The AC and DC currents are automatically derived from the nominal current. The frequency of the AC signal is set to 500 Hz.

The function outputs stator resistance R_s, direct inductance L_d, and quadrature inductance L_q.

7.9.2 Mode 1

DC stepping is automatic in this mode. The rotor is not fixed. Compared to the Mode 0, an automatic measurement of the inductances for a defined number (*NUM_MEAS*) of different DC current levels is performed using positive values of the DC current. The L_{dq} dependency map can be seen in the "Inductances (Ld, Lq)" recorder. The user has to specify the following parameters before parameters estimation:

- *MID: Config El I DC (estim Lq)* - Current to determine L_q. In most cases, nominal current.

- *MID: Config El I DC positive max* - Maximum positive DC current for the L_{dq} dependency map measurement.

Injected AC and DC currents are automatically derived from the *MID: Config El I DC (estim Lq)* and *MID: Config El I DC positive max* currents. The frequency of the AC signal is set to 500 Hz.

The function outputs stator resistance R_s, direct inductance L_d, quadrature inductance L_q, and L_{dq} dependency map.

7.9.3 Mode 2

DC stepping is automatic in this mode. The rotor must be mechanically fixed after initial alignment with the first phase. Compared to the Mode 1, an automatic measurement of the inductances for a defined number (*NUM_MEAS*) of different DC current levels is performed using both positive and negative values of the DC current. The estimated inductances can be seen in the "Inductances (Ld, Lq)" recorder. The user has to specify following parameters before parameters estimation:

- *MID: Config El I DC (estim Ld)* - Current to determine L_d. In most cases, 0 A.
- *MID: Config El I DC (estim Lq)* - Current to determine L_q. In most cases, nominal current.
- *MID: Config El I DC positive max* - Maximum positive DC current for the L_{dq} dependency map measurement. In most cases, nominal current.
- *MID: Config El I DC negative max* - Maximum negative DC current for the L_{dq} dependency map measurement.

Injected AC and DC currents are automatically derived from the *MID: Config El I DC (estim Ld)*, *MID: Config El I DC (estim Lq)*, *MID: Config El I DC positive max*, and *MID: Config El I DC negative max* currents. The frequency of the AC signal is set to 500 Hz.

The function outputs stator resistance R_s, direct inductance L_d, quadrature inductance L_q, and L_{dq} dependency map.

7.9.4 Mode 3

This mode is manual. The rotor must be mechanically fixed after alignment with the first phase. R_s is not calculated at this mode. The estimated inductances can be observed in the "Ld" or "Lq" scopes. The following parameters can be changed during the runtime:

- *MID: Config El DQ-switch* - Axis switch for AC signal injection (0 for injection AC signal to d-axis, 1 for injection AC signal to q-axis).
- *MID: Config El I DC req (d-axis)* - Required DC current in d-axis.
- *MID: Config El I DC req (q-axis)* - Required DC current in q-axis.
- *MID: Config El I AC req* - Required AC current injected to the d-axis or q-axis.
- *MID: Config El I AC frequency* - Required frequency of the AC current injected to the d-axis or q-axis.

7.10 Control parameters tuning

To check correct current measuring and proper working of back EMF observer, follow the steps below:

1. Select the scalar control in the "M1 MCAT Control" FreeMASTER variable watch.
2. Set the "M1 Application Switch" variable to "ON". The application state changes to "RUN".
3. Set the required frequency value in the "M1 Scalar Freq Required" variable; for example, 15 Hz in the "Scalar & Voltage Control" FreeMASTER project tree. The motor starts running.
4. Select the "Phase Currents" recorder from the "Scalar & Voltage Control" FreeMASTER project tree.
5. The optimal ratio for the V/Hz profile can be found by changing the V/Hz factor directly using the "M1 V/Hz factor" variable. The shape of the motor currents should be close to a sinusoidal shape (Figure 30). Use the following equation for calculating V/Hz factor:

$$VHz_{factor} = \frac{U_{phnom} \times k_{factor}}{\frac{pp \times N_{nom}}{60} \times 100} \left[V \middle/ Hz \right] \qquad (11)$$

Where,
U_{phnom} = nominal voltage
k_{factor} = ratio within range 0-100%
pp = number of pole-pairs
N_{nom} = nominal revolutions
Note: *Changes V/Hz factor is not propagated to the* `m1_pmsm_appconfig.h`.

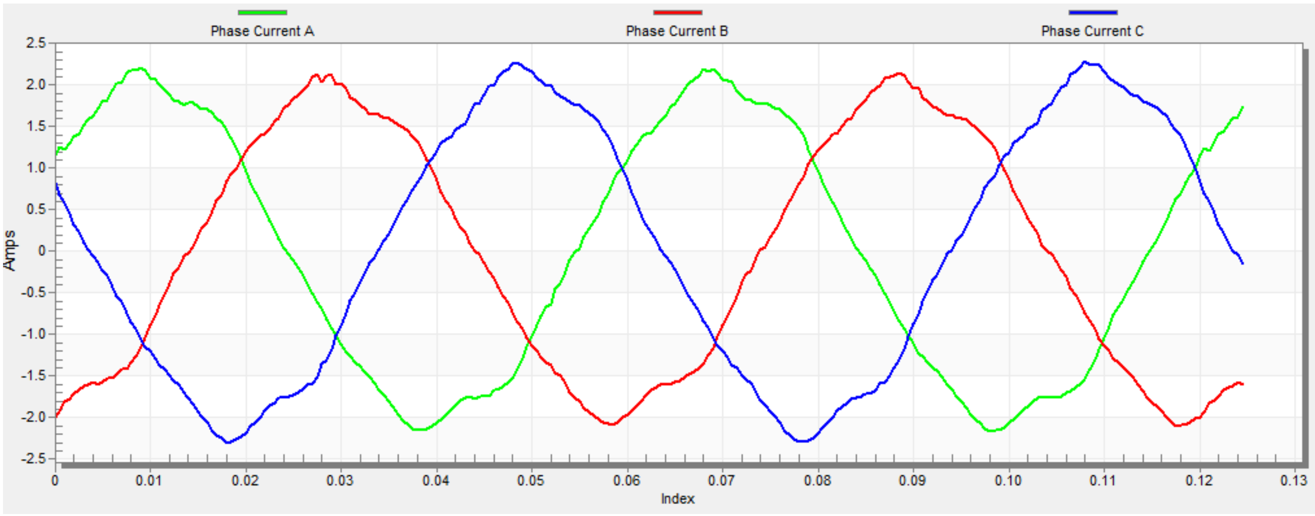


Figure 30. Phase currents

6. Select the "Position" recorder to check the observer functionality. The difference between the "Position Electrical Scalar" and the "Position Estimated" should be minimal (see Figure 31) for the Back-EMF position and speed observer to work properly. The position difference depends on the motor load. The higher the load, the bigger the difference between the positions due to the load angle.

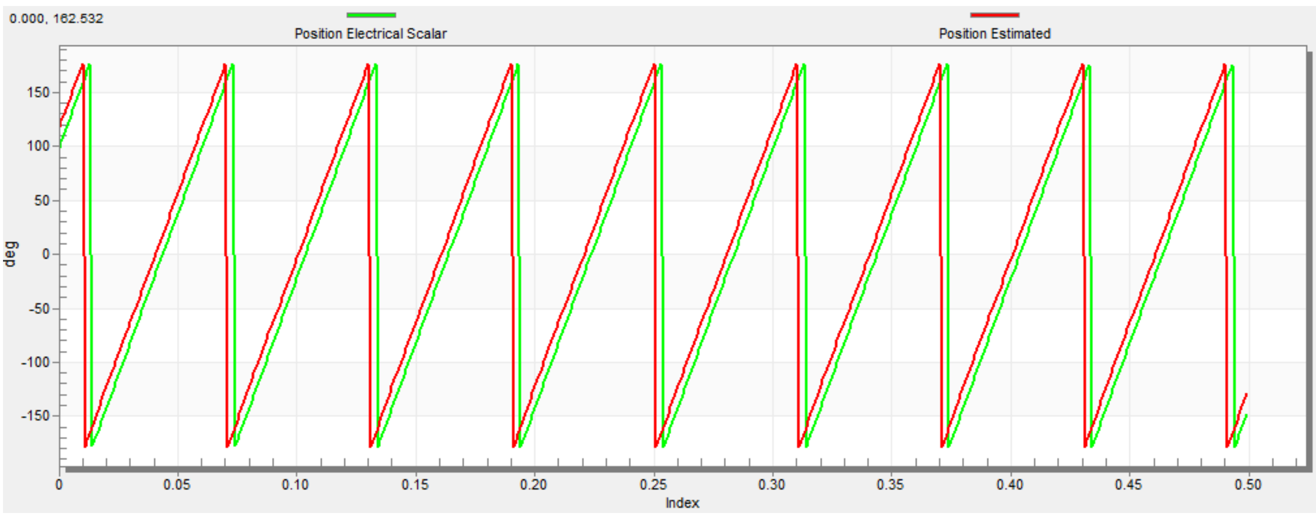


Figure 31. Generated and estimated positions

7. If an opposite speed direction is required, set a negative speed value into the "M1 Scalar Freq Required" variable.
8. The proper observer functionality and the measurement of analog quantities is expected at this step.

9. Enable the voltage FOC mode in the "M1 MCAT Control" variable while the main application switch "M1 Application Switch" is turned off.
10. Switch on the main application switch on and set a non-zero value in the “M1 MCAT Uq Required” variable. The FOC algorithm uses the estimated position to run the motor.

7.10.1 Encoder sensor setting

The encoder sensor settings are in the "Sensors" tab. The encoder sensor enables you to compute speed and position for the sensored speed. For a proper encoder counting, set the number of encoder pulses per one revolution and the proper counting direction. The number of encoder pulses is based on information about the encoder from its manufacturer. If the encoder sensor has more pulses per revolution, the speed and position computing is more accurate. The counting direction is provided by connecting the encoder signals to the NXP Freedom board and also by connecting the motor phases.

To determine the direction of rotation, follow the steps below:

1. Navigate to the "Scalar & Voltage Control" tab in the project tree and select "SCALAR_CONTROL" in the "M1 MCAT Control" variable.
2. Turn on the application switch. The application state changes to "RUN".
3. Set the required frequency value in the "M1 Scalar Freq Required" variable; for example, 15 Hz. The motor starts running.
4. Check the encoder direction. Select the "Encoder Direction Scope" from the "Scalar & Voltage Control" project tree. If the encoder direction is right, the estimated speed is equal to the measured mechanical speed. If the measured mechanical speed is opposite to the estimated speed, the direction must be changed. The first way is to change "M1 Encoder Direction" variable - only 0 or 1 value is allowed. The second way is invert the encoder wires—phase A and phase B (or the other way round).

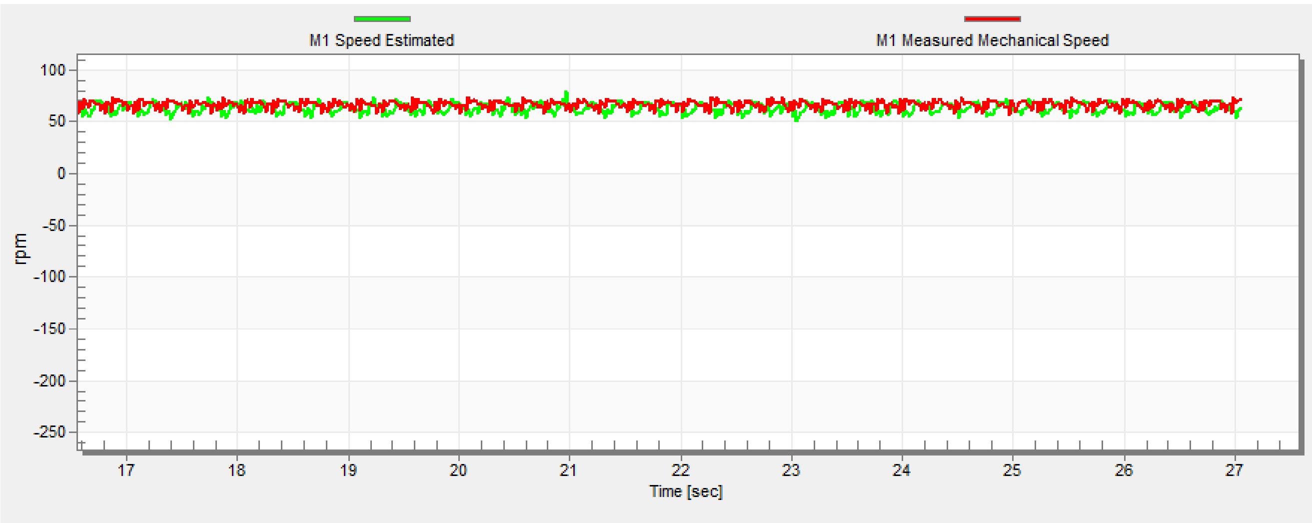


Figure 32. Encoder direction—right direction

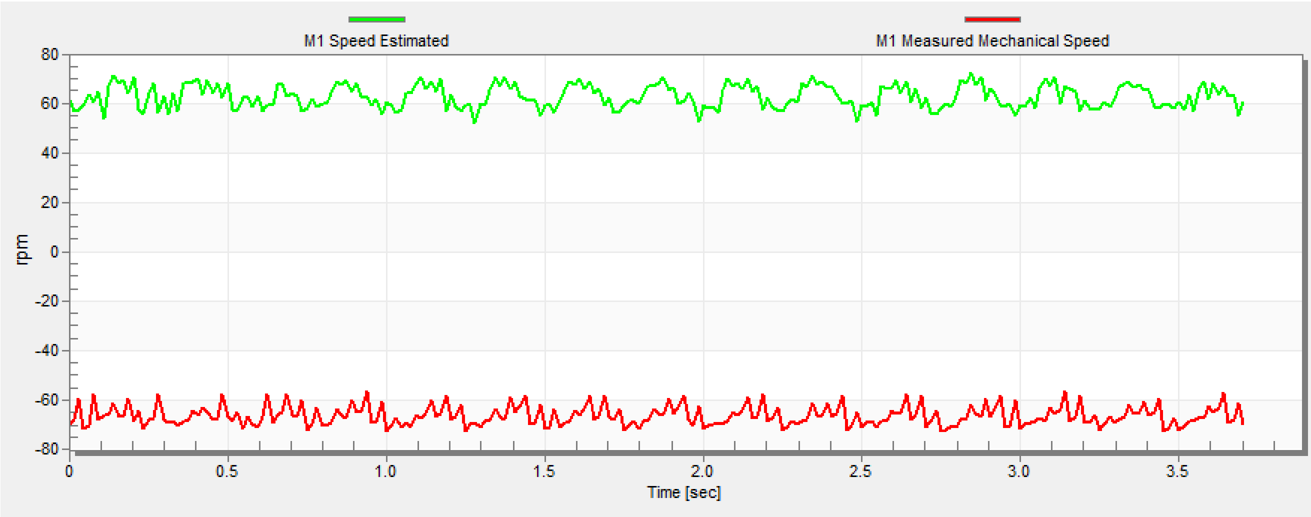


Figure 33. Encoder direction—wrong direction

7.10.2 Alignment tuning

For the alignment parameters, navigate to the "Parameters" MCAT tab. The alignment procedure sets the rotor to an accurate initial position and enables you to apply full startup torque to the motor. A correct initial position is needed mainly for high startup loads (compressors, washers, and so on). The alignment aims to have the rotor in a stable position, without any oscillations before the startup.

- The alignment voltage is the value applied to the d-axis during the alignment. Increase this value for a higher shaft load.
- The alignment duration expresses the time when the alignment routine is called. Tune this parameter to eliminate rotor oscillations or movement at the end of the alignment process.

7.10.3 Current loop tuning

The parameters for the current D, Q, and PI controllers are fully calculated using the motor parameters and no action is required in this mode. If the calculated loop parameters do not correspond to the required response, the bandwidth and attenuation parameters can be tuned.

1. Select “Openloop Control” in the FreeMASTER project tree, set “M1 MCAT Control” to “OPENLOOP_CTRL” and switch “M1 Openloop Use I Control” on.
2. Turn the application on by switching “M1 Application Switch” on and then set “M1 Openloop Requred Id” for rotor alignment. (Rotor alignment always uses Id, even when you are tuning the Q axis regulator)
3. Mechanically lock the motor schaft and turn the application off.
4. Set the required loop bandwidth and attenuation in MCAT “Current loop” tab and then click the “Update target” button. The tuning loop bandwidth parameter defines how fast the loop response is while the tuning loop attenuation parameter defines the actual overshoot magnitude.
5. Select “Current Controller Id” recorder in project tree, turn the application on and set the required step amplitude in “M1 Openloop Requred Id”. Observe the step response in the recorder.
6. Tune the loop bandwidth and attenuation until you achieve the required response. The example waveforms show the correct and incorrect settings of the current loop parameters:
   - The loop bandwidth is low (100 Hz) and the settling time of the Id current is long (Figure 1).

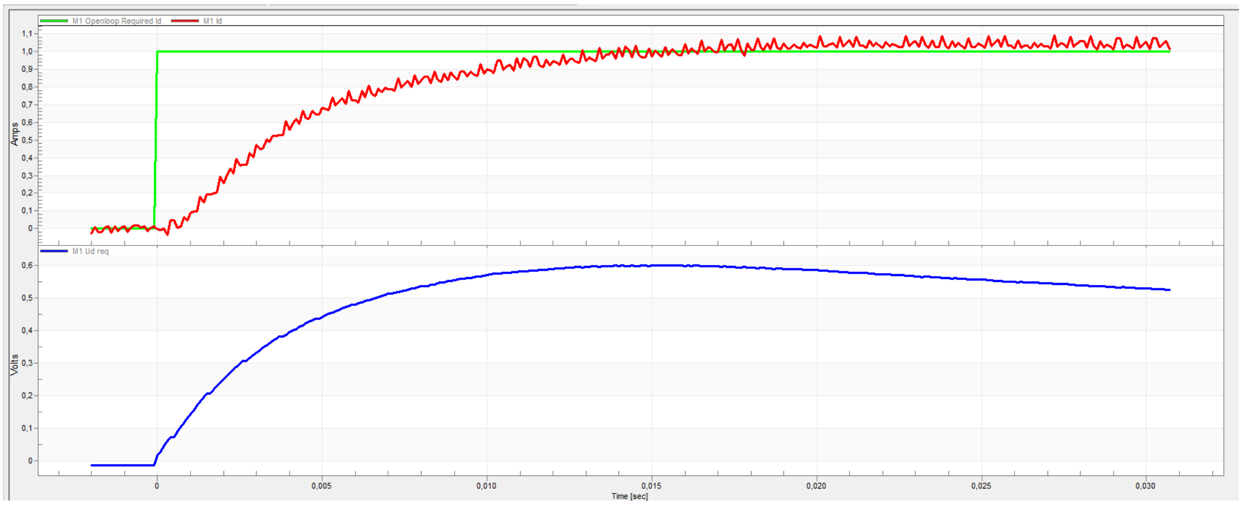


Figure 34. Slow step response of the Id current controller

- The loop bandwidth (300 Hz) is optimal and the response time of the Id current is sufficient (see Figure 2).

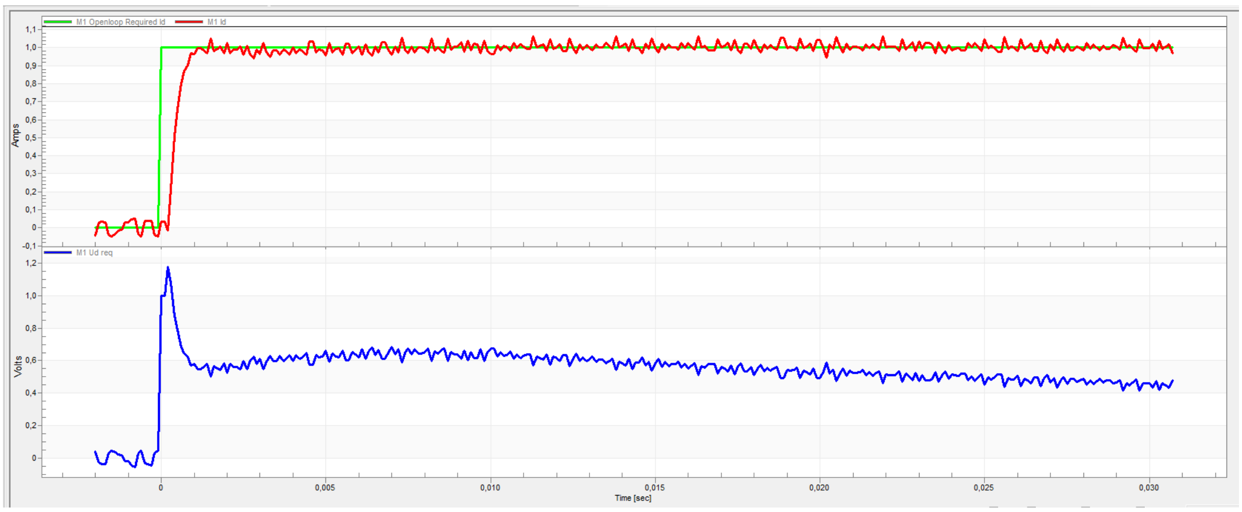


Figure 35. Optimal step response of the Id current controller

- The loop bandwidth is high (700 Hz) and the response time of the Id current is very fast, but with oscillations and overshoot (see Figure 3).

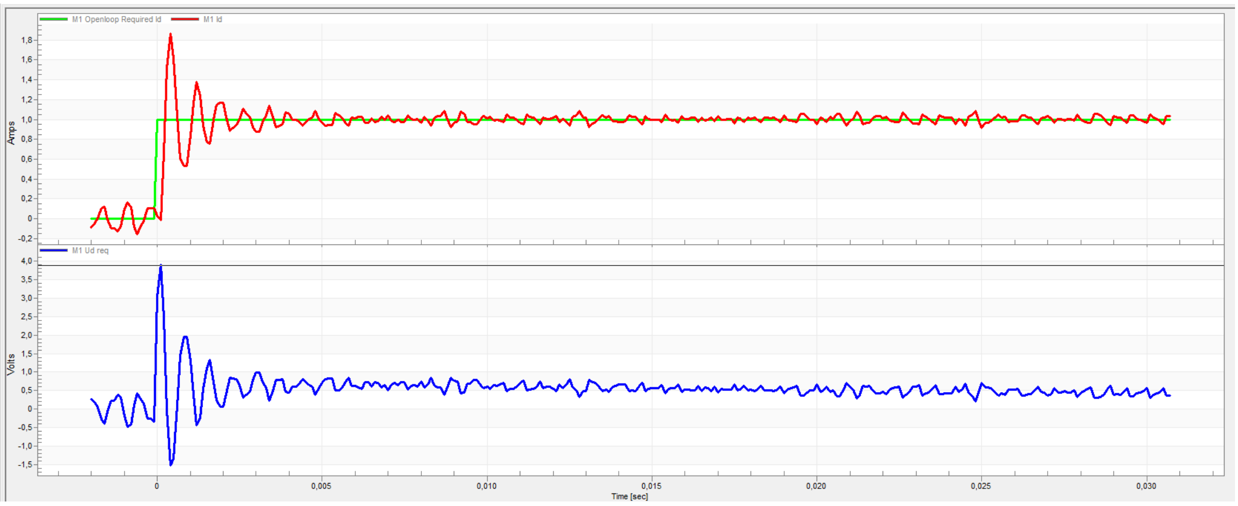


Figure 36. Fast step response of the Id current controller

7.10.4 Speed ramp tuning

To tune speed ramp parameters, follow the steps below:

1. The speed command is applied to the speed controller through a speed ramp. The ramp function contains two increments (up and down) which express the motor acceleration and deceleration per second. If the increments are very high, they can cause an overcurrent fault during acceleration and an overvoltage fault during deceleration. In the "Speed" scope, you can see whether the "Speed Actual Filtered" waveform shape equals the "Speed Ramp" profile.
2. The increments are common for the scalar and speed control. The increment fields are in the "Speed loop" tab and accessible in both tuning modes. Clicking the "Update target" button applies the changes to the MCU. An example speed profile is shown in Figure 37. The ramp increment down is set to 500 rpm/sec and the increment up is set to 3000 rpm/sec.
3. The startup ramp increment is in the "Sensorless" tab and its value is higher than the speed loop ramp.

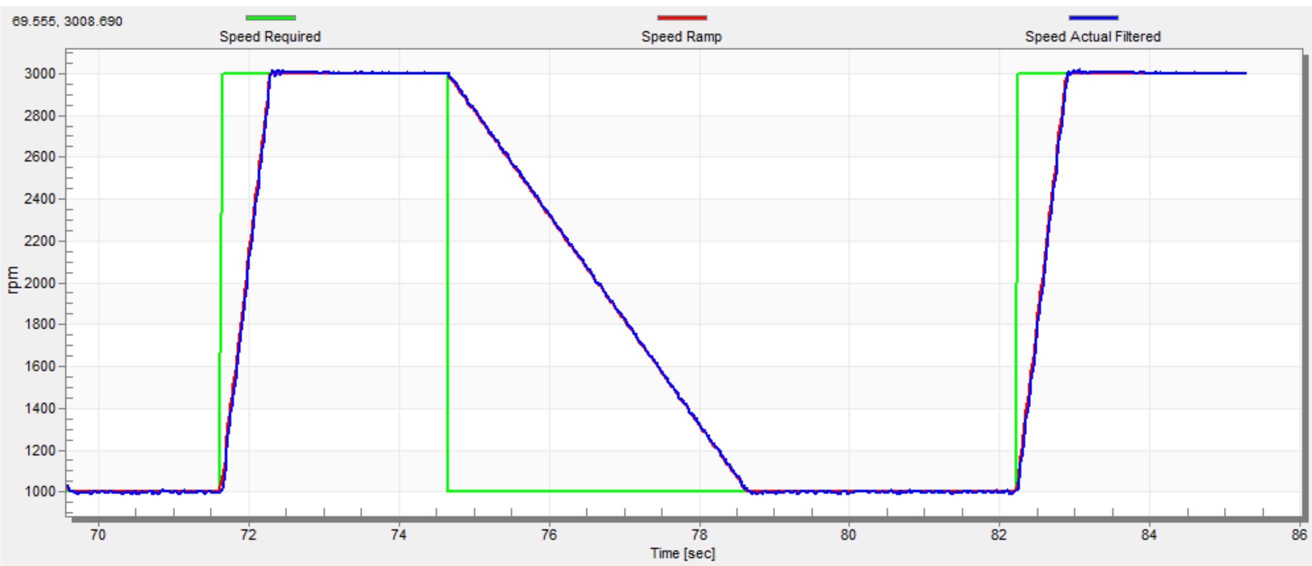


Figure 37. Speed profile

7.10.5 Open loop startup

To tune open loop startup parameters, follow the steps below:

1. The startup process can be tuned by a set of parameters located in the "Sensorless" tab. Two of them (ramp increment and current) are accessible in both tuning modes. The startup tuning can be processed in all control modes besides the scalar control. Setting the optimal values results in a proper motor startup. An example startup state of low-dynamic drives (fans, pumps) is shown in Figure 38.
2. Select the "Startup" recorder from the FreeMASTER project tree.
3. Set the startup ramp increment typically to a higher value than the speed-loop ramp increment.
4. Set the startup current according to the required startup torque. For drives such as fans or pumps, the startup torque is not very high and can be set to 15 % of the nominal current.
5. Set the required merging speed. When the open-loop and estimated position merging starts, the threshold is mostly set in the range of 5 % ~ 10 % of the nominal speed.
6. Set the merging coefficient—in the position merging process duration, 100 % corresponds to a one of an electrical revolution. The higher the value, the faster the merge. Values close to 1 % are set for the drives where a high startup torque and smooth transitions between the open loop and the closed loop are required.
7. To apply the changes to the MCU, click the "Update Target" button.
8. Select "SPEED_FOC" in the "M1 MCAT Control" variable.
9. Set the required speed higher than the merging speed.
10. Check the startup response in the recorder.
11. Tune the startup parameters until you achieve an optimal response.
12. If the rotor does not start running, increase the startup current.
13. If the merging process fails (the rotor is stuck or stopped), decrease the startup ramp increment, increase the merging speed, and set the merging coefficient to 5 %.

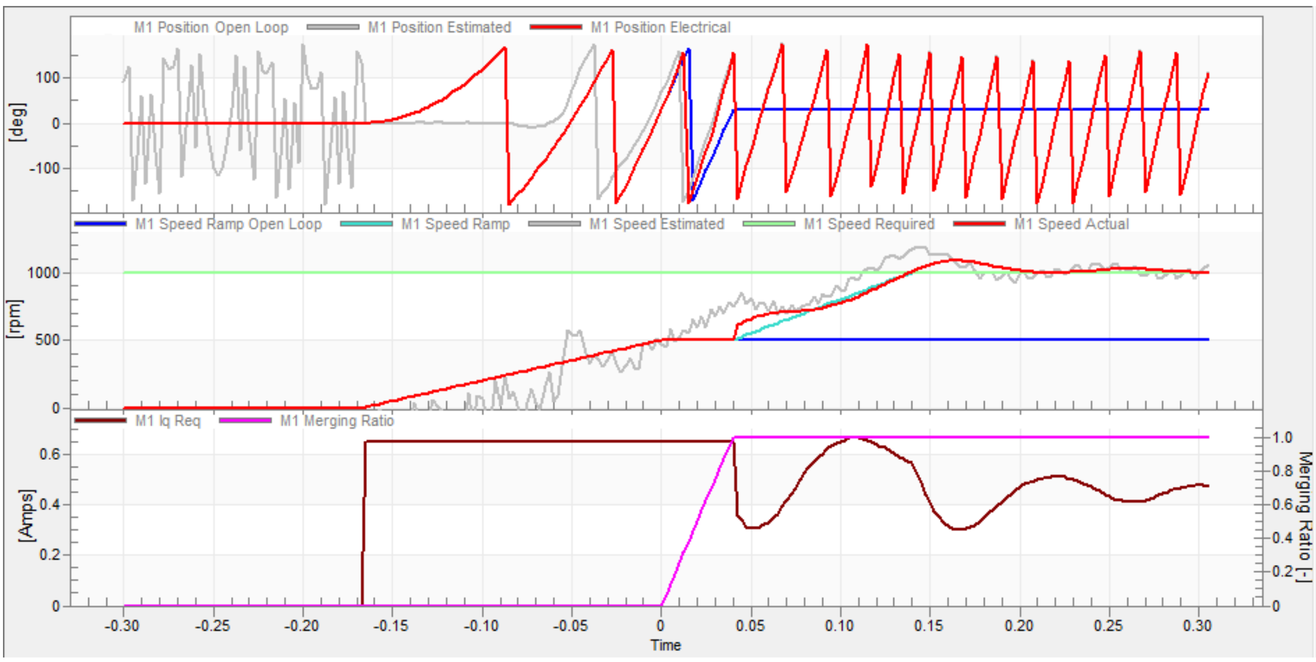


Figure 38. Motor startup

7.10.6 BEMF observer tuning

The bandwidth and attenuation parameters of the BEMF and tracking observer can be tuned. To tune the bandwidth and attenuation parameters, follow the steps below:

1. Navigate to the "Sensorless" MCAT tab.
2. Set the required bandwidth and attenuation of the BEMF observer. The bandwidth is typically set to a value close to the current loop bandwidth.
3. Set the required bandwidth and attenuation of the tracking observer. The bandwidth is typically set in the range of 10 – 20 Hz for most low-dynamic drives (fans, pumps).
4. To apply the changes to the MCU, click the "Update target" button.
5. Select the "Observer" recorder from the FreeMASTER project tree and check the observer response in the "Observer" recorder.

7.10.7 Speed PI controller tuning

The motor speed control loop is a first-order function with a mechanical time constant that depends on the motor inertia and friction. If the mechanical constant is available, the PI controller constants can be tuned using the loop bandwidth and attenuation. Otherwise, the manual tuning of the P and I portions of the speed controllers is available to obtain the required speed response (see Figure 39). There are dozens of approaches to tune the PI controller constants. To set and tune the speed PI controller for a PM synchronous motor, follow the steps below:

1. Select the "Speed Controller" option from the FreeMASTER project tree.
2. Select the "Speed loop" tab.
3. Check the "Manual Constant Tuning" option—that is, the "Bandwidth" and "Attenuation" fields are disabled and the "SL_Kp" and "SL_Ki" fields are enabled.
4. Tune the proportional gain:
   - Set the "SL_Ki" integral gain to 0.
   - Set the speed ramp to 1000 rpm/sec (or higher).
   - Run the motor at a convenient speed (about 30 % of the nominal speed).
   - Set a step in the required speed to 40 % of N_{nom}.
   - Adjust the proportional gain "SL_Kp" until the system responds to the required value properly and without any oscillations or excessive overshoot:
     - If the "SL_Kp" field is set low, the system response is slow.
     - If the "SL_Kp" field is set high, the system response is tighter.
     - When the "SL_Ki" field is 0, the system most probably does not achieve the required speed.
     - To apply the changes to the MCU, click the "Update Target" button.
5. Tune the integral gain:
   - Increase the "SL_Ki" field slowly to minimize the difference between the required and actual speeds to 0.
   - Adjust the "SL_Ki" field such that you do not see any oscillation or large overshoot of the actual speed value while the required speed step is applied.
   - To apply the changes to the MCU, click the "Update target" button.
6. Tune the loop bandwidth and attenuation until the required response is received. The example waveforms with the correct and incorrect settings of the speed loop parameters are shown in the following figures:
   - The "SL_Ki" value is low and the "Speed Actual Filtered" does not achieve the "Speed Ramp".

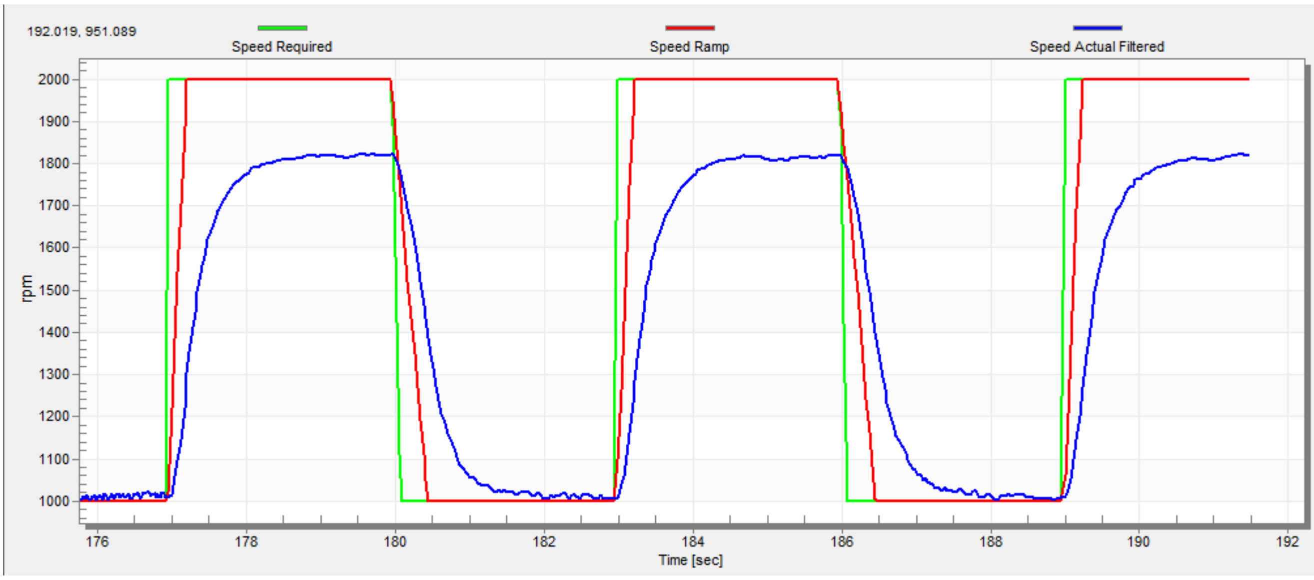


Figure 39. Speed controller response—SL_Ki value is low, Speed Ramp is not achieved

- The "SL_Kp" value is low, the "Speed Actual Filtered" greatly overshoots, and the long settling time is unwanted.

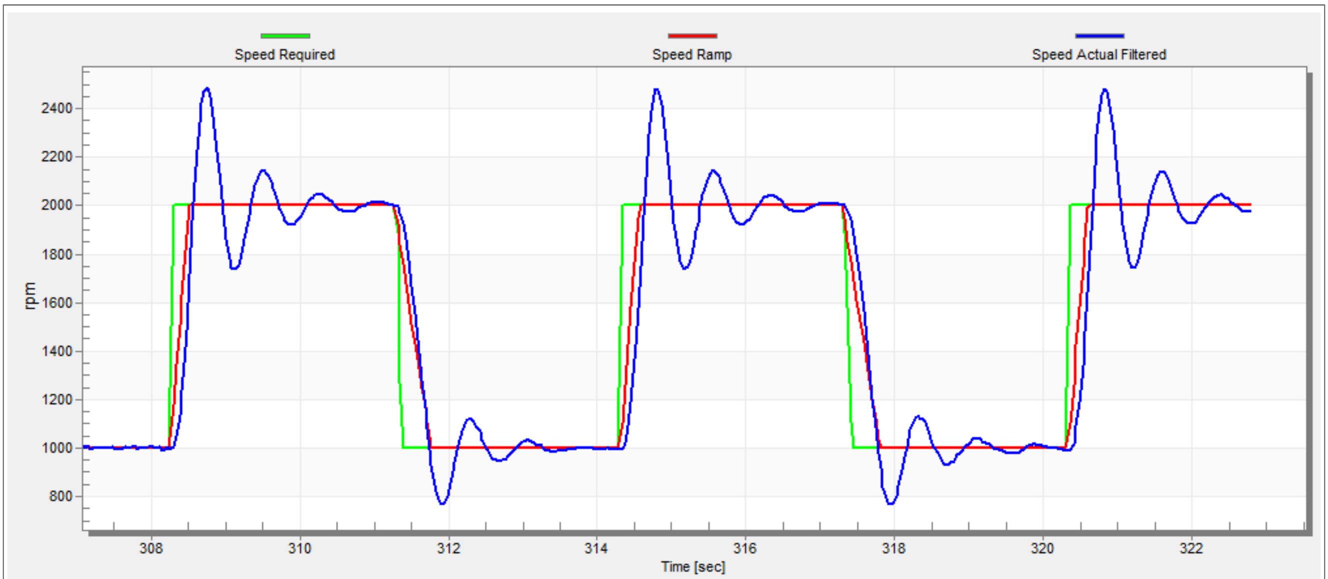


Figure 40. Speed controller response—SL_Kp value is low, Speed Actual Filtered greatly overshoots

- The speed loop response has a small overshoot and the "Speed Actual Filtered" settling time is sufficient. Such response can be considered optimal.

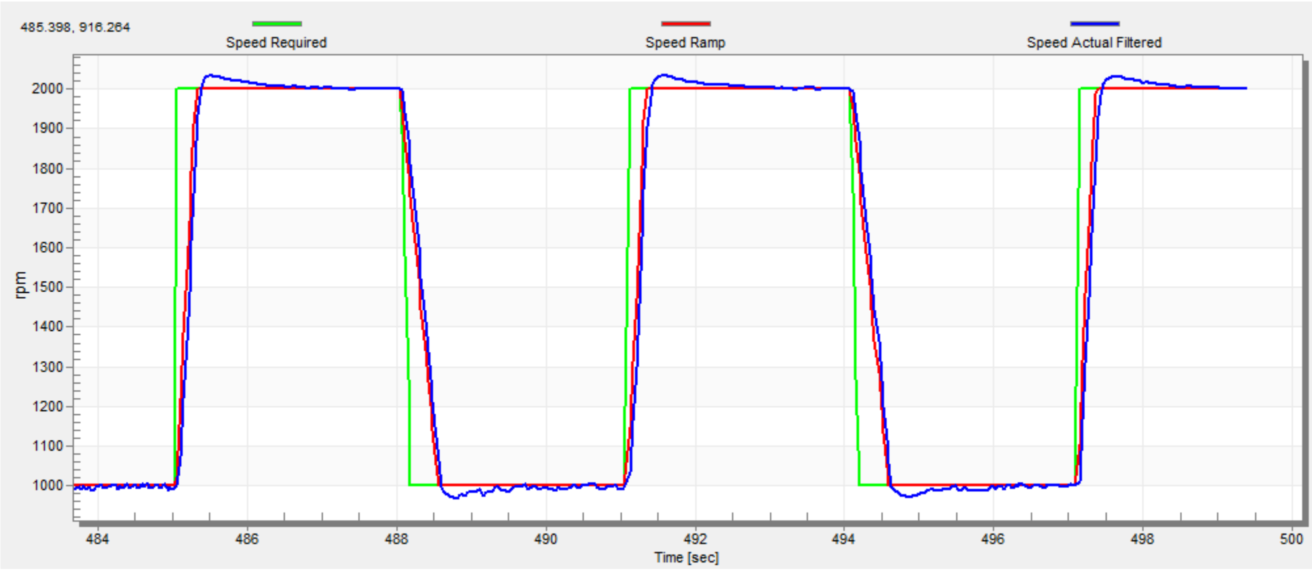


Figure 41. Speed controller response—speed loop response with a small overshoot

7.10.8 Position P controller tuning

The position control loop can be tuned using the proportional gain "M1 Position Loop Kp Gain" variable. A proportional controller can be used to unpretend the position-control systems. The key for the optimal position response is a proper value of the controller, which multiplies the error by the proportional gain (Kp) to get the controller output. The predefined base value can be manually changed. An encoder sensor must be used for a working position control. The following steps provide an example of how to set the position P controller for a PM synchronous motor:

1. Select the "Position Controller" scope in "Position Control" tab in the FreeMASTER project tree.
2. Tune the proportional gain in the position P controller constant:
   - Set a small value of "PL_Kp" (M1 Position Loop Kp Gain).
   - Select the position control, and set the required position in "M1 Position Required" variable (for example; 10 revolutions).
   - Select the "Position Controller" scope and watch the actual position response.
3. Repeat the previous steps until you achieve the required position response.

The "PL_Kp" value is low and the actual position response on the required position is very slow.

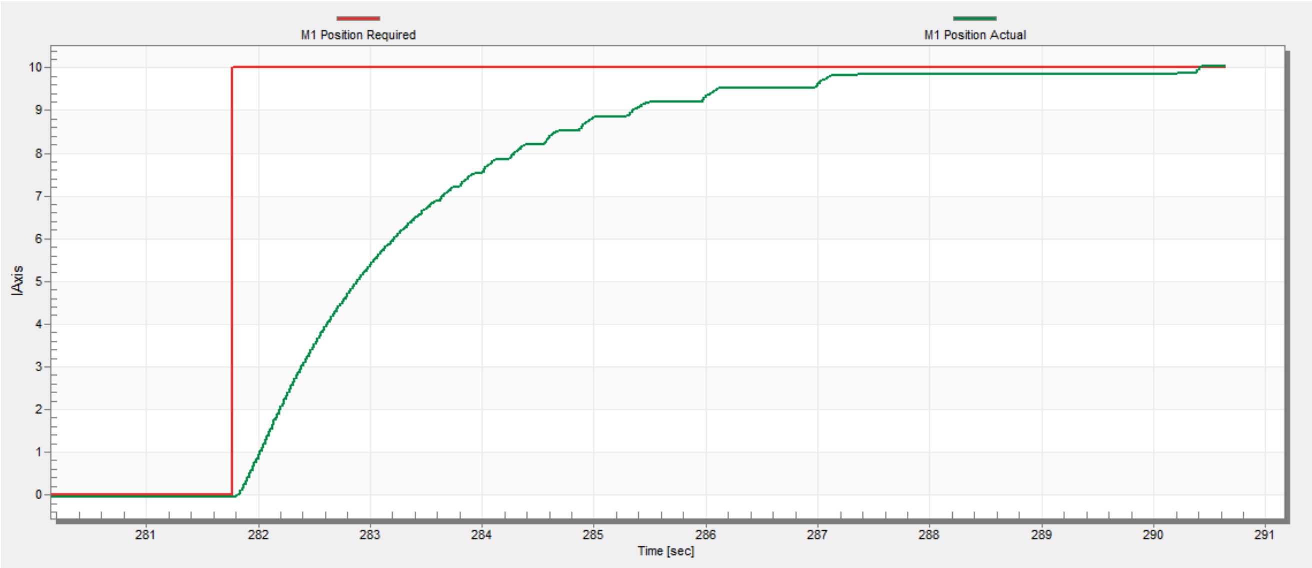


Figure 42. Position controller response—PL_Kp value is low, the actual position response is very slow

The "PL_Kp" value is too high and the actual position overshoots the required position.

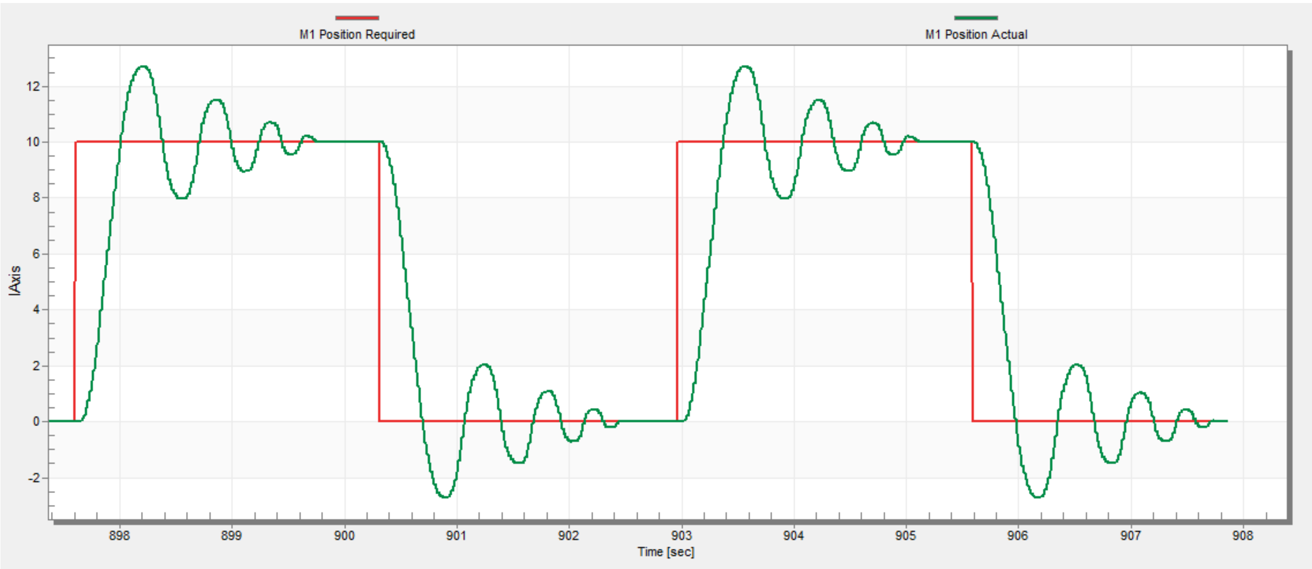


Figure 43. Position controller response—PL_Kp value is too high and the actual position overshoots

The "PL_Kp" value and the actual position response are optimal.

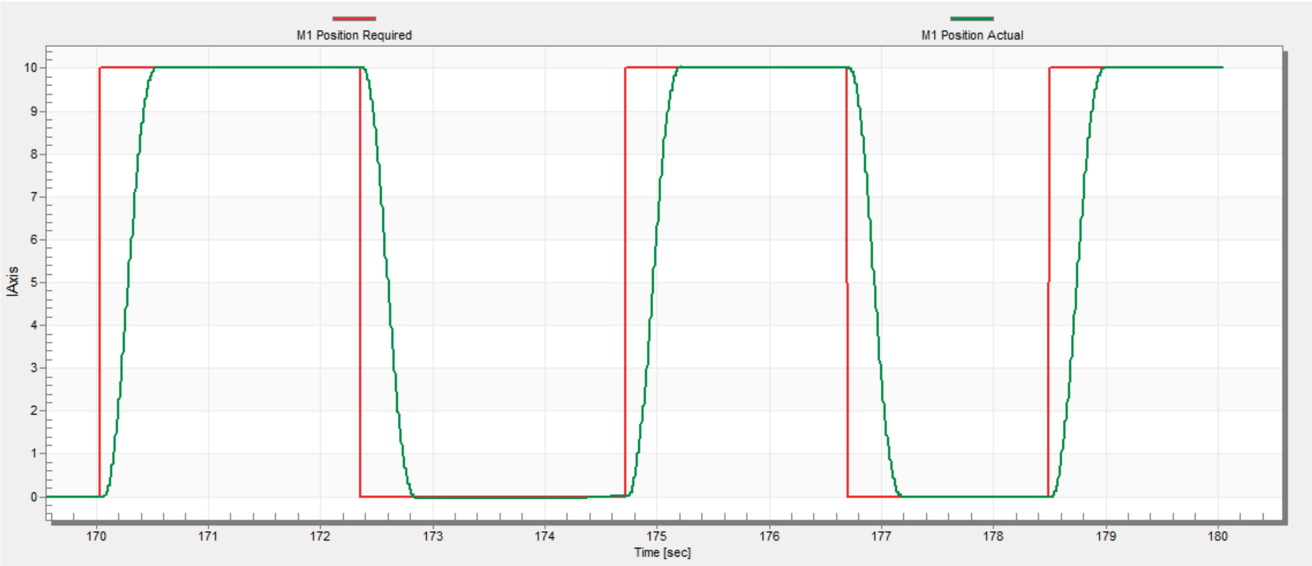


Figure 44. Position controller response—the actual position response is good

8 Conclusion

This application note describes the implementation of the sensor and sensorless field-oriented control of a 3-phase PMSM. The motor control software is implemented on NXP LPC55S36EVK board with the FRDM-MC-LVPMSM NXP Freedom development platform. The hardware-dependent part of the control software is described in Section 2. The motor-control application timing, and the peripheral initialization are described in Section 3. The motor user interface and remote control using FreeMASTER are described in Section 6. The motor parameters identification theory and the identification algorithms are described in Section 7.8.

9 Acronyms and abbreviations

Table 17 lists the acronyms and abbreviations used in this document.

Table 17. Acronyms and abbreviations

Acronym	Meaning
ADC	Analog-to-Digital Converter
ACIM	Asynchronous Induction Motor
ADC_ETC	ADC External Trigger Control
AN	Application Note
BLDC	Brushless DC motor
CCM	Clock Controller Module
CPU	Central Processing Unit
DC	Direct Current
DRM	Design Reference Manual
ENC	Encoder
FOC	Field-Oriented Control
GPIO	General-Purpose Input/Output
LPIT	Low-power Periodic Interrupt Timer
LPUART	Low-power Universal Asynchronous Receiver/Transmitter
MCAT	Motor Control Application Tuning tool
MCDRV	Motor Control Peripheral Drivers
MCU	Microcontroller
PDB	Programmable Delay Block
PI	Proportional Integral controller
PLL	Phase-Locked Loop
PMSM	Permanent Magnet Synchronous Machine
PWM	Pulse-Width Modulation
QD	Quadrature Decoder
TMR	Quad Timer
USB	Universal Serial Bus
XBAR	Inter-Peripheral Crossbar Switch
IOPAMP	Internal operational amplifier

10 References

These references are available on www.nxp.com:

- *Sensorless PMSM Field-Oriented Control* (document DRM148)
- *Motor Control Application Tuning (MCAT) Tool for 3-Phase PMSM* (document AN4642)

11 Useful links

- MCUXpresso SDK for Motor Control www.nxp.com/sdkmotorcontrol
- LPC553x/S3x: Advanced Analog Arm®Cortex®-M33-Based MCU Family
- MCUXpresso IDE - Importing MCUXpresso SDK
- MCUXpresso Config Tool
- MCUXpresso SDK Builder (SDK examples in several IDEs)
- Model-Based Design Toolbox (MBDT)

12 Revision history

Section 12 summarizes the changes done to the document since the initial release.

Table 18. Revision history

Revision number	Date	Substantive changes
0	11/2021	Initial release
1	01/2022	Add pmsm_enc_iopamp and pmsm_ enc_dual examples
2	07/2023	New documentation structure

13 Legal information

13.1 Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

13.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

13.3 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Tables

Figures

Contents

Date of release: 31 July 2023
Document identifier: PMSMLPC55S36EVK