Document identifier: PMSMLPC55S36EVK
Rev. 0, 11/2021

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

Contents

Chapter 1 Introduction.. 3

Chapter 2 Hardware setup..4

Chapter 3 LPC5500 series features and peripheral settings.............................. 10

Chapter 4 Project file and IDE workspace structure.. 14

Chapter 5 Tools.. 16

Chapter 6 Motor-control peripheral initialization.. 17

Chapter 7 User interface..19

Chapter 8 Remote control using FreeMASTER...20

Chapter 9 Conclusion.. 40

Chapter 10 Acronyms and abbreviations...41

Chapter 11 References..42

Chapter 12 Useful links..43

Chapter 13 Revision history..44

Chapter 1
Introduction

This user's guide describes the implementation of the sensor and sensorless motor-control software for a 3-phase Permanent Magnet Synchronous Motor (PMSM). The software is intended for PMSM with sinusoidal Back Electromotive Force (back-EMF) but is also very well usable for brushless motors (BLDC) with trapezoidal back-EMF.

The software also includes the motor parameters identification algorithm, on NXP 32-bit LPC series MCUs. The sensorless control software itself and the PMSM control theory, in general, are described in DRM148: Sensorless PMSM Field-Oriented Control.

The Freedom power stage (FRDM-MC-LVPMSM) is used as hardware platform for the PMSM control reference solution.

The hardware-dependent part of the sensorless control software, including a detailed peripheral setup and the Motor Control (MC) peripheral drivers, is described as well.

The motor parameters identification theory and algorithms are described in this document.

The last part of the document introduces and explains the user interface represented by the Motor Control Application Tuning (MCAT) page based on the FreeMASTER run-time debugging tool. These tools present a simple and user-friendly way for motor parameter identification, algorithm tuning, software control, debugging, and diagnostics.

Table 1. Supported devices and control methodes

		Possible control methods in SDK example					
Device	**Default motor**	Scalar	Voltage	Current FOC (Torque)	Sensorless Speed FOC	Sensored Speed FOC	Sensored Position FOC (Servo)
LPC55S36-EVK	Teknic M-2310P motor (with ENC)	✓	✓	✓	✓	✓	✓

Chapter 2
Hardware setup

The PMSM Field-Oriented Control (FOC) application runs on the FRDM-MC-LVPMSM development platform with the LPC55S36-EVK development tool, in combination with the Teknic M-2310P or Linix 45ZWN24-40 permanent magnet synchronous motors.

2.1 FRDM-MC-LVPMSM

This evaluation board, in a shield form factor, effectively turns an NXP Freedom development board or an evaluation board into a complete motor-control reference design, 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 the power supply input voltage of 24-48 VDC with a 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.

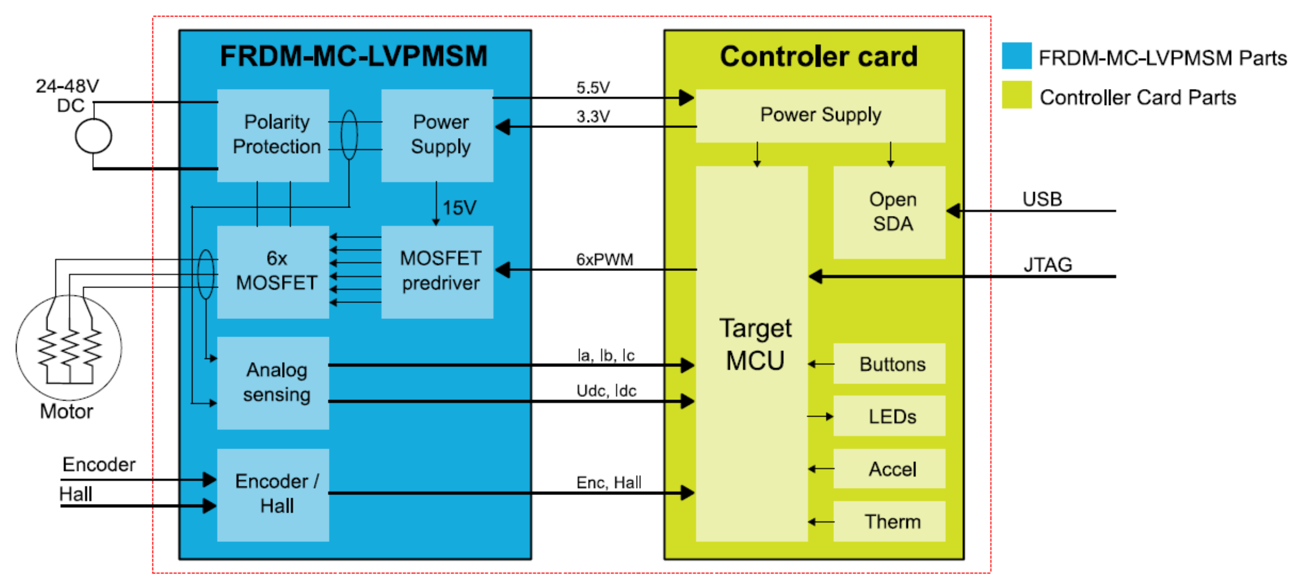


Figure 1. Motor-control development platform block diagram

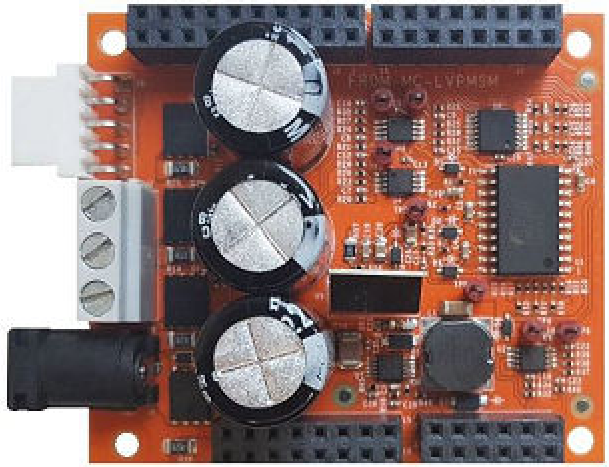


Figure 2. FRDM-MC-LVPMSM

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

2.2 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 3. 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.

2.3 Teknic M-2310P motor

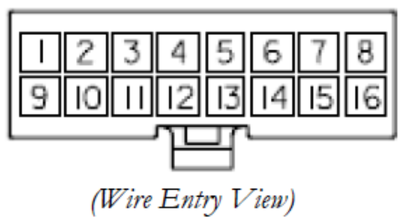
The Teknic M-2310P-LN-04K motor is a low-voltage 3-phase permanent-magnet motor used in PMSM applications. The motor has two feedback sensors (hall and encoder). For information on the wiring of feedback sensors, see the datasheet on the manufacturer web page. The motor parameters are listed in Table 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
Rated power	P	170	W
Continuous current	Ics	7.1	A
Number of pole-pairs	pp	4	-

Figure 4. Teknic M-2310P permanent magnet synchronous motor

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

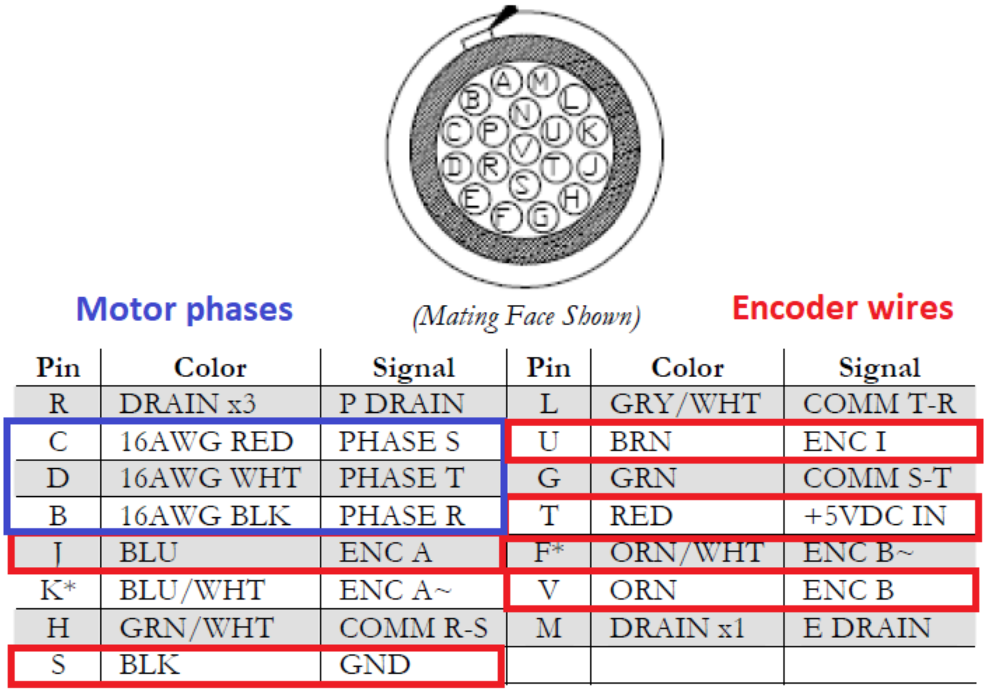


Motor phases

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 I
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 5. Teknic motor connector type 1



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 6. Teknic motor connector type 2

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 jumper and resistor settings according to Table 4 for the motor-control application to work properly on Motor 2 connector.

Table 4. LPC55S36-EVK jumper and resistor settings

Jumper	Setting	Jumper	Setting	Resistor	Setting
JP41	2-3	JP50	2-3	R495	2-3
JP42	2-3	JP51	2-3	R496	2-3
JP44	2-3	JP52	2-3	R497	2-3
JP45	2-3	JP53	2-3	R499	2-3
JP46	2-3	-	-	-	-

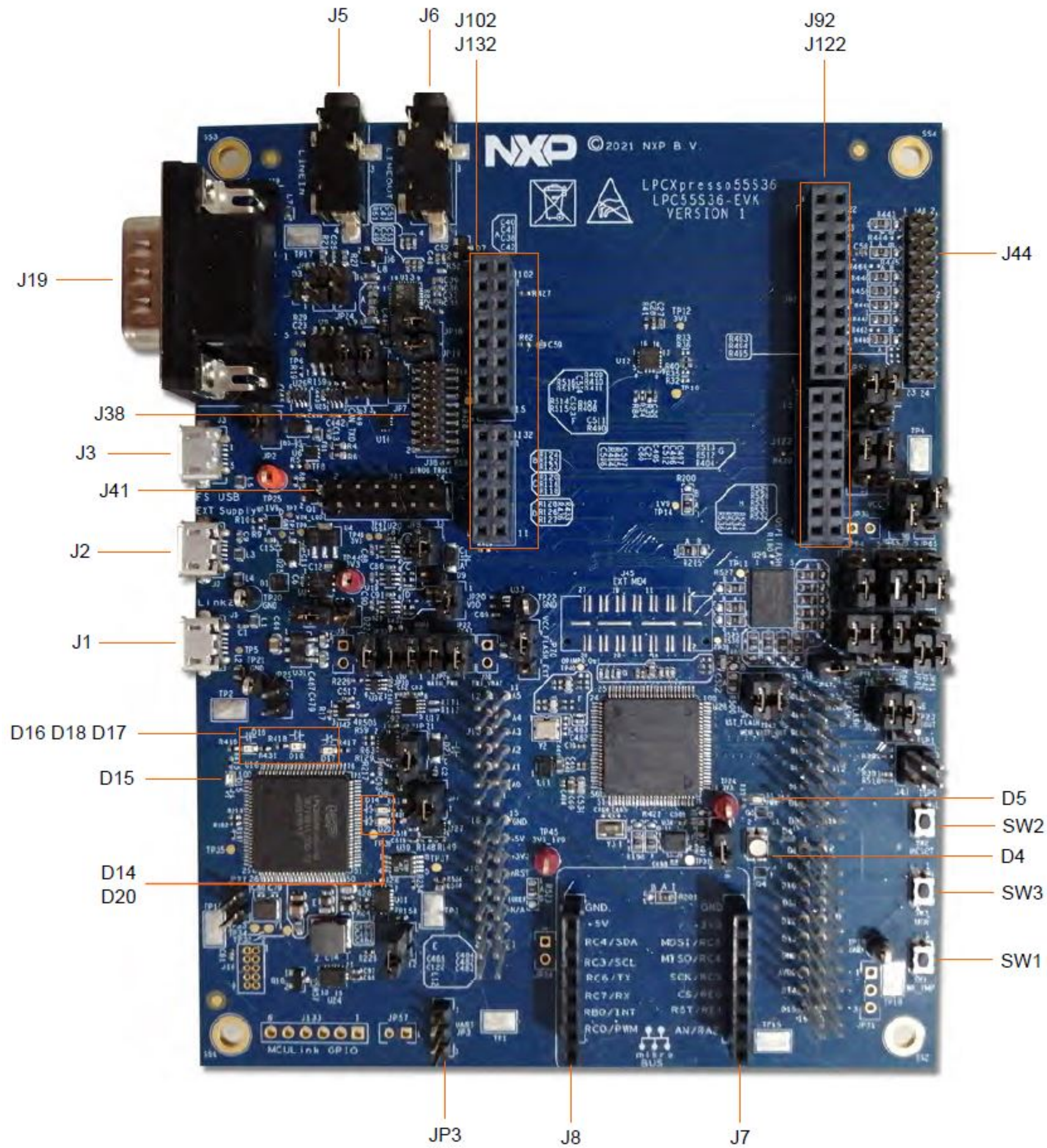


Figure 7. LPC55S36-EVK board with highlighted jumper settings

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.

Chapter 3
LPC5500 series features and peripheral settings

This section describes the peripheral settings and application timing. The LPC5500 MCU series contains Arm's newest Cortex-M33 technology. It combines significant product architecture enhancements and greater integration over previous generations with dramatic power consumption improvements and advanced security features, including the SRAM PUF-based root of trust and provisioning, real-time execution from encrypted images (internal flash), and asset protection with Arm TrustZone-M. In addition, the LPC5500 MCU series features seven scalable families with broad package and memory options, as well as the comprehensive MCUXpresso software and tools ecosystem and low-cost development boards.

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.

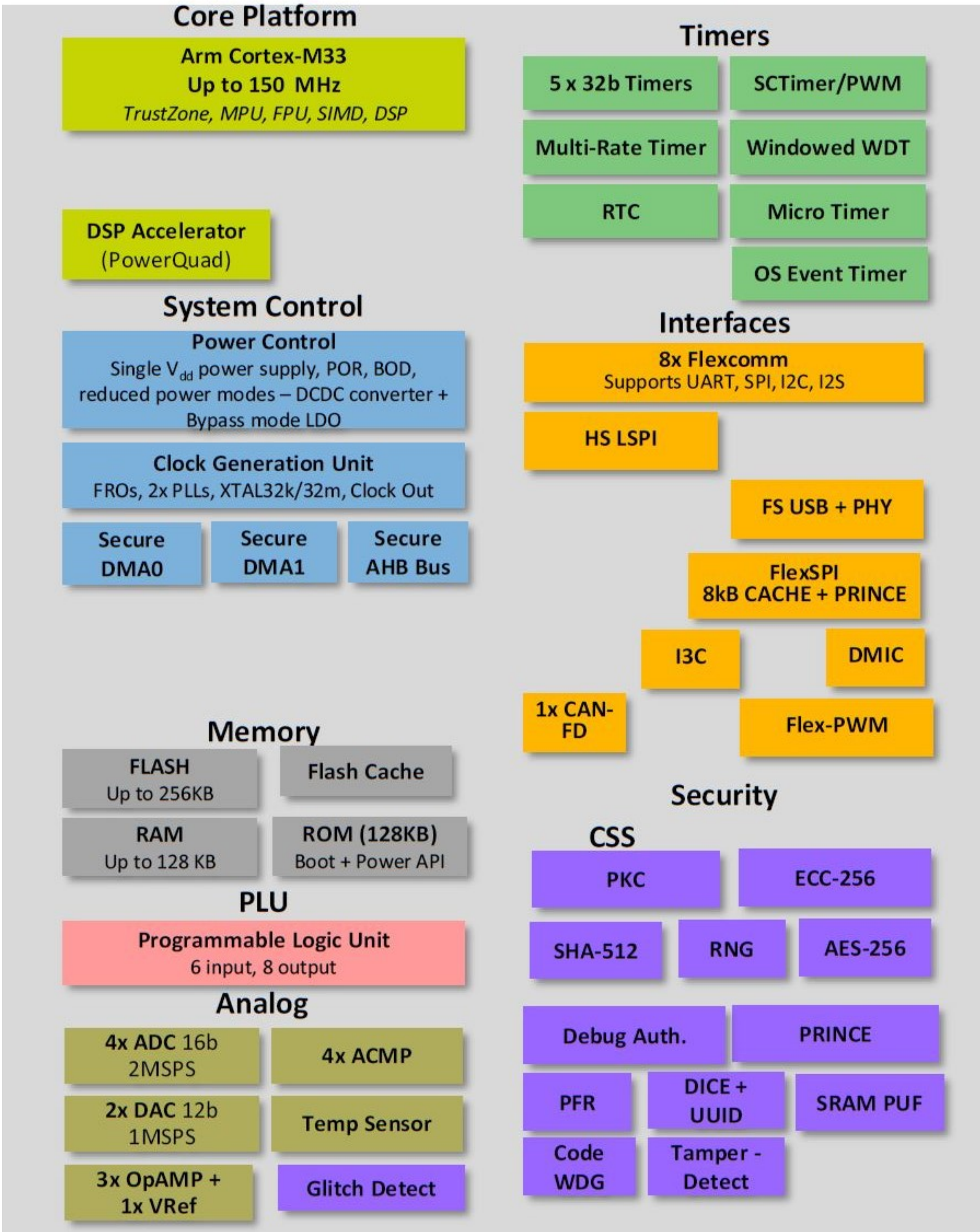


Figure 8. LPC55S3x block diagram

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.

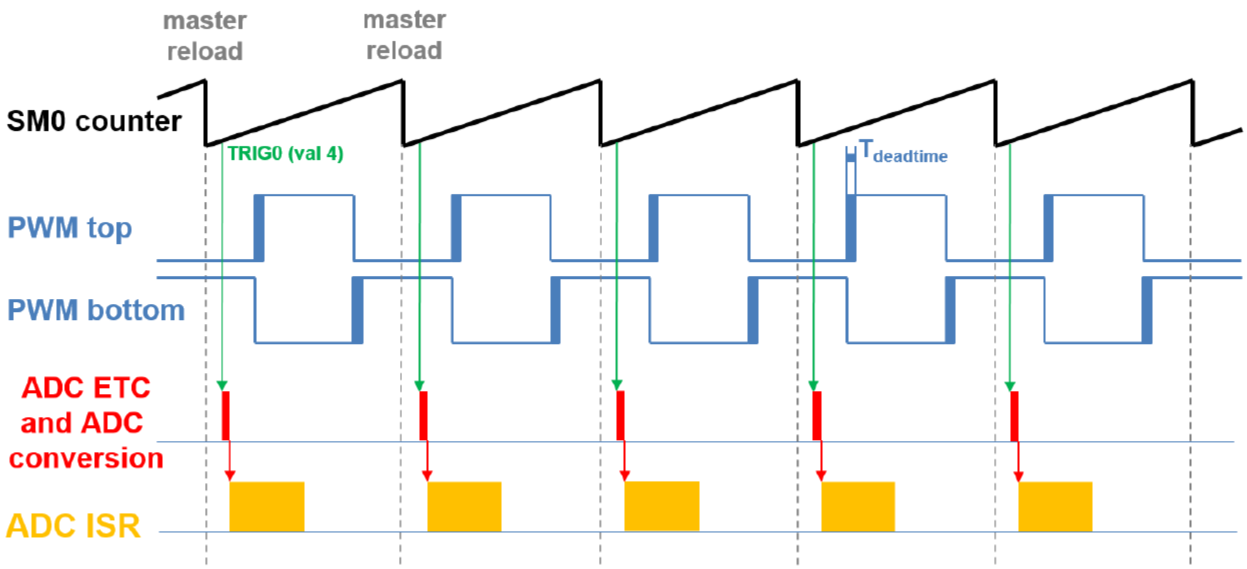


Figure 9. Hardware timing and synchronization on LPC55S36

- 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 TRIG4 value) for the ADC_ETC (ADC External Trigger Control) with a delay of approximately $T_{deatime}/2$. This delay ensures correct current sampling at the duty cycles close to 100 %.
- ADC_ETC 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 apply to the application built using the MCUXpresso IDE in the Debug and Release configurations. Table 5 and Table 6 show the memory usage and CPU load. The memory usage is calculated from the *.map* linker file, including the 4-KB FreeMASTER recorder buffer allocated in RAM. 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. In this case, it applies to the fast-loop frequency of 10 KHz and the slow-loop frequency of 1 KHz. The total CPU load is calculated using these equations:

$$CPU_{fast} = cycles_{fast} \frac{f_{fast}}{f_{CPU}} 100\ [\%]$$

$$CPU_{slow} = cycles_{slow} \frac{f_{slow}}{f_{CPU}} 100\ [\%]$$

$$CPU_{total} = CPU_{fast} + CPU_{slow}\ \ [\%]$$

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 (10 KHz).

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 (1 KHz).

CPU_{total} - the total CPU load consumed by the motor control.

Table 5. LPC-55S36 memory usage

	Debug configuration		Release configuration	
Program flash	93 428 B	Usage 37.09%	53 964 B	Usage 21.42%
SRAM	16 428 B	Usage 14.32%	16 384 B	Usage 14.29%

Table 6. LPC-55S36 CPU load

	Fast-loop	Slow-loop
Maximum CPU load	46.1 %	34.62 %

NOTE
Memory usage and maximum CPU load can differ depending on the used IDEs and settings.

Chapter 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 in a logical manner. The folder structure used in the IDE is different from the structure of the PMSM package installation, but it uses the same files. The different organization is chosen due to a better manipulation with folders and files in workplaces and due to the possibility to add or remove files and directories. The "*pack_motor_board*" project includes the available functions and routines, MID functions, scalar and vector control of the motor, FOC control, and FreeMASTER MCAT project. This project serves for development and testing purposes.

4.1 PMSM project structure

The directory tree of the PMSM project is shown in Figure 10.

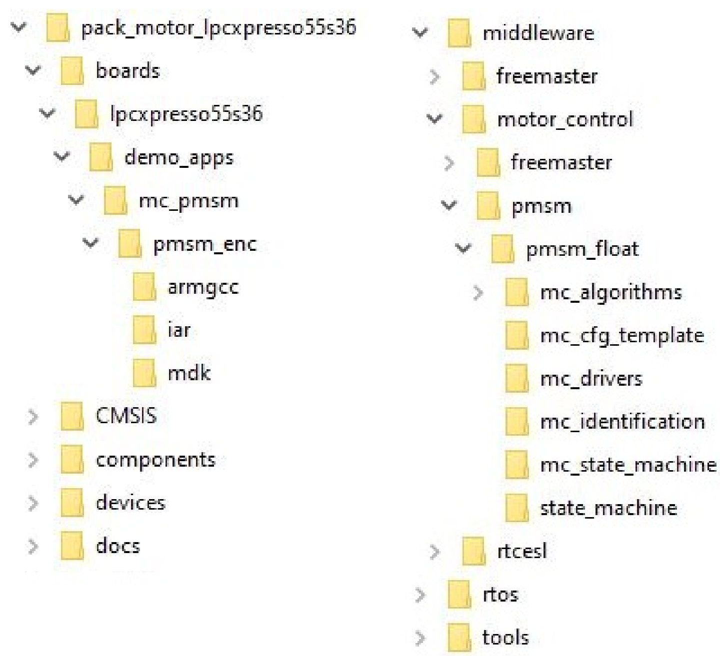


Figure 10. Directory tree

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 file *pmsm_float_enc.pmp*. Open this file 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.

Chapter 5 Tools

Install the FreeMASTER Run-Time Debugging Tool 3.1.2 and one of the following IDEs on your PC to run and control the PMSM application properly:

- IAR Embedded Workbench IDE v9.10.2 or higher
- MCUXpresso v11.4.0
- ARM-MDK - Keil µVision version 5.34

For pin_mux.c, clock_config.c or peripherals.c modifications is recommended use MCUXpresso Configuration Tool v11 or higher.

NOTE

For information on how to build and run the application in your IDE, see the *Getting Started with MCUXpresso SDK* document located in the *pack_motor_<booard>/docs* folder or find the related documentation at MCUXpresso SDK builder.

5.1 Compiler warnings

Warnings are diagnostic messages that report constructions that are not inherently erroneous and warn about potential runtime, logic, and performance errors. In some cases, warnings can be suspended and these warnings do not show during the compiling process. One of such special cases is the “unused function” warning, where the function is implemented in the source code with its body, but this function is not used. This case occurs when you implement the function as a supporting function for better usability, but you do not use the function for any special purposes for a while.

The IAR Embedded Workbench IDE suppresses these warnings:

- Pa082 - undefined behavior; the order of volatile accesses is not defined in this statement.
- Pa050 - non-native end of line sequence detected.

The Arm-MDK Keil µVision IDE suppresses these warnings:

- 6314 - No section matches pattern xxx.o (yy).

By default, there are no other warnings shown during the compiling process.

Chapter 6
Motor-control peripheral initialization

The motor-control peripherals are initialized by calling the *MCDRV_Init_M1()* function during the 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).

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.

The *mc_periph_init.h* header file provides several macros that can be defined by the user:

- *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_SLOW_LOOP_FREQ*—the value of this definition sets the speed-loop frequency.
- *M1_PWM_DEADTIME*—the value of the PWM dead time in nanoseconds.

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

- The available APIs for the ADC are:
  - *mcdrv_adc_t*—MCDRV ADC structure data type.
  - *void InitADCx()*—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. The function always returns true. Not available for all devices.
  - *void M1_MCDRV_CURR_3PH_CALIB_INIT(mcdrv_adc_t*)*—this function initializes the phase-current channel-offset measurement. This function always returns true.
  - *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. This function always returns true.
  - *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. This function always returns true.
  - *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. This function always returns true.
- The available APIs for the quadrature encoder are:
  - *mcdrv_qd_enc_t*—MCDRV QD structure data type.
  - *bool_t InitQDx()*—this function is by default called during the QD periphery initialization procedure invoked by the MCDRV_Init_M1() function.
  - *bool_t M1_MCDRV_QD_GET(mcdrv_qd_enc_t*)*—this function returns the actual position and speed. This functionalways returns true.
  - *bool_t M1_MCDRV_QD_SET_DIRECTION(mcdrv_qd_enc_t*)*—this function sets the direction of the quadrature encoder. This function always returns true.

    — *bool_t M1_MCDRV_QD_CLEAR(mcdrv_qd_enc_t*)*—this function clears the internal variables and decoder counter. This function always returns true.

- The available APIs for the PWM are:
    — *mcdrv_eflexpwm_t*—MCDRV PWM structure data type.
    — *void InitPWMx()*—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. This function always returns true.
    — *void M1_MCDRV_PWM3PH_EN(mcdrv_pwma_pwm3ph_t*)*—calling this function enables all PWM channels. This function always returns true.
    — *void M1_MCDRV_PWM3PH_DIS (mcdrv_pwma_pwm3ph_t*)*—calling this function disables all PWM channels. This function always returns true.

Chapter 7
User interface

The application contains the demo mode to demonstrate motor rotation. You can operate it using FreeMASTER. 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. Data is transferred between the PC and the embedded application via the serial interface. This interface is provided by the debugger included in the boards.

The application can be controlled using these two interfaces:

- The button on the EVK development board (controlling the demo mode):
  - LPC55S36-EVK - SW3
- Remote control using FreeMASTER (chapter Remote control using FreeMASTER):
  - Using the Motor Control Application Tuning (MCAT) interface.
  - Setting a variable in the FreeMASTER Variable Watch.

If you are using your own motor (different from the default motors), make sure to identify all motor parameters. The automated parameter identification is described in the following sections.

Chapter 8
Remote control using FreeMASTER

This section provides information about the tools and recommended procedures to control the 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 3.1.2 at www.nxp.com/freemaster. To run the FreeMASTER application including the MCAT tool, double-click the *pmsm_float_enc.pmp* (or *pmsm_float.pmp*) file located in the *pack_motor_lpcxx\middleware\motor_control\freemaster* folder. The FreeMASTER application starts and the environment is created automatically, as defined in the **.pmp* file.

NOTE

In MCUXpresso can be FreeMASTER application run directly from IDE in *motor_control/freemaster* folder

8.1 Establishing FreeMASTER communication

The remote operation is provided by FreeMASTER via the USB interface. Perform the following steps to control a PMSM motor using FreeMASTER:

1. Download the project from your chosen IDE to the MCU and run it.
2. Open the FreeMASTER file *pmsm_x.pmp*. The PMSM project uses the TSA by default, so it is not necessary to select a symbol file for FreeMASTER.
3. Click the communication button (the red “STOP” button in the top left-hand corner) to establish the communication.

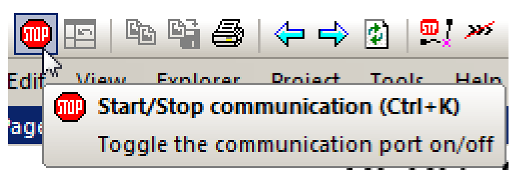


Figure 11. Red “STOP” button placed in top left-hand corner

4. If the communication is established successfully, the FreeMASTER communication status in the bottom right-hand corner changes from “Not connected” to “RS232 UART Communication; COMxx; speed=19200”. Otherwise, the FreeMASTER warning popup window appears.

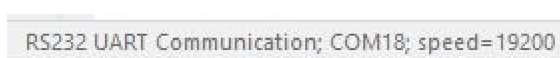


Figure 12. FreeMASTER—communication is established successfully

5. Press *F5* to reload the MCAT HTML page and check the App ID.
6. Control the PMSM motor using the MCAT “Control structure” tab, the MCAT “Application demo control” tab, or by directly writing to a 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 19200 bps.

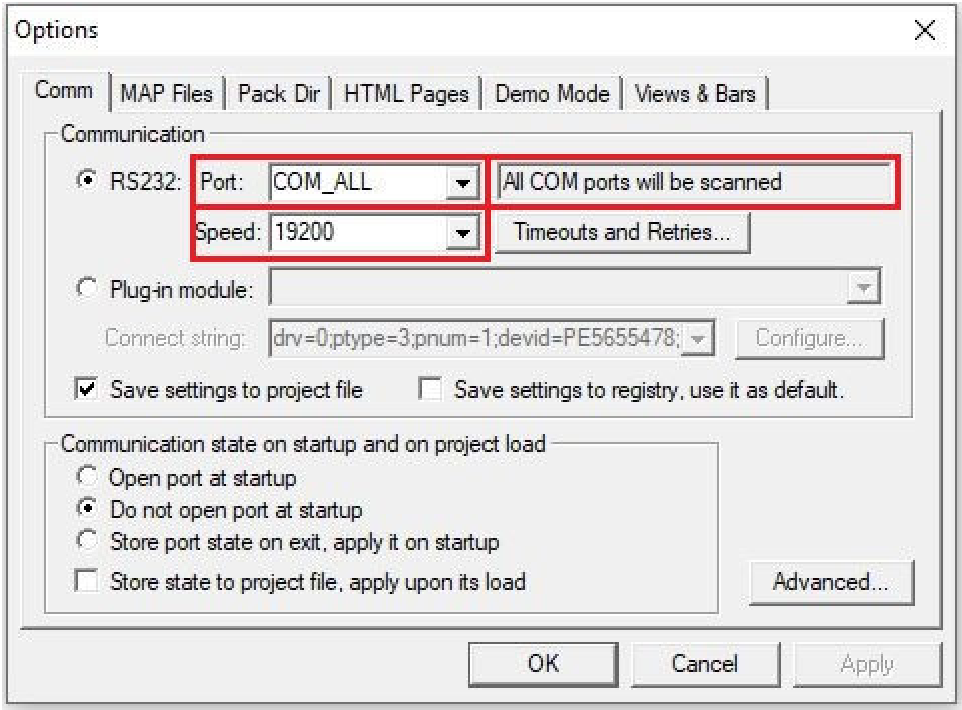


Figure 13. FreeMASTER communication setup window

2. If “OpenSDA-CDC Serial Port” is not printed out in the message box next to the “Port” drop-down menu, unplug and then plug in the USB cable and reopen the FreeMASTER project.

Make sure to supply your development board from a sufficient energy source. Sometimes the PC USB port is not sufficient to supply the development board.

8.2 MCAT FreeMASTER interface (Motor Control Application Tuning)

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 modular page, which runs within FreeMASTER. The tool consists of the tab menu, tuning mode selector, and workspace shown in Figure 14. Each tab from the tab menu represents one sub-module which enables you to tune or control different aspects of the application. Besides the MCAT page for PMSM, several scopes, recorders, and variables in the project tree are predefined in the FreeMASTER project file to further simplify the motor parameter tuning and debugging. When the FreeMASTER is not connected to the target, the “Board found” line shows “Board ID not found”. When the communication with the target MCU is established using a correct software, the “Board found” line displays the same board name as "Board ID" variable watch and all stored parameters for the given MCU are loaded. If the connection is established and the board ID is not shown, press *F5* to reload the MCAT HTML page.

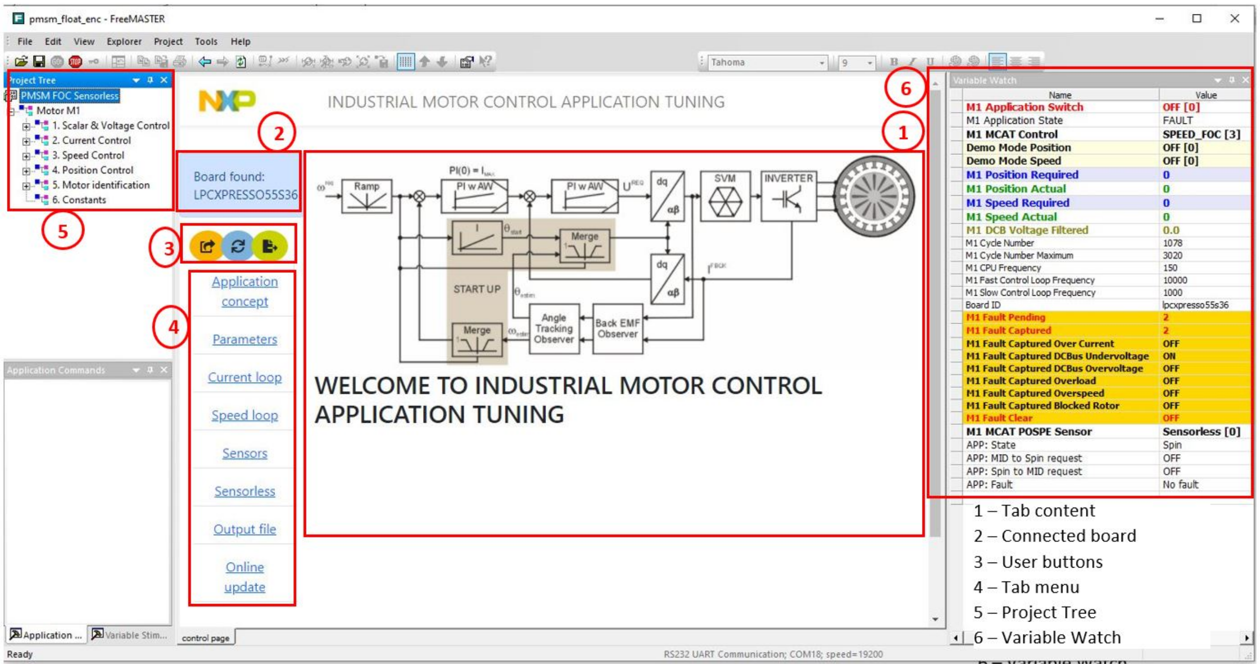


Figure 14. MCAT layout

In the default configuration, the following tabs are available:

- "Application concept"—welcome page with the PMSM sensor/sensorless FOC diagram and a short description of the application.
- "Parameters"—this page enables you to modify the motor parameters, specification of hardware and application scales, 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. Not available for all devices.
- "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.

Most tabs offer the possibility to immediately load the parameters specified in the MCAT into the target using the "Update target" button and save (or restore) them from the hard drive file using the "Save data" and "Reload data" buttons.

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

Control structure

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

The scalar control diagram is shown in Figure 15. 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. Hence, the control method is sometimes called Volt per Hertz (or V/Hz). The position estimation BEMF observer and tracking observer algorithms (see Sensorless PMSM Field-Oriented Control (document DRM148) for more information) run in the background, even if the estimated position information is not directly used. This is useful for the BEMF observer tuning.

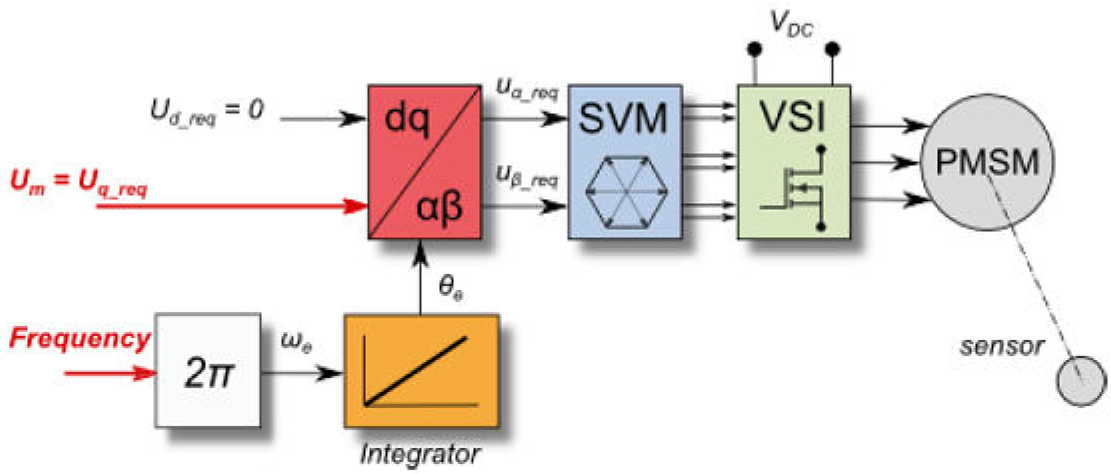


Figure 15. Scalar control mode

The block diagram of the voltage FOC is in Figure 16. 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.

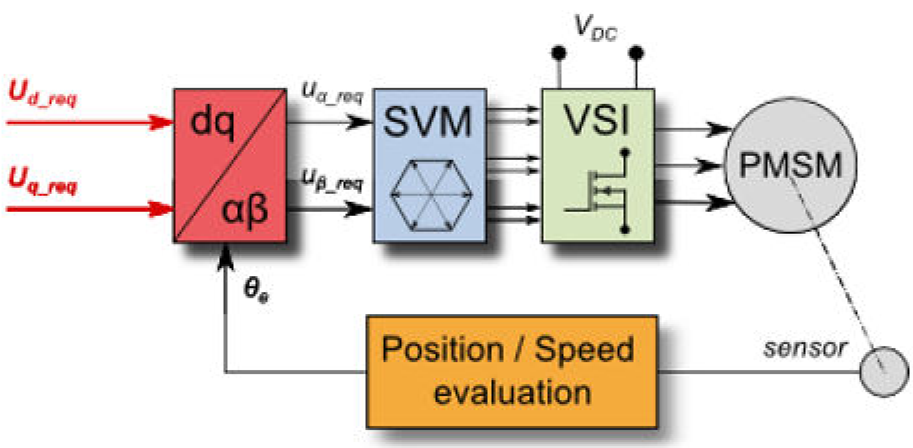


Figure 16. Voltage FOC control mode

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 the block diagram in Figure 17. The d-axis current component "M1 MCAT Id Required" is responsible for the rotor flux control. 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 that the BEMF observer is tuned correctly, the current PI controllers can be tuned using the current FOC control structure.

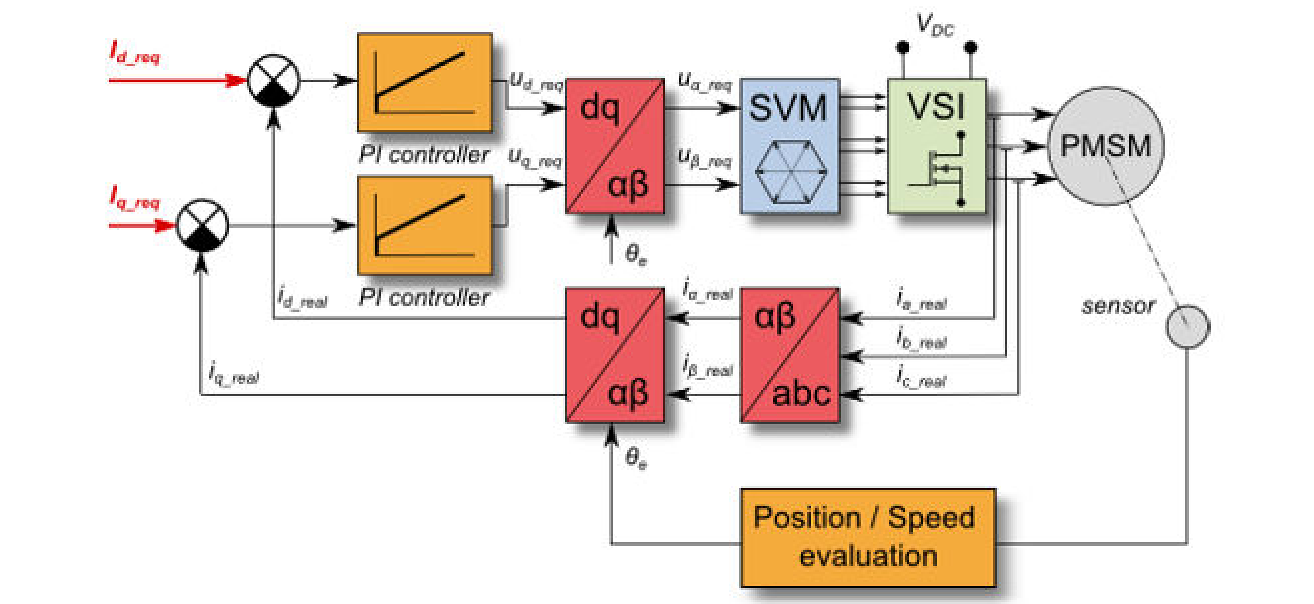


Figure 17. Current (torque) control mode

The speed PMSM sensor/sensorless FOC (its diagram is shown in Figure 18) 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.

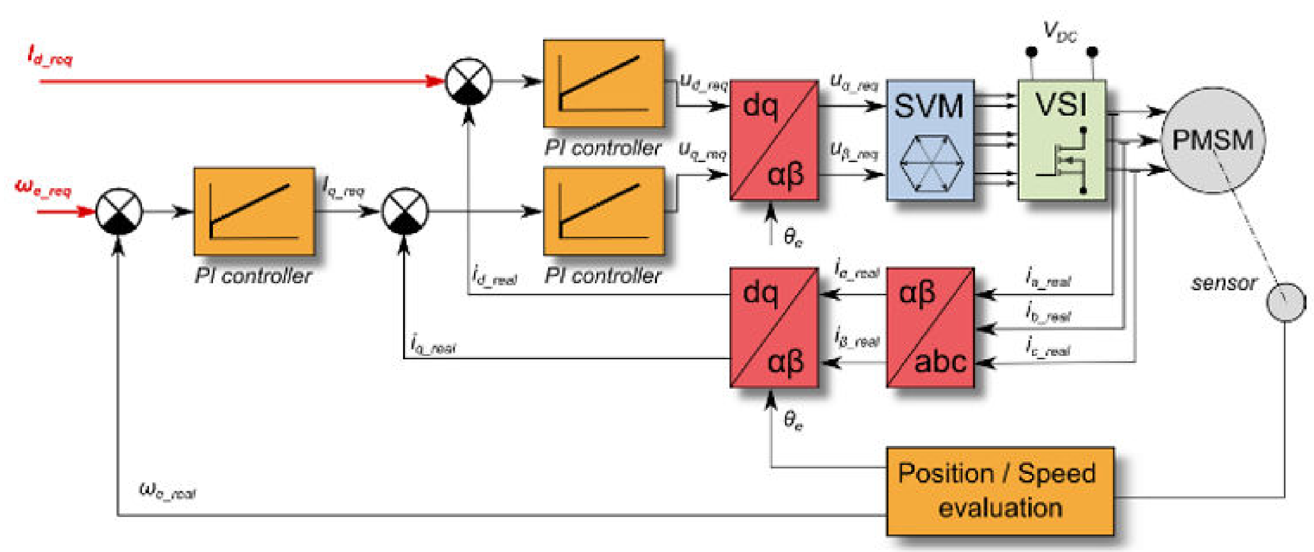


Figure 18. Speed FOC control mode

The position PMSM sensor FOC is shown in Figure 19 (Not available for all devices). 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 needed 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 the overvoltage fault.

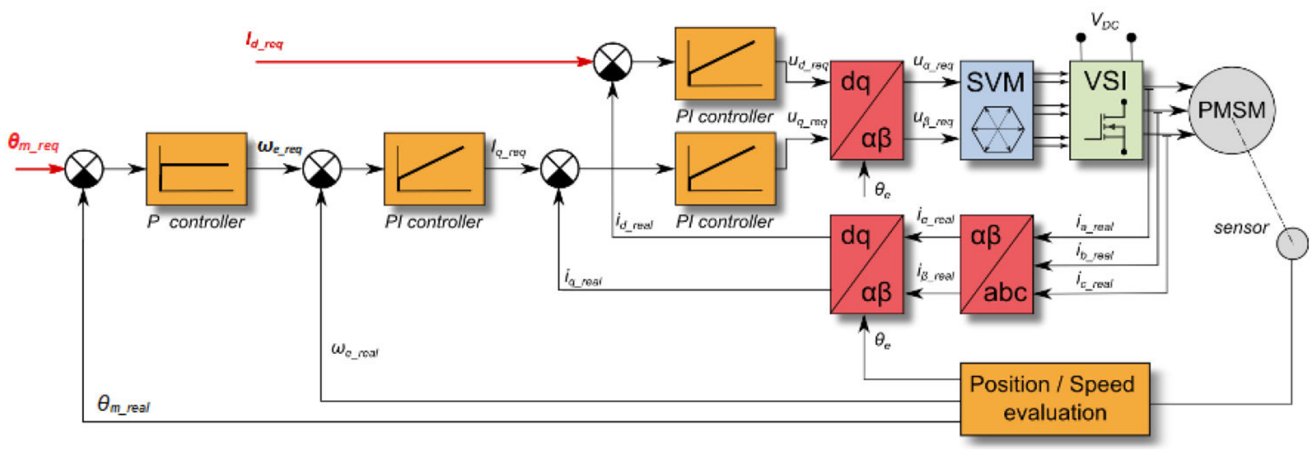


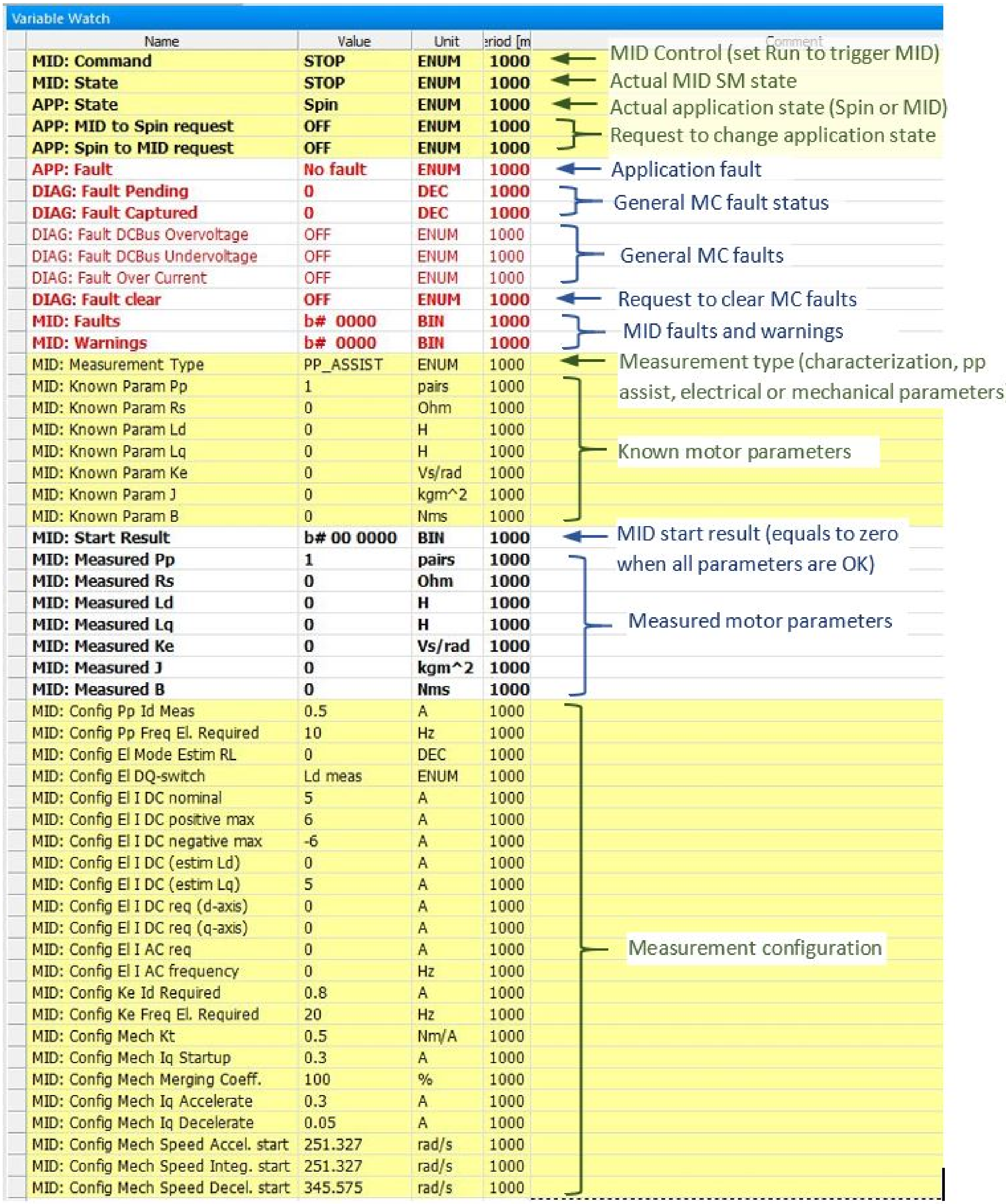
Figure 19. Position FOC control mode

8.3 Switch between Spin and MID

User can switch between two main modes of application: *Spin* and *MID*. *Spin* is for control PMSM (see MCAT FreeMASTER interface (Motor Control Application Tuning)). *MID* is for motor parameters identification (see Motor parameter identification using MID). Actual mode of application is shown in *APP: State* variable watch. The mode change can be made by *APP: MID to Spin request* or *APP: Spin to MID request* variables watch. The result of the change mode request shows *APP: Fault* variable watch. *MID fault* occurs when parameters identification still runs or MID state machine is in the fault state. *Spin fault* occurs when *M1 Application switch* variable watch is ON or *M1 Application state* variable watch is not STOP.

8.4 Identifying parameters of user motor (MID)

Because the model-based control methods of the PMSM drives provide high performance (e.g. 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 described in detail in MID algorithms. MID is controlled via the FreeMASTER "Motor Identification" page shown in Figure 20



Variable Watch

Name	Value	Unit	riod [m	Comment
MID: Command	STOP	ENUM	1000	
MID: State	STOP	ENUM	1000	
APP: State	Spin	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	0	DEC	1000	
MID: Config El DQ-switch	Ld meas	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 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 20. MID FreeMASTER control

Motor parameter identification using MID

The whole MID is controlled via the FreeMASTER "Variable Watch". Motor Identification (MID) sub-block shown in Figure 20. The motor parameter identification is as follows:

1. Set the *MID: Command* variable to STOP.
2. Select the measurement type 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 to infer other parameters (if necessary).
4. Set the measurement configuration paramers 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 9) and the actual *MID: State*, *MID: Faults* (see Table 7), and *MID: Warnings* (see Table 8) variables.
7. When the measurement is successfully finished, the measured motor parameters are in the *MID: Measured* set of variables.

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 by automatically starting a new measurement. If a MID fault appears, the measurement process immeadiatelly 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. See Table 7 and Table 8 for more details on individual faults and warnings.

Table 7. 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 (acceleartion, deceleration) took too long and exceeded 10 seconds.	Raise the *MID: Config Mech Iq Accelerate* or lower the *MID: Config Mech Iq Decelerate*.

Table 8. Measurement warnings

Warning mask	Warning description	Warning reason	Troubleshooting
b#0001	K_e is out of range.	The measured K_e is negative.	Visualy 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 9. 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.

8.5 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 can be selected by *MID: Config El Mode Estim RL* variable. 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 definied be user. In the scopes under "Motor identification" FreeMASTER sub-block can be observed measured currents, estimated parameters etc.

Mode 0

This mode is automatic. Rotor is not fixed. User has to specify nominal current (*MID: Config El I DC nominal* variable).

Mode 1

This mode is automatic. Rotor is not fixed. In this mode will be performed automatic measurement of the inductances for a definied number (*NUM_MEAS*) of different DC current levels using positive values of the DC current. The L_{dq} dependency map can be seen in the "Inductances (Ld, Lq)" recorder. User has to specify 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 current.

Mode 2

This mode is automatic. Rotor must be mechanically fixed in alignment with the first phase. In this mode will be performed automatic measurement of the inductances for a definied number (*NUM_MEAS*) of different DC current levels using both positive and negative values of the DC current. The estimated inductances can be seen in the "Inductances (Ld, Lq)" recorder. 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 current. In most cases nominal current.
- *MID: Config El I DC negative max* - Maximum negative current.

Mode 3

This mode is manual. Rotor must be mechanically fixed in alignment with the first phase. In this mode won't be calculated R_s. 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 (0 for d-axis, 1 for 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.
- *MID: Config El I AC frequency* variable - Required frequency of the AC signal.

8.6 MID algorithms

This section describes how each MID algorithm works.

Stator resistance measurement

For the stator resistance R_s measurement, please, refer to the documentation of *AMCLIB_EstimRL* function from AMMCLib.

Stator inductance

For the inductances L_d and L_q measurement, please, refer to the documentation of *AMCLIB_EstimRL* function from AMMCLib.

BEMF constant measurement

Before the actual BEMF constant (K_e) measurement, the MCAT tool calculates the current controllers and BEMF observer constants from the previously measured R_s, L_d, and L_q. To measure K_e, the motor must spin. I_d is controlled through $I_{d\,meas}$ and the electrical open-loop position is generated by integrating the required speed, which is derived from N_{nom}. When the motor reaches the required speed, the BEMF voltages obtained by the BEMF observer are filtered and K_e is calculated:

$$K_e = \frac{U_{BEMF}}{\omega_{el}} \; [\Omega]$$

When K_e is being measured, you have to visually check to determine whether the motor is spinning properly. If the motor is not spinning properly, perform these steps:

- 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 $I_{d\,meas}$ to produce higher torque when spinning during the open loop.
- Decrease N_{nom} to decrease the required speed for the K_e measurement.

Number of pole-pair assistant

The number of pole-pairs cannot be measured without a position sensor. However, there is a simple assistant to determine the number of pole-pairs (*pp*). The number of the *pp* assistant performs one electrical revolution, stops for a few seconds, and then repeats it. 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 not to count the stops during the first mechanical revolution because the alignment occurs during the first revolution and affects the number of stops. During the *pp* measurement, the current loop is enabled and the I_d current is controlled to $I_{d\,meas}$. The electrical position is generated by integrating the open-loop speed. If the rotor does not move after the start of the number of *pp* assistant, stop the assistant, increase $I_{d\,meas}$, and restart the assistant.

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}.

$$\frac{d\omega_m}{dt} = \frac{1}{J}(T - T_{load} - B\omega_m)\ [rad/s^2]$$

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 first phase of measurement, 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:

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

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

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

The final mechanical parameter estimation can be calculated by integrating:

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

Te moment of inertia is:

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

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.

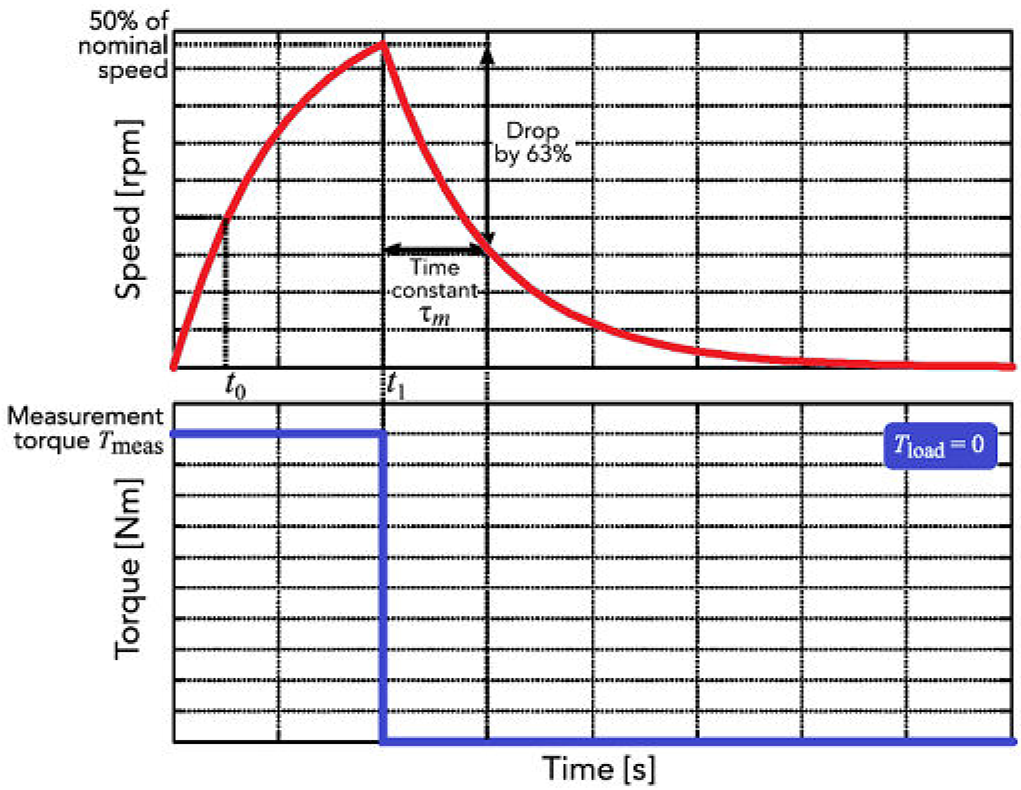


Figure 21. PMSM identification tab

8.7 Initial configuration setting and update

1. Open the PMSM control application FreeMASTER project containing the dedicated MCAT plug-in module.
2. Select the “Parameters” tab.
3. Leave the measured motor parameters or 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 10 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 10. 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

Table continues on the next page...

Table 10. MCAT motor parameters (continued)

Parameter	Units	Description	Typical range
Uph nom	[V]	Motor nominal phase voltage	10-300
N nom	[rpm]	Motor nominal speed	1000-2000

4. 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.
5. Check the fault limits—these fields are calculated using the motor parameters and hardware scales (see Table 11).

Table 11. 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
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

6. Check the application scales—these fields are calculated using the motor parameters and hardware scales.

Table 12. 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	-

7. 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.
8. Click the “Store data” button to save the modified parameters into the inner file.

8.8 Control structure modes

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 22). Use the following equation for calculation V/Hz factor:

$$VHz_{factor} = \frac{U_{phnom} \cdot k_{factor}}{\frac{pp \cdot N_{nom}}{60} \cdot 100} \; [V / Hz]$$

where U_{phnom} is the nominal voltage, k_{factor} is ratio within range 0-100%, pp is the number of pole-pairs and N_{nom} are the nominal revolutions. Changes V/Hz factor won't be propagated to the m1_pmsm_appconfig.h!

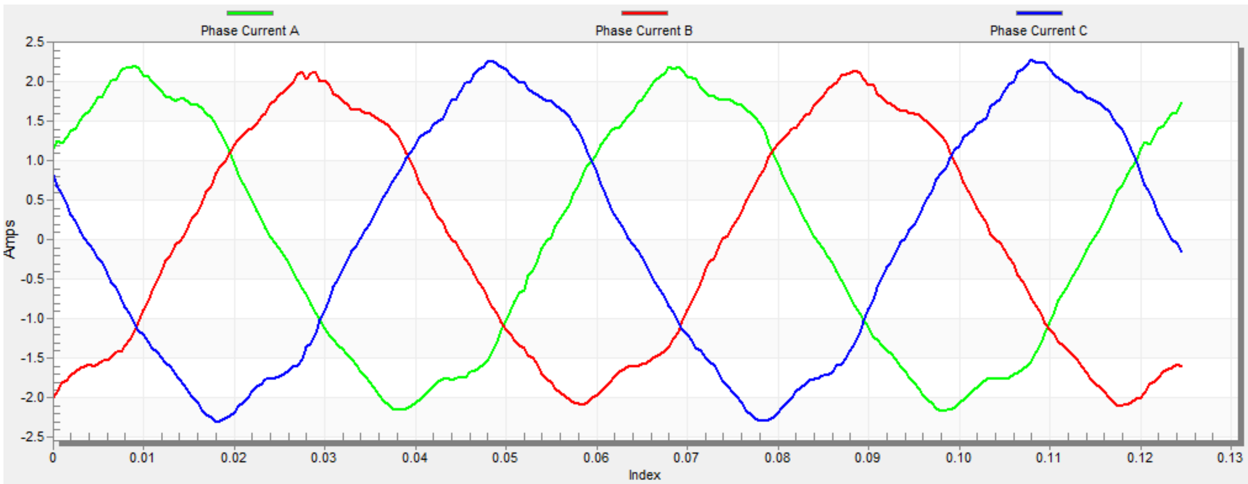


Figure 22. Phase currents

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 23) 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.

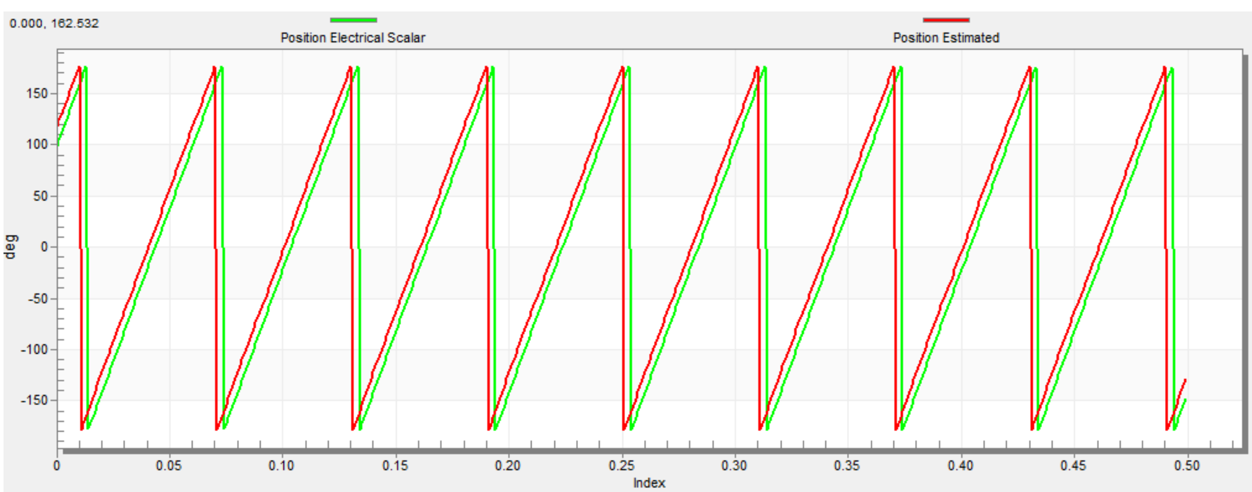


Figure 23. Generated and estimated positions

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 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.

8.9 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 start-up torque to the motor. A correct initial position is needed mainly for high start-up loads (compressors, washers, and so on). The aim of the alignment is to have the rotor in a stable position, without any oscillations before the startup.

1. The alignment voltage is the value applied to the d-axis during the alignment. Increase this value for a higher shaft load.
2. 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.

8.10 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. Lock the motor shaft.
2. Set the required loop bandwidth and attenuation and click the “Update target” button in the “Current loop” tab. The tuning loop bandwidth parameter defines how fast the loop response is whilst the tuning loop attenuation parameter defines the actual quantity overshoot magnitude.
3. Select the “Current Controller Id” recorder.
4. Select the “Current Control” in the FreeMASTER project tree, select "CURRENT_FOC" in "M1 MCAT Control" variable. Set the “M1 MCAT Iq required” variable to a very low value (for example 0.01), and set the required step in “M1 MCAT Id required”. The control loop response is shown in the recorder.
5. 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 (110 Hz) and the settling time of the Id current is long (Figure 24).

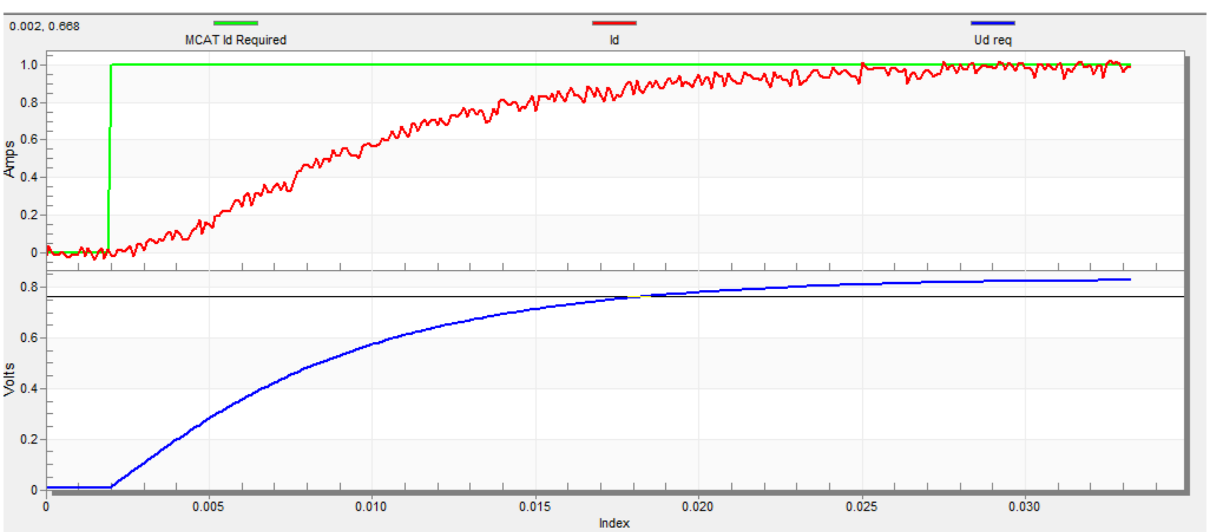


Figure 24. Slow step response of the Id current controller

   - The loop bandwidth (400 Hz) is optimal and the response time of the Id current is sufficient (see Figure 25).

Figure 25. 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 oscillation and overshoot (see Figure 26).

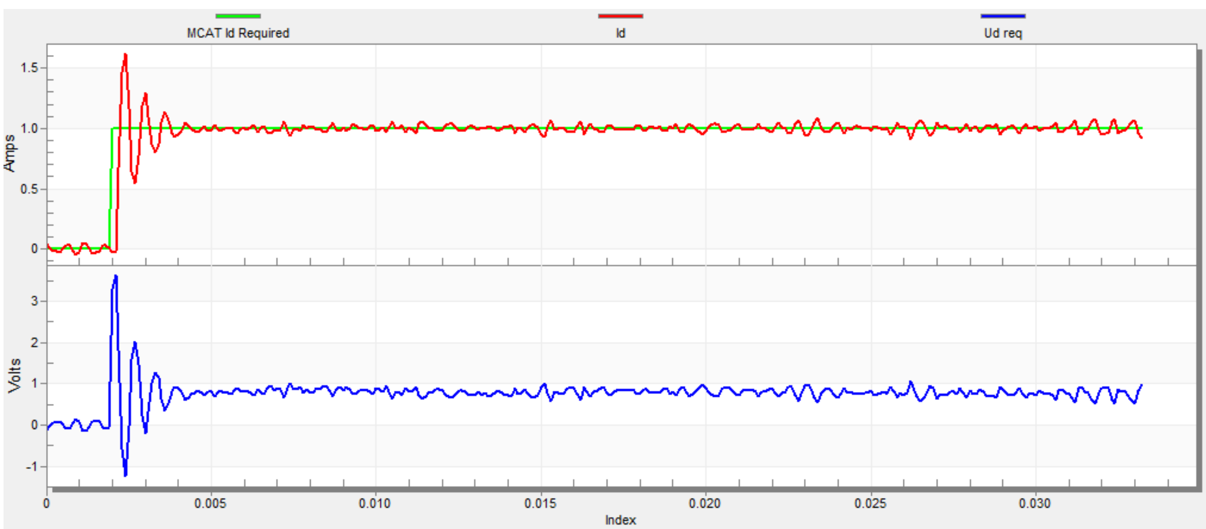


Figure 26. Fast step response of the Id current controller

8.11 Speed ramp tuning

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 27. The ramp increment down is set to 500 rpm/sec and the increment up is set to 3000 rpm/sec.
3. The start-up ramp increment is in the “Sensorless” tab and its value is usually higher than that of the speed loop ramp.

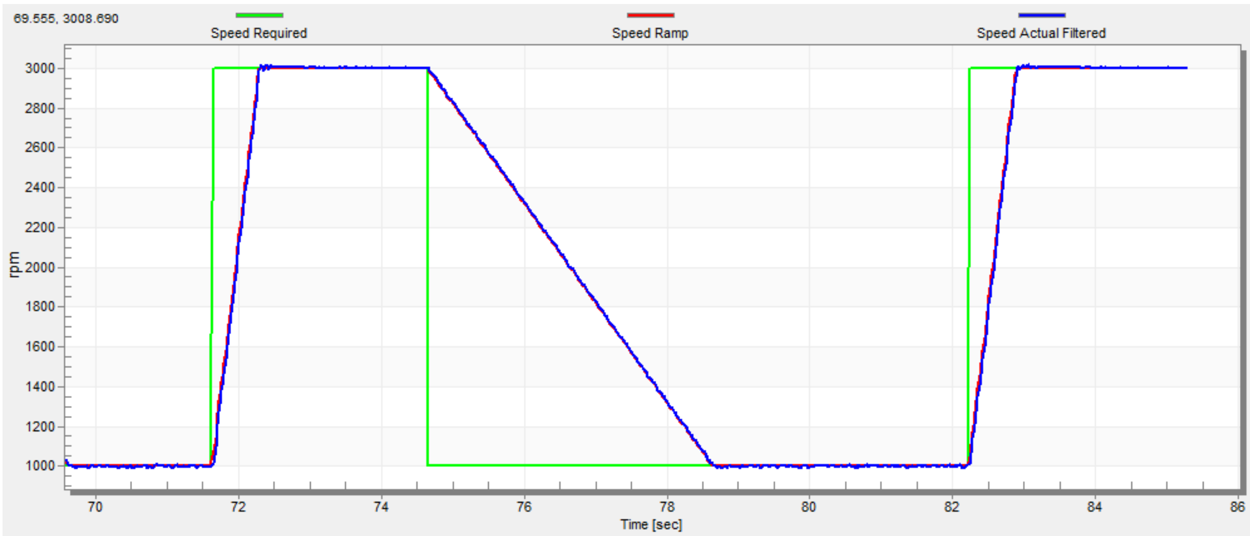


Figure 27. Speed profile

8.12 Open loop startup

1. The start-up 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 start-up tuning can be processed in all control modes besides the scalar control. Setting the optimal values results in a proper motor startup. An example start-up state of low-dynamic drives (fans, pumps) is shown in Figure 28.
2. Select the “Startup” recorder from the FreeMASTER project tree.
3. Set the start-up ramp increment typically to a higher value than the speed-loop ramp increment.
4. Set the start-up current according to the required start-up torque. For drives such as fans or pumps, the start-up 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 half of an electrical revolution. The higher the value, the faster the merge. Values close to 1 % are set for the drives where a high start-up torque and smooth transitions between the open loop and the closed loop are required.
7. Click the “Update Target” button to apply the changes to the MCU.
8. Select “SPEED_FOC” in the "M1 MCAT Control" variable.
9. Set the required speed higher than the merging speed.
10. Check the start-up response in the recorder.
11. Tune the start-up parameters until you achieve an optimal response.
12. If the rotor does not start running, increase the start-up current.
13. If the merging process fails (the rotor is stuck or stopped), decrease the start-up ramp increment, increase the merging speed, and set the merging coefficient to 5 %.

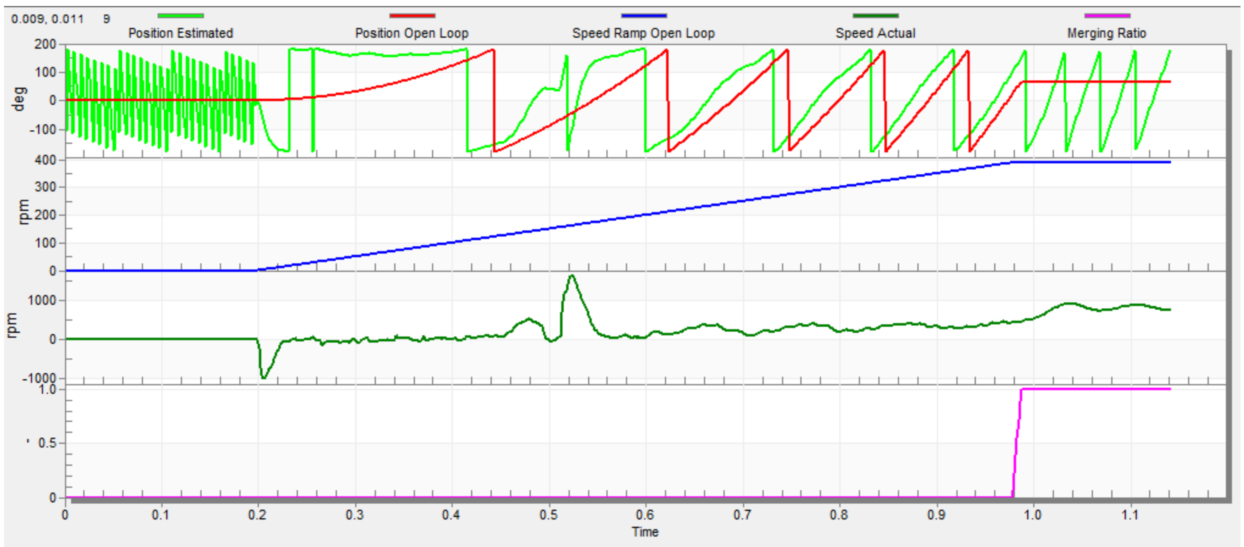


Figure 28. Motor startup

8.13 BEMF observer tuning

1. The bandwidth and attenuation parameters of the BEMF observer and the tracking observer can be tuned. 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. Click the “Update target” button to apply the changes to the MCU.
5. Select the “Observer” recorder from the FreeMASTER project tree and check the observer response in the "Observer" recorder.

8.14 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 the example response in Figure 29). There are dozens of approaches to tune the PI controller constants. The following steps provide an approach to set and tune the speed PI controller for a PM synchronous motor:

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.
  - — Click the “Update Target” button to apply the changes to the MCU.

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.
   - Click the “Update target” button to apply the changes to the MCU.
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” (see Figure 29).

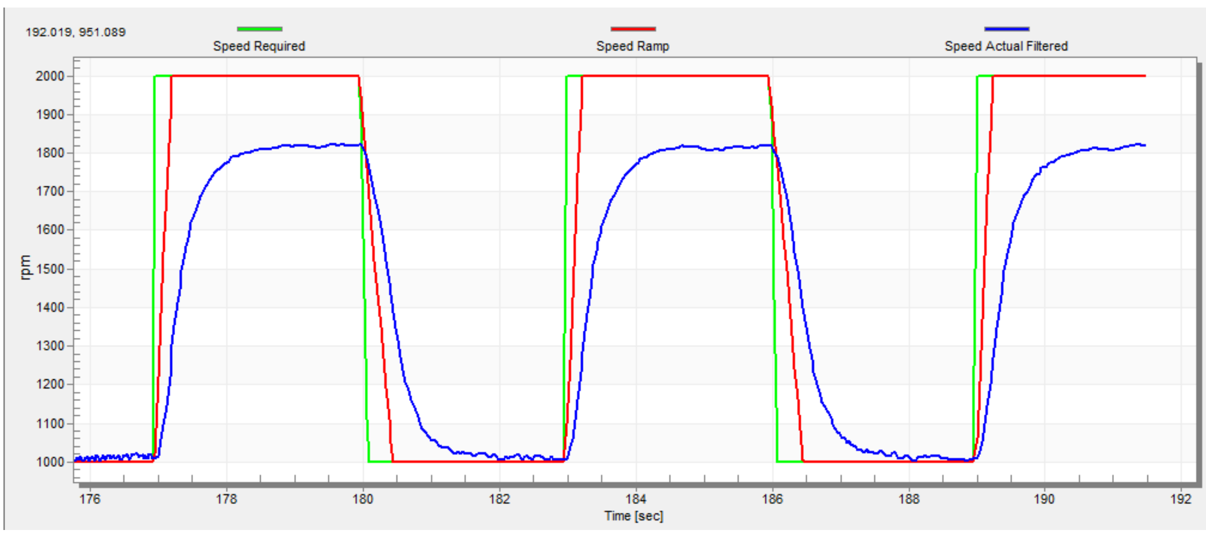


Figure 29. Speed controller response—SL_Ki value is low, Speed Ramp is not achieved

- The “SL_Kp” value is low, the “Speed Actual Filtered” greatly overshoots, and the long settling time is unwanted (see Figure 30).

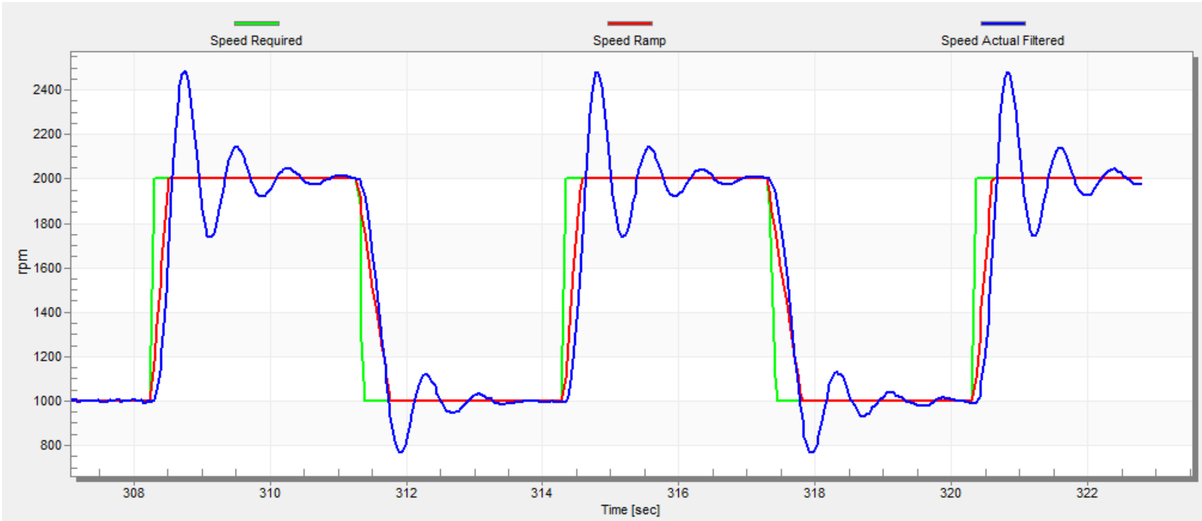


Figure 30. Speed controller response—SL_Kp value is low, Speed Actual Filtered greatly overshoots

- The speed loop response has a small overshoot and the “Speed Actual Filtered” settling time is sufficient. Such response can be considered optimal (see Figure 31).

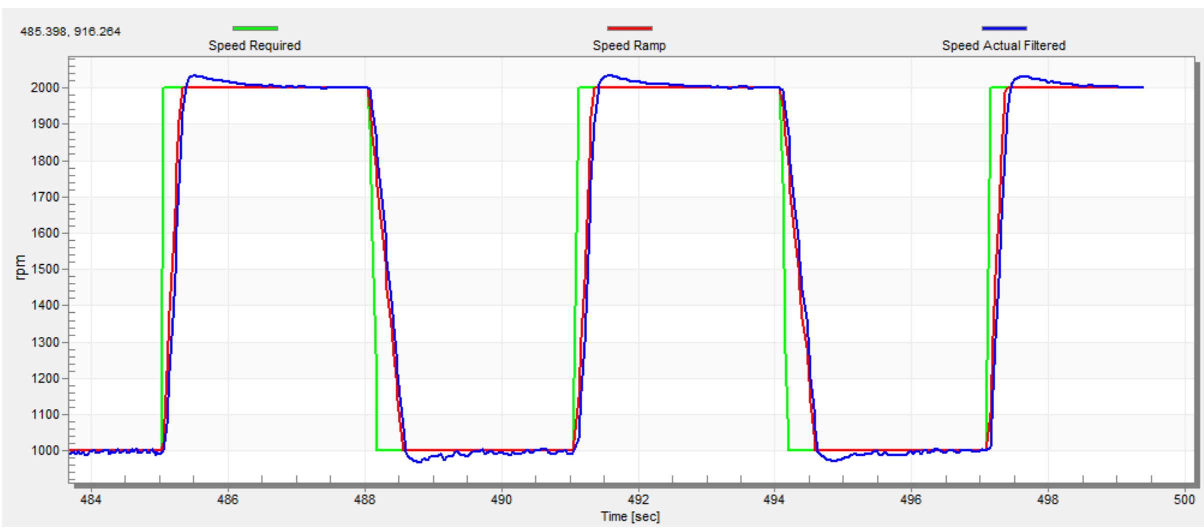


Figure 31. Speed controller response—speed loop response with a small overshoot

Chapter 9 Conclusion

This user's guide describes the implementation of the sensor and sensorless Field-Oriented Control of a 3-phase PMSM on the NXP LPC55S36 with the FRDM-MC-LVPMSM NXP Freedom Development Platform. The hardware-dependent part of the control software is described in Hardware setup. The motor-control application timing is described in LPC55S36 hardware timing and synchronization and the peripheral initialization is described in Motor-control peripheral initialization. The motor user interface and remote control using FreeMASTER are as follows. The motor parameters identification theory and the identification algorithms are described in Identifying parameters of user motor (MID).

Chapter 10
Acronyms and abbreviations

Table 13. 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

Chapter 11
References

These references are available on www.nxp.com:

1. *Sensorless PMSM Field-Oriented Control* (document DRM148).
2. *Motor Control Application Tuning (MCAT) Tool for 3-Phase PMSM* (document AN4642).
3. *NXP Automotive Math and Motor Control Library (AMMCLib) set* (e.g. document Automotive Math and Motor Control Library Set for S32K14x).

Chapter 12
Useful links

1. PMSM Control Reference Design www.nxp.com/motorcontrol_pmsm
2. BLDC Control Reference Design www.nxp.com/motorcontrol_bldc
3. ACIM Control Reference Design www.nxp.com/motorcontrol_acim
4. FRDM-MC-PMSM Freedome Development Platform
5. *SCTimer/PWM Cookbook* (document AN11538)
6. MCUXpresso IDE - Importing MCUXpresso SDK
7. MCUXpresso Config Tool
8. MCUXpresso SDK Builder (SDK examples in several IDEs) https://mcuxpresso.nxp.com/en/welcome

Chapter 13
Revision history

Table 14 summarizes the changes done to the document since the initial release.

Table 14. Revision history

Revision number	Date	Substantive changes
0	11/2021	Initial release

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer’s applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, UMEMS, EdgeScale, EdgeLock, eIQ, and Immersive3D are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© NXP B.V. 2021. All rights reserved.

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 11/2021

Document identifier: PMSMLPC55S36EVK