NXP Semiconductors
User's Guide

Document identifier: 3PPMSMCLPCUG
Rev. 0, 02/2020

MCUXpresso SDK 3-Phase PMSM Control (LPC)

Contents

Chapter 1
Introduction

This user's guide describes the implementation of the sensorless Motor Control software for the 3-phase Permanent Magnet Synchronous Motor (PMSM), including the motor parameters identification algorithm, on the NXP LPC55S6x MCU based on the Arm® Cortex®-M33 architecture. The sensorless control software and PMSM control theory in general are described in *Sensorless PMSM Field-Oriented Control* (documentDRM148). The NXP Freedom board (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 addressed 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.

This document provides instructions for running and controlling the PMSM project using the LPCXpresso55S69 development board with the Freedom development board. The software provides the sensorless field-oriented speed, torque, and scalar control. You can control the application using the board buttons or the FreeMASTER application. The motor identification and application tuning is done using the MCAT tool integrated in the FreeMASTER page. The required software, hardware setup, jumper settings, project arrangement, and user interface is described in the following sections. For more information, visit www.nxp.com/motorcontrol_pmsm.

Chapter 2
Hardware setup

The PMSM Field-Oriented Control (FOC) application runs on the FRDM-MC-LVPMSM development platform with the LPC55S69-EVK development tool, in combination with the Linix 45ZWN24-40 permanent magnet synchronous motors.

2.1 FRDM-MC-LVPMSM

This evaluation board, in a shield form factor, effectively turns a NXP Freedom development board into a complete motor control reference design, compatible with existing NXP Freedom development boards. Freedom motor control headers are compatible with 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 a complete NXP Freedom motor-control development kit is shown below.

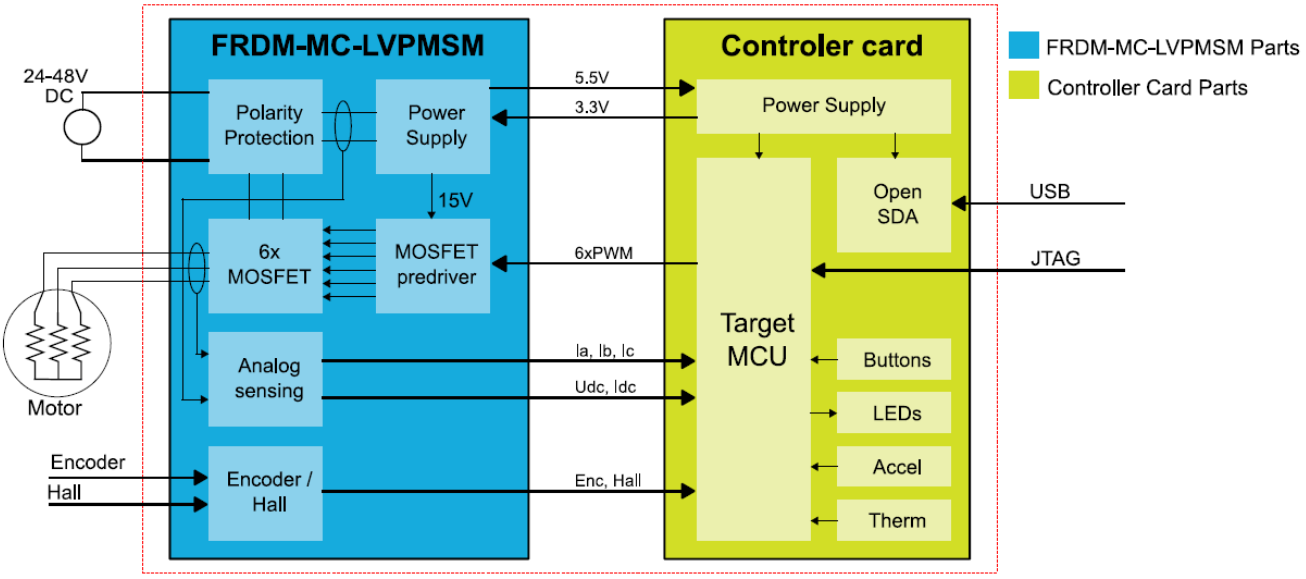

Figure 1. Motor-control development platform block diagram

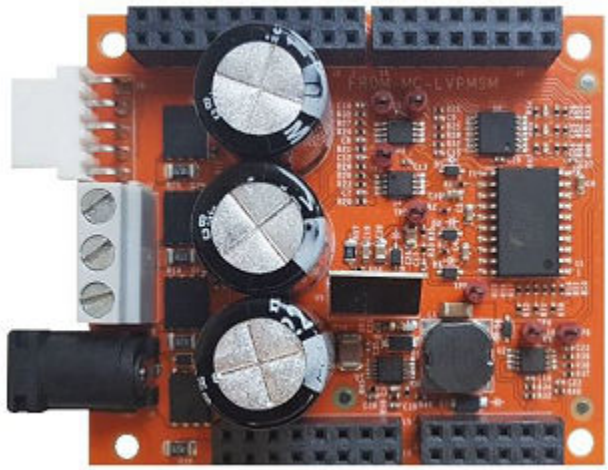

Figure 2. FRDM-MC-LVPMSM

The FRDM-MC-LVPMSM does not require any complicated setup. For more information about Freedom development platform see 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 below.

Table 1. 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 LPC55S69-EVK

The LPCXpresso55S69 development board is an ideal platform for evaluation and development with the LPC55S6x 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. Configure the jumper settings according to Table 2 for the motor-control application to work properly.

Table 2. LPC55S69-EVK jumper settings

Jumper	Setting	Jumper	Setting	Jumper	Setting
J3	1-2	J7	1-2	P1	open
J4	open	J12	open	P4	3.3V
J6	FS	-	-	-	-

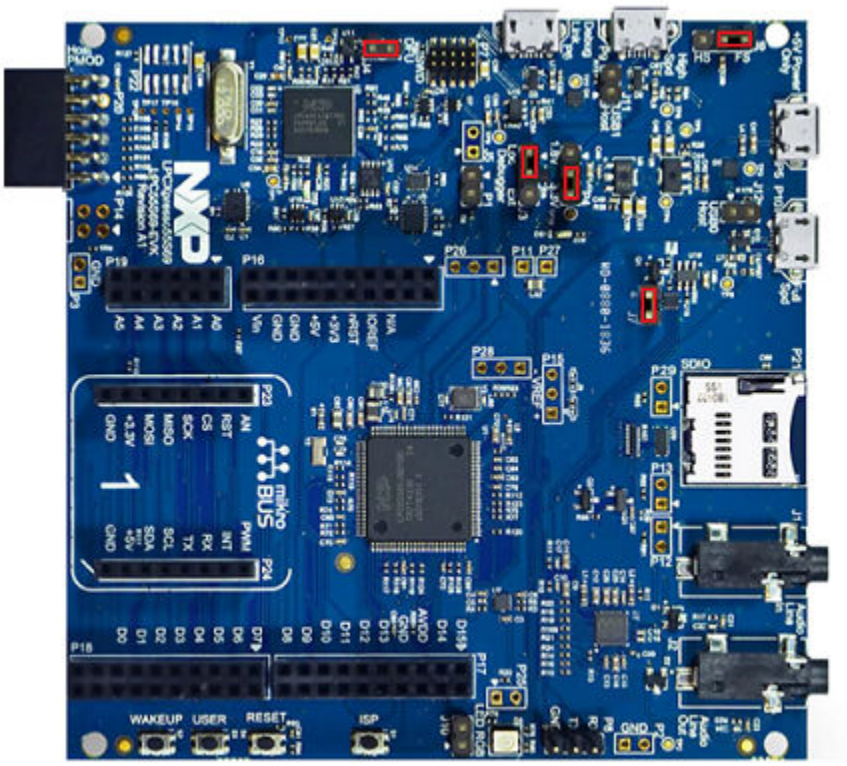

Figure 4. LPC55S69-EVK board with highlighted jumper settings

Hardware assembling

1. Wire the FRDM-MC-LVPMSM power stage to the LPC55S69-EVK board according to the pin assignment (Table 3) and the interconnection diagram (Figure 5).
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 USB connector (Debug Link - P6) on the EVK board.
4. Plug the 24-V DC power supply to the DC power connector on the Freedom PMSM power stage.

Table 3. LPC55S69-EVK pin assignment

FRDM-MC-LVPMSM	Connection	LPC55S69-EVK	
PWM_AT	J3, 15 <-> P18, 5	LEDR/PWM_ARD	PIO1_4
PWM_AB	J3, 13 <-> P18, 20	PIO0_18_GPIO_ARD	PIO0_18
PWM_BT	J3, 11 <-> P18, 6	PLU_CLKIN/GPIO/SD1_D0	PIO1_25
PWM_BB	J3, 9 <-> P18, 3	PIO1_10_GPIO_ARD	PIO1_10
PWM_CT	J3, 7 <-> P18, 19	SD1_CARD_DET	PIO1_17
PWM_CB	J3, 5 <-> P17, 13	LSPI_HS_MOSI	PIO0_26
3V3	J3, 4 <-> P16, 12	VDD_TARGET	-
GND	J3, 14 <-> P16, 16	GND	-
CUR_A	J2, 1 <->P19, 2	ADC0_N	PIO0_16

Table continues on the next page...

Table 3. LPC55S69-EVK pin assignment (continued)

FRDM-MC-LVPMSM	Connection	LPC55S69-EVK	
CUR_B	J2, 3 <-> P19, 4	ADC0_P	PIO0_23
CUR_C	J2, 5 <->P18, 11	PIO0_15_GPIO_ARD	PIO0_15
VOLT_DCB	J2, 7 <->P17, 19	PIO1_8_GPIO_ARD	PIO1_8
CUR_DCB	J2, 9 <-> P19, 6	COMPARATOR	PIO0_0

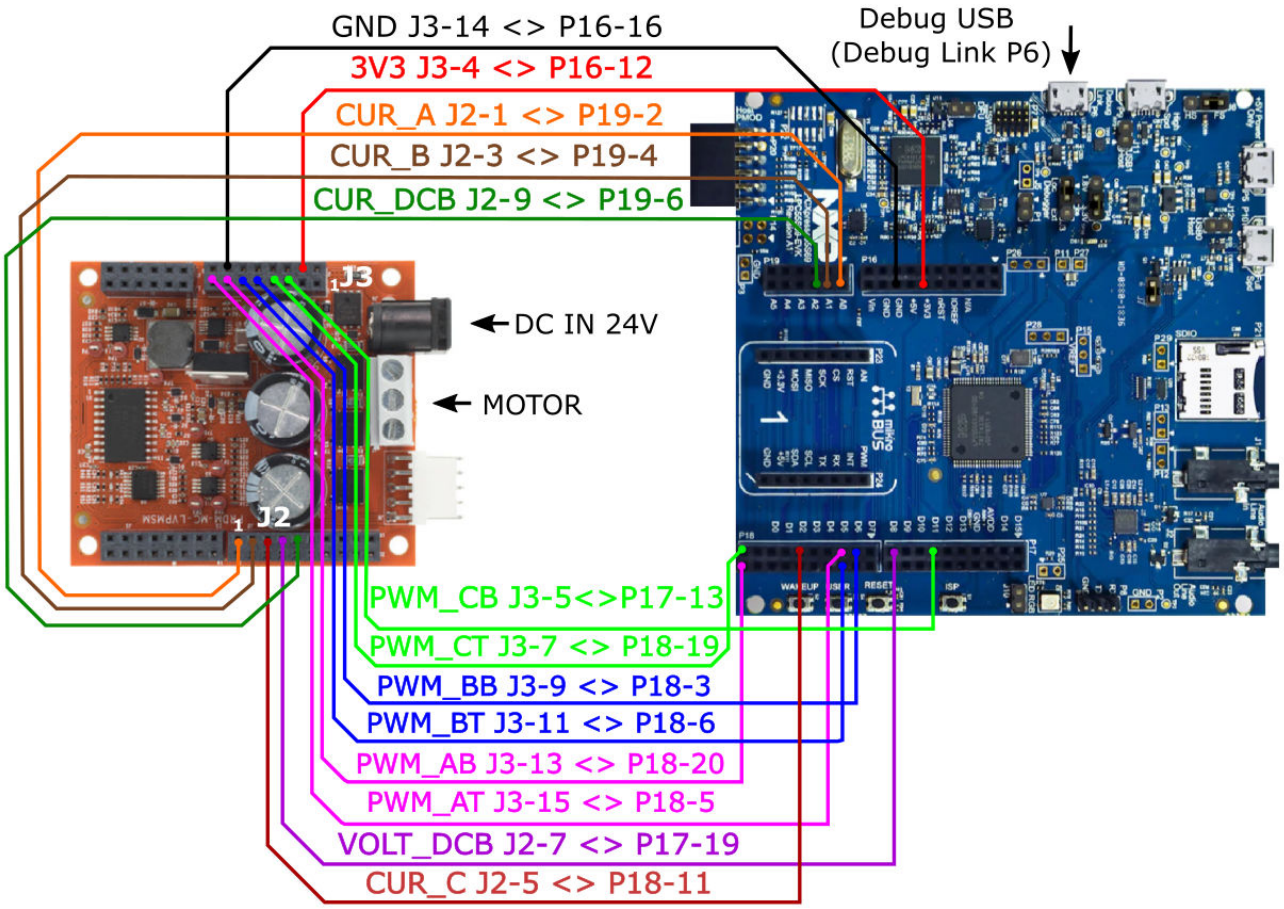

Figure 5. LPC55S69-EVK interconnection diagram

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-55S6x

The LPC55S6x MCU family is built upon the world's first general-purpose 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 LPC55S6x 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 LPC55S6x 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.

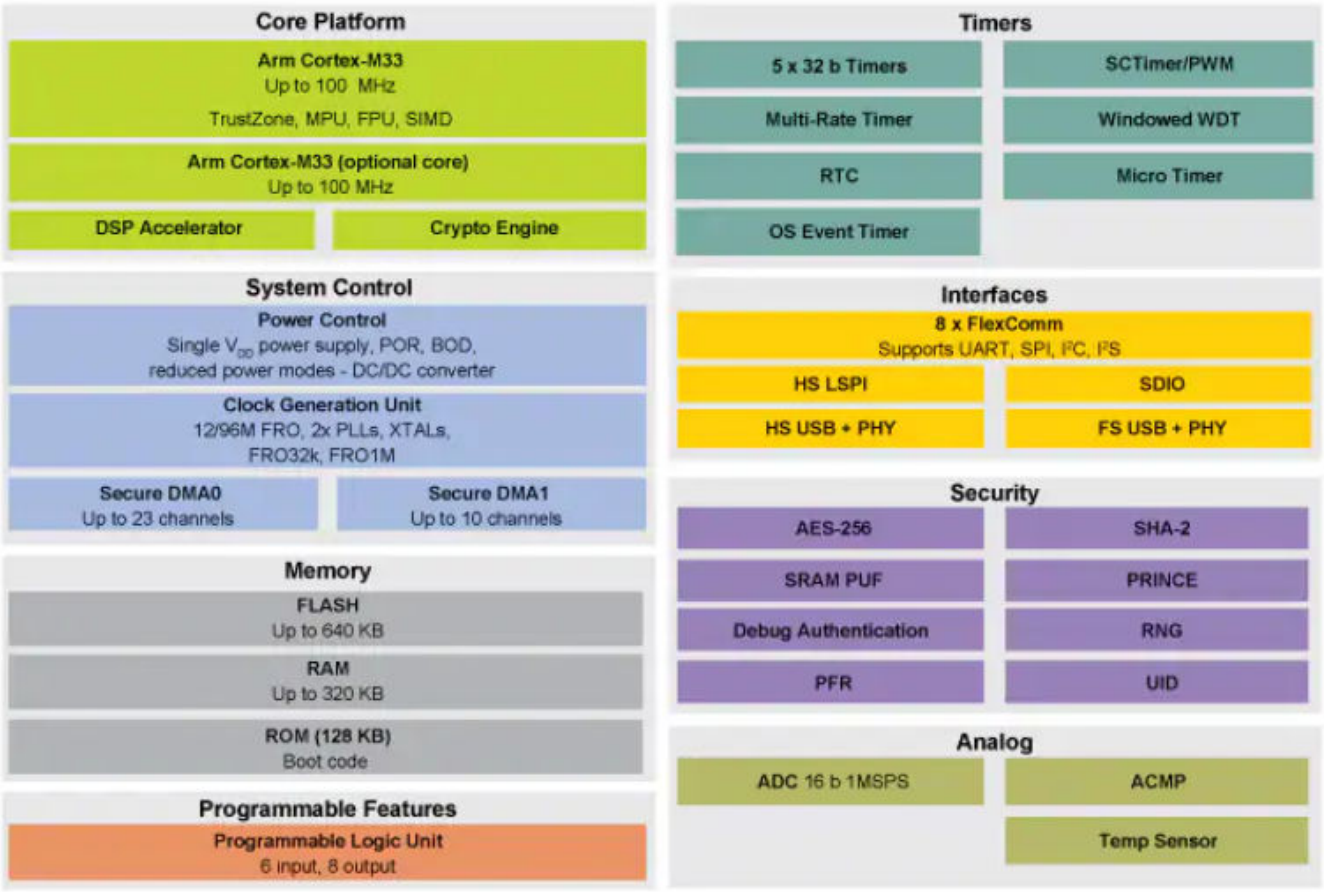

Figure 6. LPC55S6x block diagram

3.1.1 LPC55S69 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, it is possible to 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. The timing diagram is shown in Figure 7.

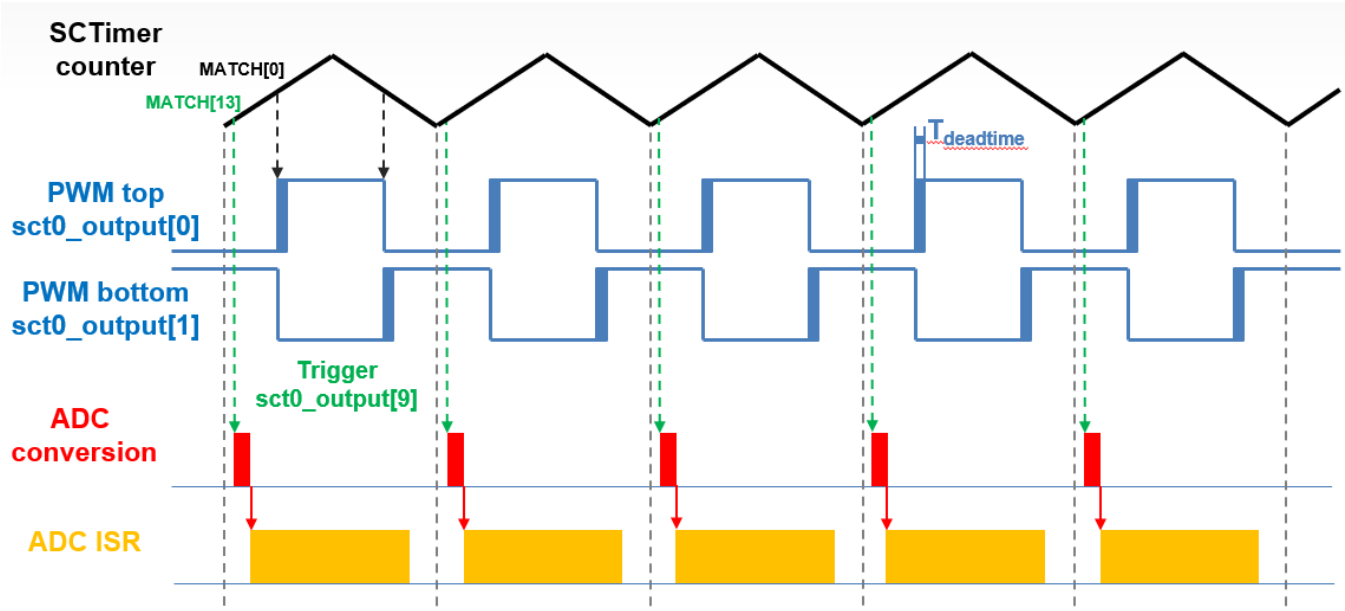

Figure 7. Hardware timing and synchronization on LPC55S69

- The top signal shows the SCT counter (SCT0 counter). The dead time is emphasized at the PWM top and PWM bottom signals.
- The sct0_output[9] generates a trigger for the ADC with a short delay. This delay ensures correct current sampling at duty cycles close to 100 %.
- When the ADC conversion is completed, the ADC ISR (ADC interrupt) is entered. The FOC calculation is done in this interrupt.

3.1.2 LPC55S69 peripheral settings

This section describes the peripherals used for the motor control. On LPC55S69, the SCTimer is used for 6-channel PWM generation. The 16-bit ADC is used for the phase currents and DC-bus voltage measurement. The SCTimer and ADC are synchronized via a trigger from "SCTimer sct0_output[9]". The following settings are in the *mcdrv_lpcexpresso55s69.c* and *board.c* files and in their header files.

Clock select and control (SYSCON)

The clock source for the registers and memories is derived from the main clock. The main clock is selected from the FRO high-speed output (fro_hf) of the 96-MHz internal oscillator.

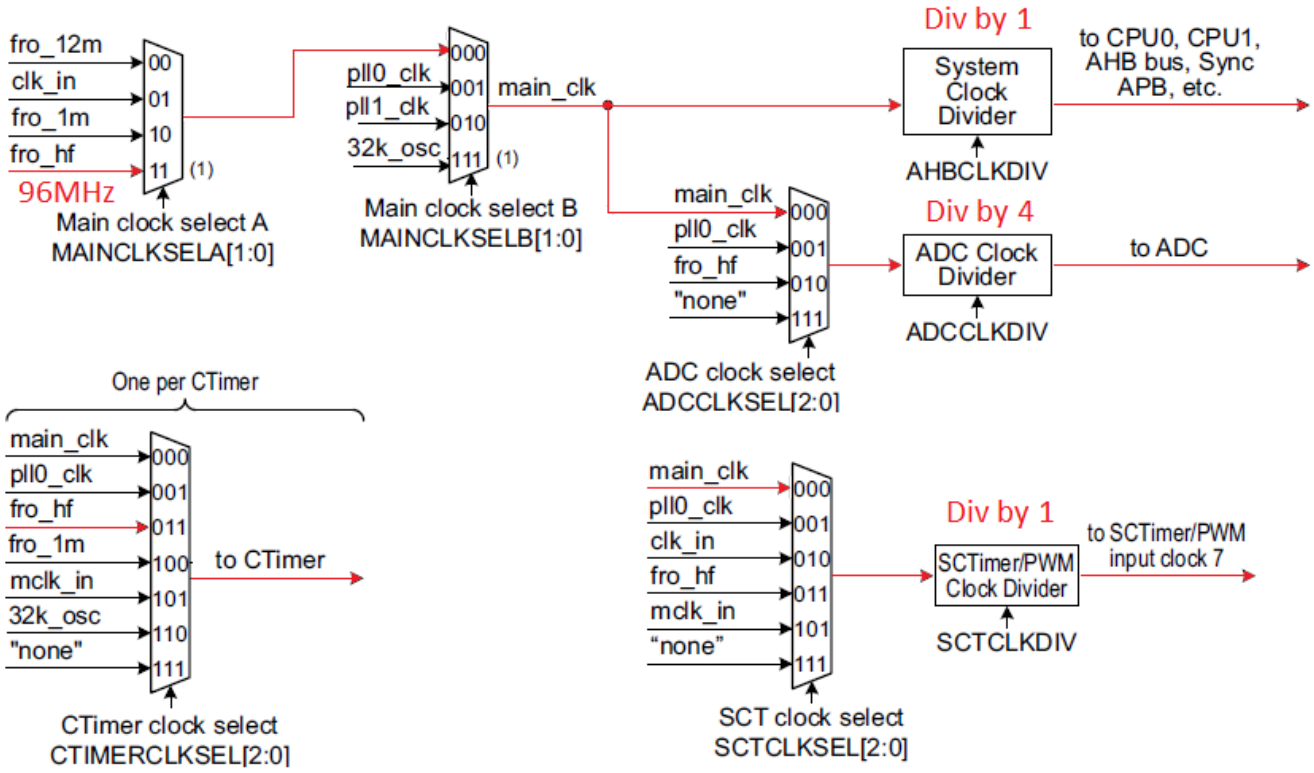

Figure 8. LPC55S69 clock source for motor-control peripherals

The clock sources for the peripherals used for the motor control are listed in Table 4.

Table 4. LPC55S69 clock source for motor-control peripherals

	Clock source	Clock div	Clock root frequency
SCTimer	main_clk	SCTCLKDIV	96 MHz
CTimer	main_clk	-	96 MHz
ADC	main_clk	ADCCLKDIV	24 MHz

For more details, see the *LPC55S6x User Manual* (document UM11126).

PWM generation - SCT0

- The SCT0 is clocked from "main_clk" (96 MHz).
- SDK initialization structures are used for the SCT0 initiallization.
- SCTIMER_Out_0 - SCTIMER_Out_5 are used for the 3-phase PWM generation.
- kSCTIMER_Out_9 is used for the ADC generation trigger.
- Dead time is inserted using the "ui16DeadTimePWM in g_sM1Pwm3ph" structure.

Analog sensing - ADC

- The clock frequency of the ADC is 24 MHz. It is taken from "main_clk" and divided by four.
- The ADCs operate as 10-bit with the single-ended conversion and hardware trigger selected. The ADCs are triggered from "sct0_output[9]".

- The watermark interrupt is enabled and serves the FOC fast loop algorithm generated after the last scan is completed.

Slow loop interrupt generation - CTIMER2

The standard timer 2 is used to generate the slow-loop interrupt.

- The CTIMER2 is clocked from the "main_clk" divided with a clock frequency of 96 MHz.
- The slow loop is usually ten times slower than the fast loop. Therefore, the interrupt is generated after the counter counts to CTIMER_CLK_FREQ / g_sClockSetup.ui16M1SpeedLoopFreq. The speed loop frequency is set in the M1_SPEED_LOOP_FREQ macro and equals 1000 Hz.
- An interrupt (which serves the slow-loop period) is enabled and generated at the reload event.

Analog comparator - ACMP0

- The analog comparator is used for the over-current detection.
- SDK initialization structures are used for the SCT0 initiallization.
- Channel 1 is used as the positive input.
- Channel 0 is used as the negative input.

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 shows 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-55S69 CPU load and memory usage

	Debug configuration	Release configuration
Program flash	76 136 B	41 048 B

Table continues on the next page...

Table 5. LPC-55S69 CPU load and memory usage (continued)

	Debug configuration	Release configuration
SRAM	19 864 B	19 800 B
Maximum CPU load (core 0)	44.8 %	44.8 %

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 9.

Figure 9. Directory tree

The main project folder *pack_motor_lpcxx\boards\lpcxpressoxx\demo_apps\mc_pmsm\pmsm_snsless\cm33_corex* contains the following folders:

- *iar*—for the IAR Embedded Workbench IDE.
- *armgcc*—for the GNU Arm IDE.

- *mdk*—for the uVision Keil IDE.

The folder contains also the following files:

- *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*—contains the CPU clock setup functions. These files are going to be generated by the clock tool in the future.
- *mcdrv.h*—this file ensures the abstraction of the *mcdrv_lpcxpressoxx.h* file inclusion.
- *mcdrv_lpcxpressoxx.c*—contains the motor-control driver peripherals initialization functions that are specific for the board and MCU used.
- *mcdrv_lpcxpressoxx.h*—header file for *mcdrv_lpcxpressoxx.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*—port configuration files. It is recommended to generate these files in the pin tool.

The *pack_motor_lpcxx\boards\lpcxpressoxx\demo_apps\mc_pmsm\pmsm_snsless* folder contains the FreeMASTER project file *pmsm_float.pmp*. Open this file in the FreeMASTER tool and use it to control the application.

The *pack_motor_lpcxx\middleware\motor_control\freemaster\pmsm_float* folder 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.
- *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 3.0 and one of the following IDEs on your PC to run and control the PMSM application properly:

- IAR Embedded Workbench IDE v8.40.2 or higher.
- MCUXpresso v11.1.0.
- ARM-MDK - Keil µVision version 5.25.2.
- FreeMASTER Run-Time Debugging Tool 3.0.

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 in situations where 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.
- Ta022—possible ROM access (<ptr>) from within a __ramfunc function.
- Ta023—call to a non __ramfunc function (somefunction) from within a __ramfunc function.

The Arm-MDK Keil µVision IDE suppresses these warnings:

- 66—the enumeration value is out of the “int” range.
- 1035—a single-precision operand is implicitly converted to a double-precision operand.
- 1296—the extended constant initializer is used.

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 *mcdrv_lpcxx.c* source file and the *mcdrv_lpcxx.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 *mcdrv_lpcxx.h* header file provides several macros that can be defined by the user:

- *M1_MCDRV_ADC*—this macro specifies which ADC periphery is used. If you select an unsupported periphery, a preprocessor error is issued.
- *M1_MCDRV_PWM3PH*—this macro specifies which PWM periphery is used. If you select an unsupported periphery, a preprocessor error is issued.
- *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.
- *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). Change the order of the motor phases this way.
- *M1_ADC[1,2]_PH_[A..C]*—these macros are used to assign the ADC channels for the phase current measurement. The general rule is that at least one of the phase currents 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, these API-serving ADC and PWM peripherals are available:

- The available APIs for the ADC are:
  - *mcdrv_adc_t*—MCDRV ADC structure data type.
  - *bool_t 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.
  - *bool_t 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.
  - *bool_t M1_MCDRV_CURR_3PH_CALIB_INIT(mcdrv_adc_t*)*—this function initializes the phase-current channel-offset measurement. This function always returns true.
  - *bool_t 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.
  - *bool_t 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.

  - bool_t *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 PWM are:
  - *mcdrv_pwma_pwm3ph_t*—MCDRV PWM structure data type.
  - *bool_t M1_MCDRV_PWM_PERIPH_INIT()*—this function is by default called during the PWM periphery initialization procedure invoked by the *MCDRV_Init_M1()* function.
  - *bool_t M1_MCDRV_PWM3PH_SET(mcdrv_pwma_pwm3ph_t*)*—this function updates the PWM phase duty cycles. This function always returns true.
  - *bool_t M1_MCDRV_PWM3PH_EN(mcdrv_pwma_pwm3ph_t*)*—calling this function enables all PWM channels. This function always returns true.
  - *bool_t M1_MCDRV_PWM3PH_DIS (mcdrv_pwma_pwm3ph_t*)*—calling this function disables all PWM channels. This function always returns true.
  - *bool_t M1_MCDRV_PWM3PH_FLT_GET(mcdrv_pwma_pwm3ph_t*)*—this function returns the state of the over-current fault flags and automatically clears the flags (if set). This function returns true when an over-current event occurs. Otherwise, it returns false.

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 remotely controlled using FreeMASTER:

- Using the Motor Control Application Tuning (MCAT) interface in the “Control Structure” tab or the “Application control” tab (controlling the demo mode).
- 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 3.0 at www.nxp.com/freemaster. To run the FreeMASTER application including the MCAT tool, double-click the *pmsm_float.pmp* file located in the *pack_motor_lpcxx\boards\lpcxpressoxx\demo_apps\mc_pmsm\pmsm_snsless* folder. The FreeMASTER application starts and the environment is created automatically, as defined in the **.pmp* file.

8.1 Establishing FreeMASTER communication

The remote operation is provided by FreeMASTER via the USB interface. Perform these 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.

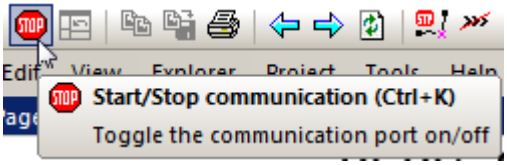

Figure 10. Red “STOP” button placed in top left-hand corner

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=115200”. Otherwise, the FreeMASTER warning popup window appears.

Figure 11. 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 these steps:

1. Go to the “Project -> Options -> Comm” tab and make sure that “SDA” is set in the “Port” option and the communication speed is set to 115200 bps.

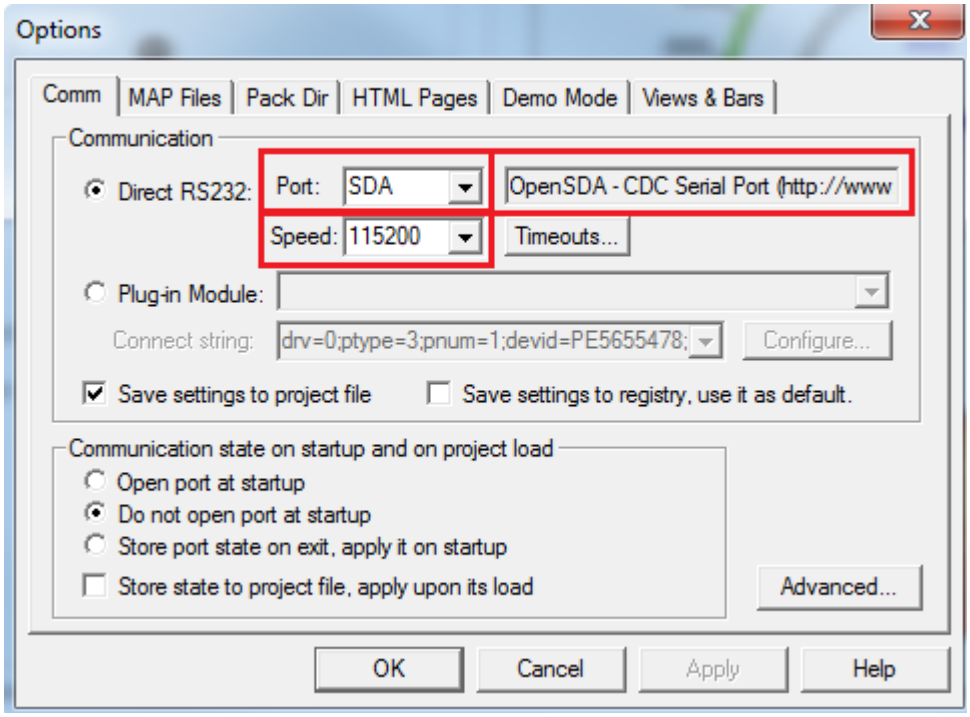

Figure 12. FreeMASTER communication setup window

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 13. 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 “App ID” line shows “offline”. When the communication with the target MCU is established using a correct software, the “App ID” line displays the board name “pmsm_used_board” 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.

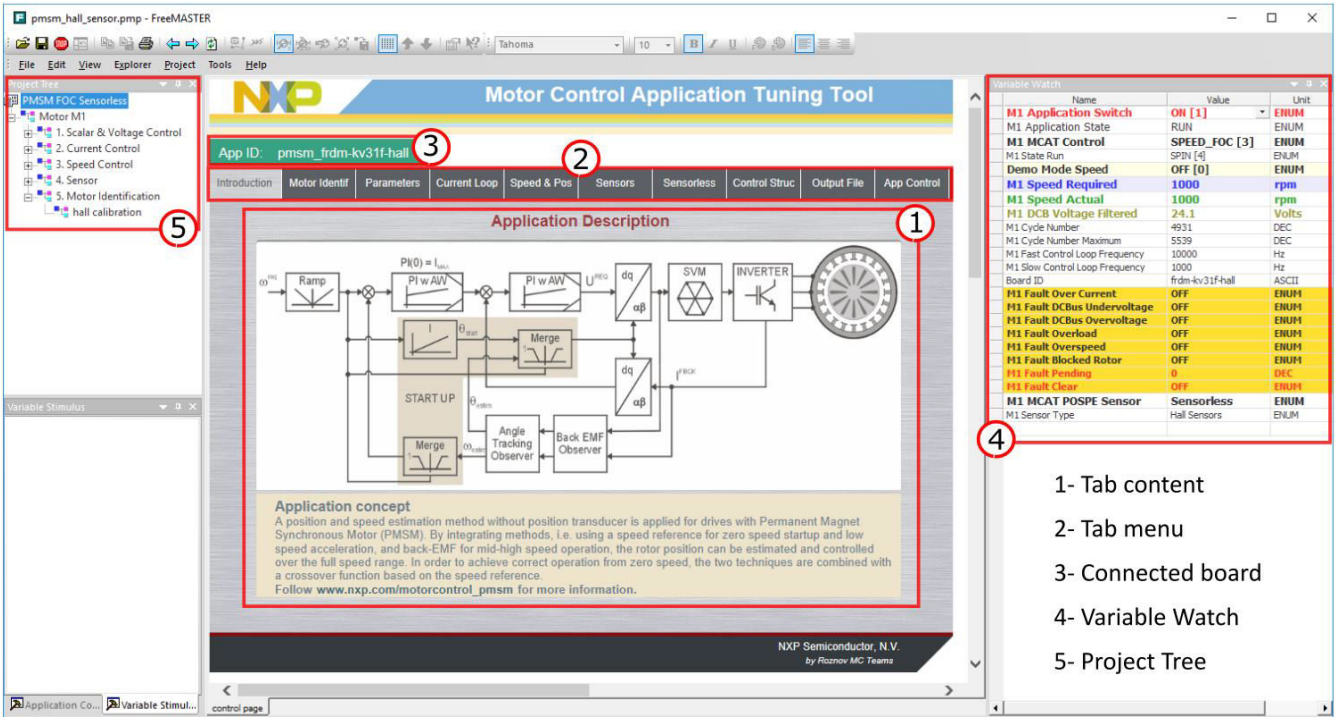

Figure 13. MCAT layout

In the default configuration, these tabs are available:

- “Introduction”—welcome page with the PMSM sensor/sensorless FOC diagram and a short description of the application.
- “Motor Identif”—PMSM semi-automated parameter measurement control page. The PMSM parameter identification is more closely described further on in this document.
- “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 & Pos”—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.
- “Control Struc”—this application control page enables you to select and control the PMSM using different techniques (scalar—Volt/Hertz control, voltage FOC, current FOC, speed FOC, and position FOC). The application state is also shown in this tab.
- “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_acim_appconfig.h* file, which is then used to preset all application parameters permanently at the project rebuild.
- “App page”—this tab contains the graphical elements like the speed gauge, DC-bus voltage measurement bar, and variety of switches which enable a simple, quick, and user-friendly application control. The fault clearing and the demo mode (which sets various predefined required speeds and positions over time) can be also controlled from here.

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 “Reload Data” and “Store 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—“Control Struc” tab

The application can be controlled through the “Control Struc” tab, which is shown in Figure 14. The state control area on the left side of the screen shows the current application state and enables you to turn the main application switch on or off (turning a running application off disables all PWM outputs). The “Cascade Control Structure” area is placed in the right-hand side of the screen. Here you can choose between the scalar control and the FOC control using the appropriate buttons. The selected parts of the FOC cascade structure can be enabled by selecting “Voltage FOC”, “Current FOC”, and “Speed FOC” (sensor/sensorless). This is useful for application tuning and debugging.

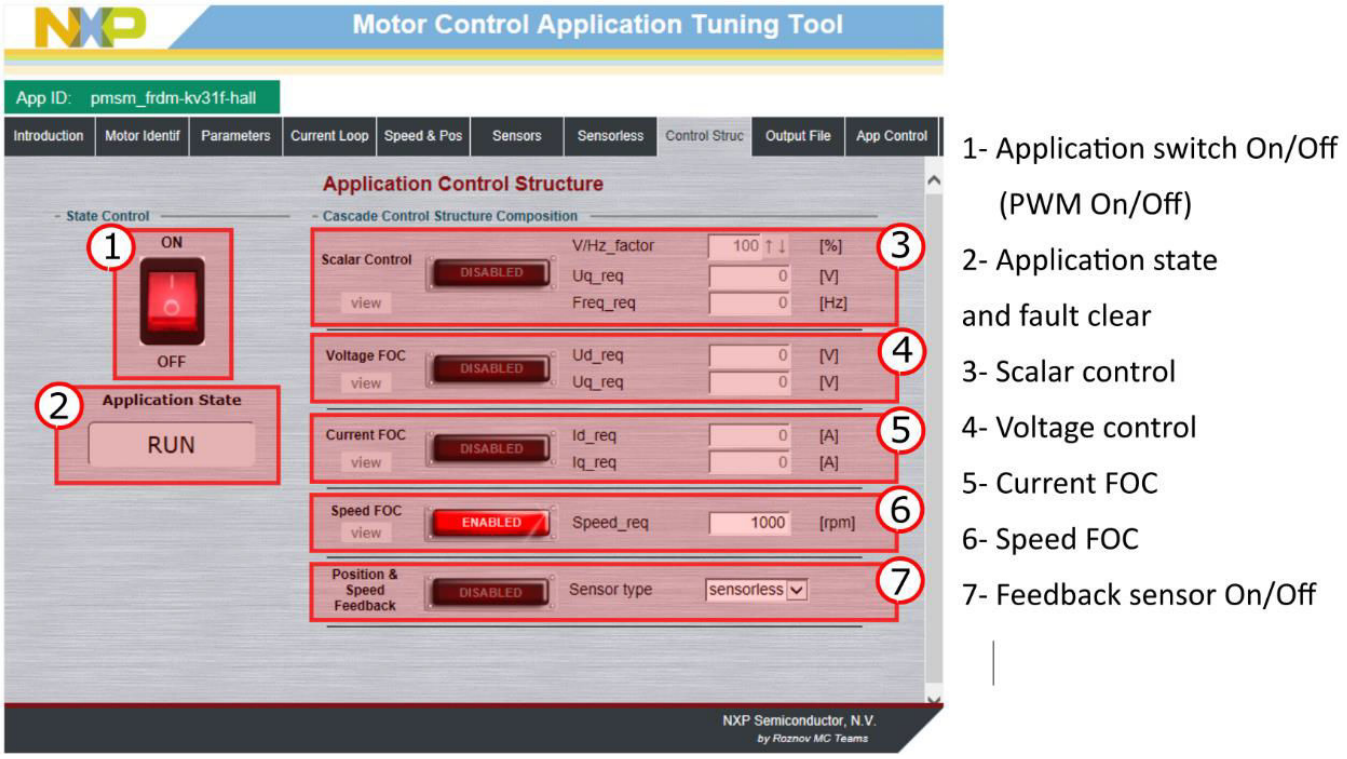

Figure 14. MCAT for PMSM control page

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.

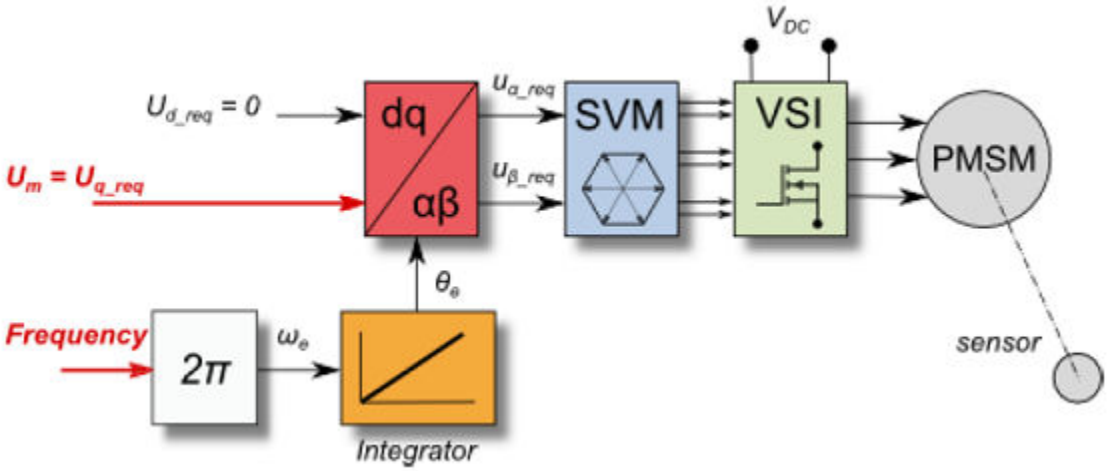

Figure 15. Scalar control mode

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 “Ud_req” and “Uq_req” fields. This control method is useful for the BEMF observer functionality check.

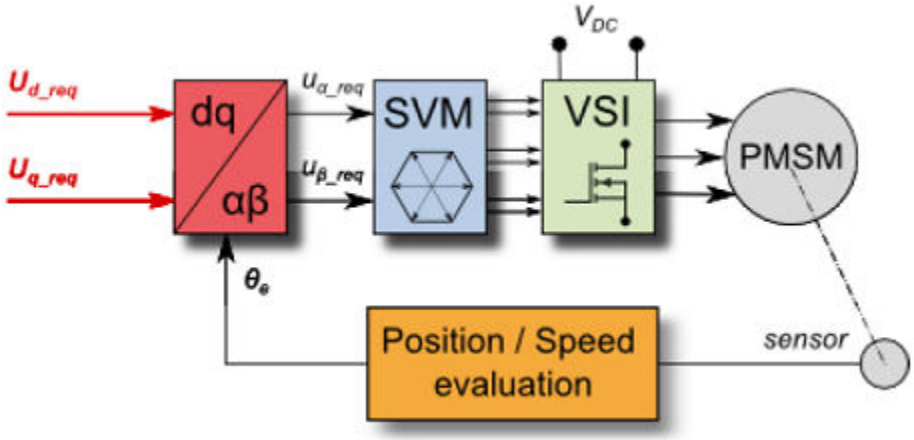

Figure 16. Voltage FOC control mode

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 (“Id_req” and “Iq_req”) available for the motor control, as shown in the block diagram in Figure 17. The d-axis current component isd_req is responsible for the rotor flux control. The q-axis current component of the current isq_req generates torque and, by its application, the motor starts running. By changing the polarity of the current isq_req, 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.

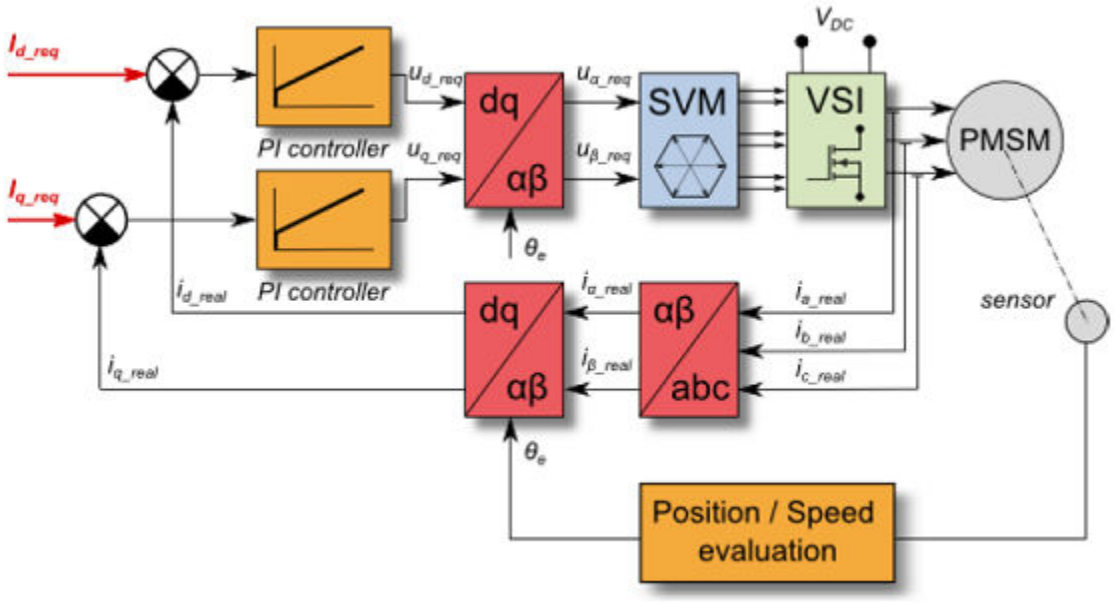

Figure 17. Current (torque) control mode

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 “Speed_req” field. The d-axis current reference is held at 0 during the entire FOC operation.

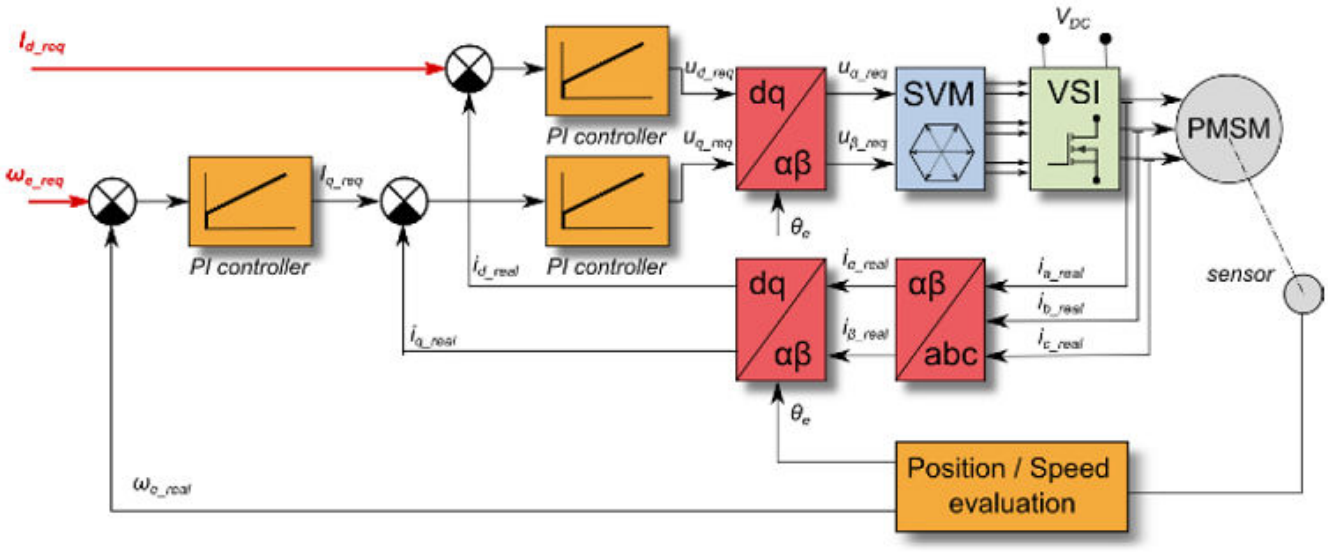

Figure 18. Speed FOC control mode

Application demo control—“App control” tab

After launching the application and performing all necessary settings, you can control the PMSM motor using the FreeMASTER application demo control page. This page contains:

- Speed gauge—shows the actual and required speeds.
- Required speed slider—sets up the required speed.
- DC-bus voltage—shows the actual DC-bus voltage.
- Current i_q—shows the actual torque-producing current.

- Current limitation—sets up the torque-producing current limit.
- Demo mode on/off button—turns the demonstration mode on/off.
- RUN/STOP PWM button—runs/stops the whole application (sets the PWM on and off).
- Notification—shows the notification about the actual application state (or faults).

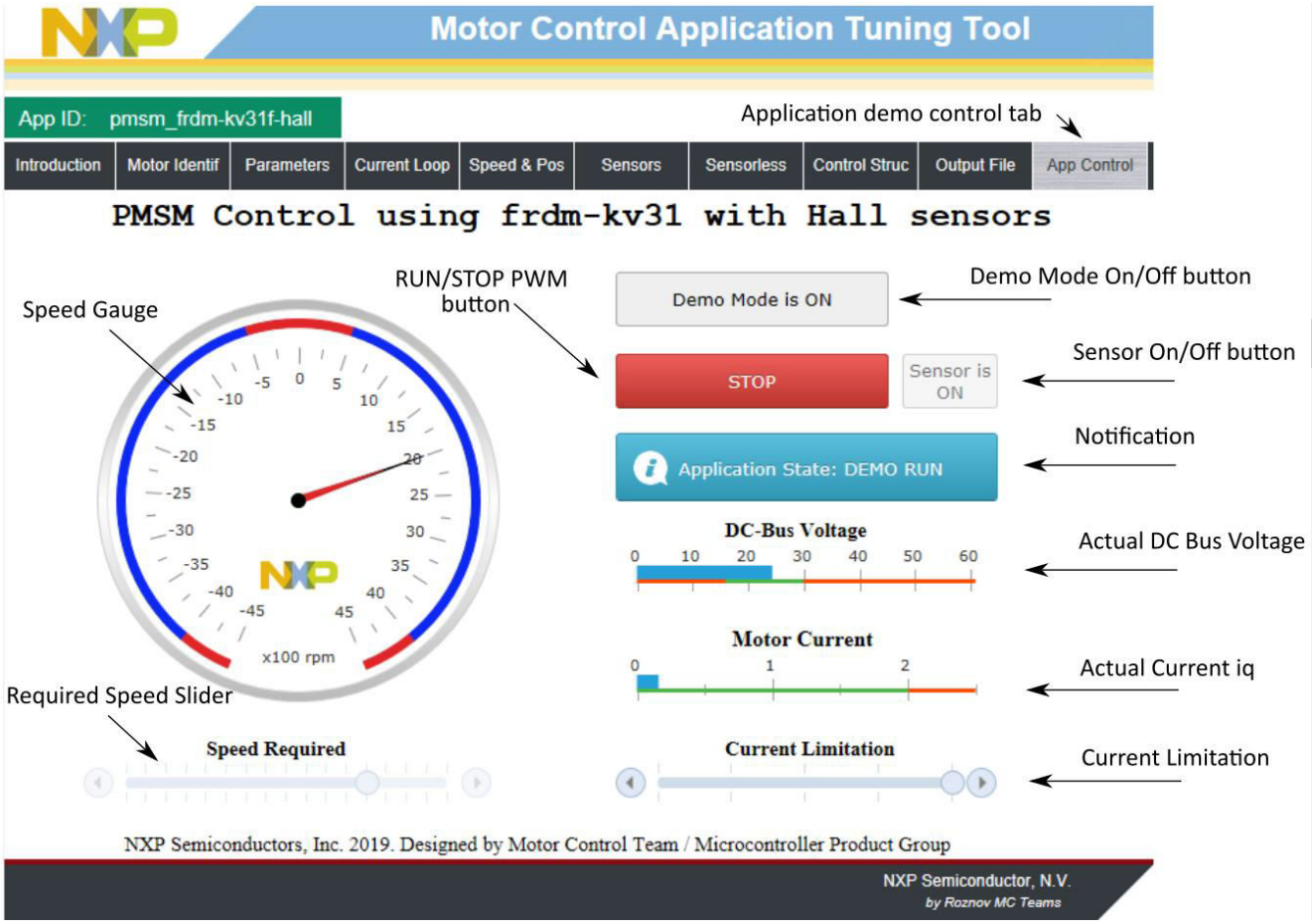

Figure 19. FreeMASTER control page

These are the basic instructions for controlling a motor:

- To start the motor, set the required speed using the speed slider.
- In case of a fault, click on the fault notification to clear the fault.
- Click the “Demo Mode On/Off” button to turn the demonstration mode on/off.
- Click the “RUN/STOP” button to stop the motor.

Chapter 9
Identifying parameters of user motor using MCAT

This section provides a guide on how to run your own motor or tune the default motor in several steps. It is highly recommended to go through all the steps carefully to eliminate any possible issues during the tuning process. The state diagram in Figure 20 shows a typical PMSM sensor/sensorless control tuning process.

Because the model-based control methods of the PMSM drives are the most effective and usable, 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 Rs, direct inductance Ld, quadrature inductance Lq, and BEMF constant Ke. If your connected PMSM motor is not the default Teknic or Linix motor described in the previous sections, identify the parameters of your motor first. Each tuning phase is described in more detail in the following sections.

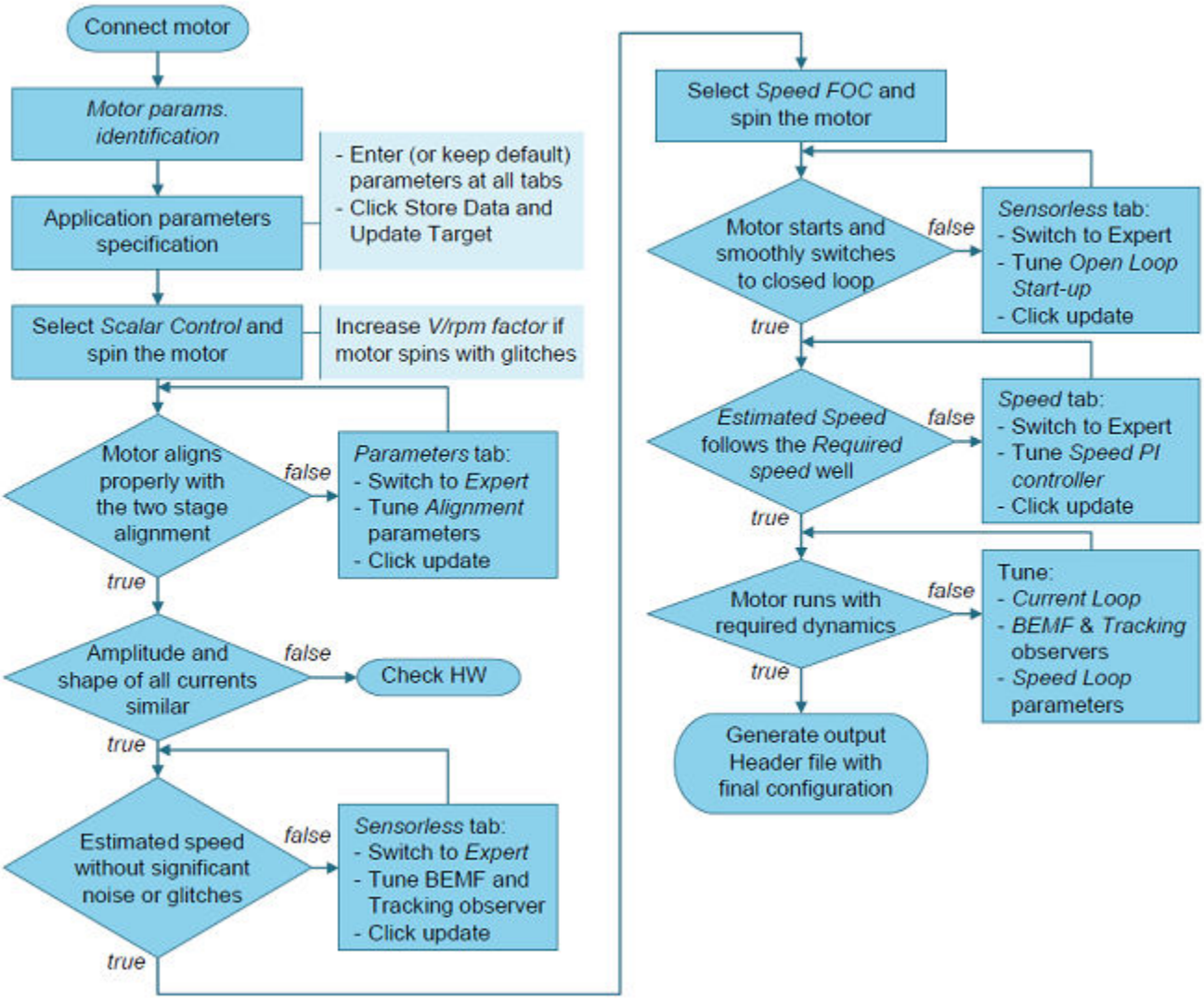

Figure 20. Running a new PMSM

Power stage characterization

Each inverter introduces the total error voltage U_{error}, which is caused by the dead time, current clamping effect, and transistor voltage drop. The total error voltage U_{error} depends on the phase current i_s and this dependency is measured during the power

stage characterization process. An example of the inverter error characteristic is shown in Figure 21. The power stage characterization is a part of the MCAT and it can be controlled from the “Motor Identif” tab. To perform the characterization, connect the motor with a known stator resistance R_s and enter this value into the “Calib Rs” field. Then specify the “Calibration Range”, which is the range of the stator current i_s, in which the measurement of U_{error} is performed. Start the characterization by pressing the “Calibrate” button. The characterization gradually performs 65 i_{sd} current steps (from $i_s = -I_{s,calib}$ to $i_s = I_{s,calib}$) with each taking 300 ms, so be aware that the process takes about 20 seconds and the motor must withstand this load. The acquired characterization data is saved to a file and used later for the phase voltage correction during the R_s measurement process. The following R_s measurement can be done with the I_s,calib maximum current. It is recommended to use a motor with a low R_s for characterization purposes.

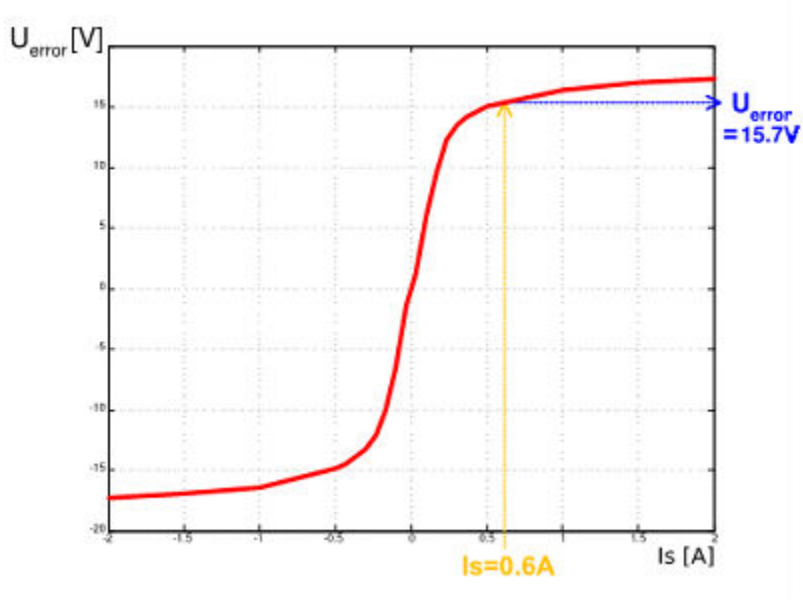

Figure 21. Example power stage characteristic

The power stage characterization is necessary only for the user hardware board. When the NXP power stages (TWR, FRDM, or HVP) are used with the application, the characterization process can be omitted. The acquired characterization data is saved into a file, so it is necessary to do it only once for a given hardware.

Stator resistance measurement

The stator resistance R_s is measured using the DC current I_{phN} value, which is applied to the motor for 1200 ms. The DC voltage U_{DC} is held using current controllers. Their parameters are selected conservatively to ensure stability. The stator resistance R_s is calculated using the Ohm’s law as:

$$R_s = \frac{U_{DC} - U_{error}}{I_{phN}} \ [\Omega]$$

Stator inductance

For the stator inductance (L_s) identification purposes, a sinusoidal measurement voltage is applied to the motor. During the L_S measurement, the voltage control is enabled. The frequency and amplitude of the sinusoidal voltage are obtained before the actual measurement, during the tuning process. The tuning process begins with a 0-V amplitude and the *F start* frequency, which are applied to the motor. The amplitude is gradually increased by *Ud inc* up to a half of the DC-bus voltage (DCbus/2), until *Id ampl* is reached. If *Id ampl* is not reached even with the DCbus/2 and *F start*, the frequency of the measuring signal is gradually decreased by *F dec* down to *F min* again, until *Id ampl* is reached. If *Id ampl* is still not reached, the measurement continues with DCbus/2 and *F min*. The tuning process is shown in Figure 22.

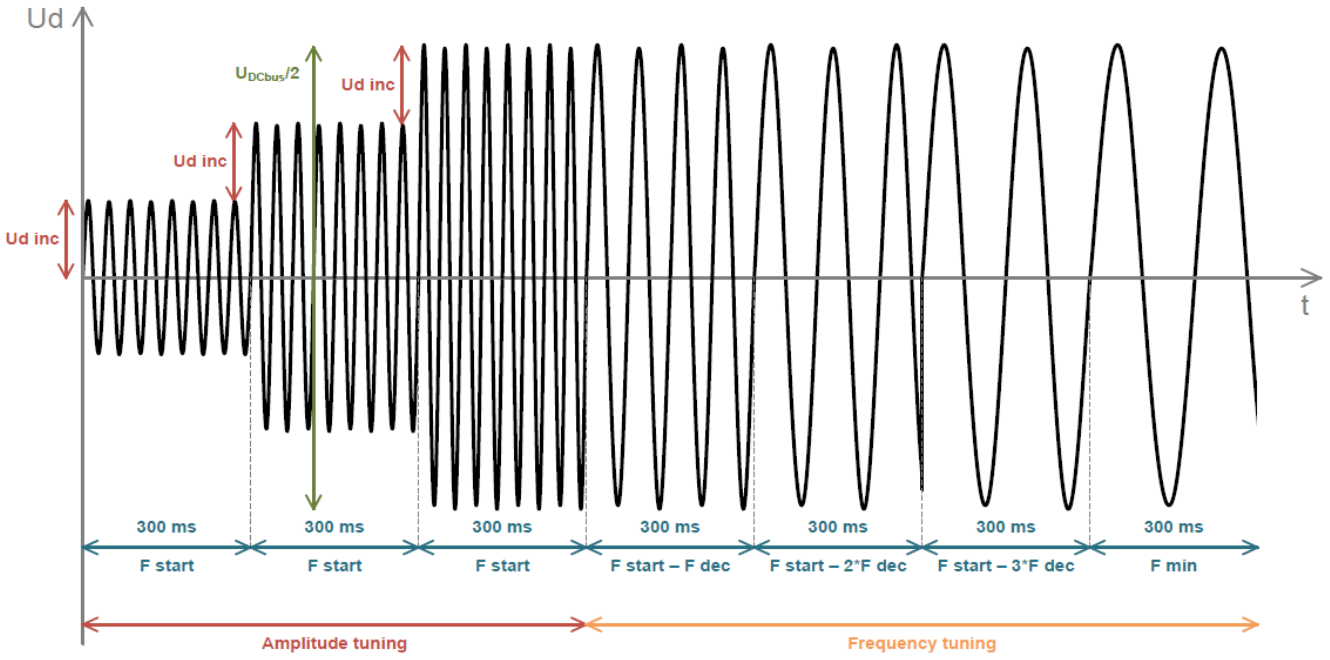

Figure 22. Tuning Ls measuring signal

When the tuning process is complete, the sinusoidal measurement signal (with the amplitude and frequency obtained during the tuning process) is applied to the motor. The total impedance of the RL circuit is then calculated from the voltage and current amplitudes and L_S is calculated from the total impedance of the RL circuit.

$$Z_{RL} = \frac{U_d}{I_{d\ ampl}}\ [\Omega]$$

$$X_{Ls} = \sqrt{Z_{RL}^2 - R_s^2}\ [\Omega]$$

$$L_s = \frac{X_{Ls}}{2\pi f}\ [\Omega]$$

The direct inductance L_d and quadrature inductance L_q measurements are made in the same way as L_S. Before the L_d and L_q measurement is made, DC current is applied to the D-axis, which aligns the rotor. For the L_d measurement, the sinusoidal voltage is applied in the D-axis. For the L_q measurement, the sinusoidal voltage is applied in the Q-axis.

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:

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

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.

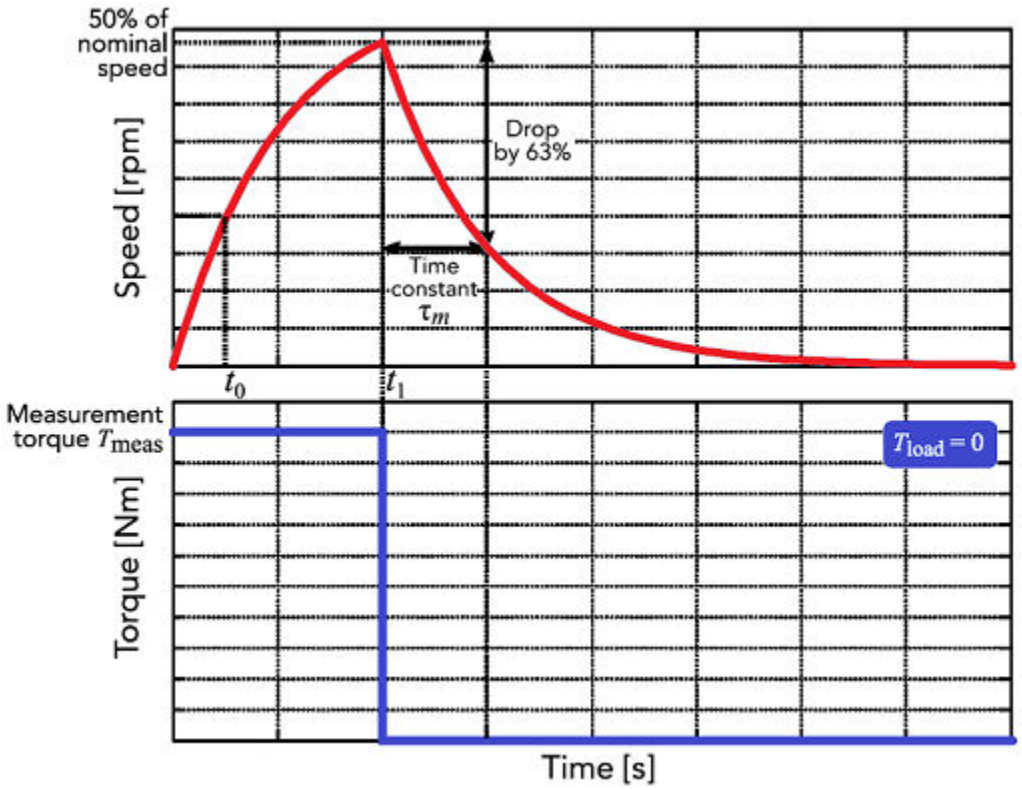

Figure 23. PMSM identification tab

9.1 PMSM electrical and mechanical parameters measurement process

The motor identification process can be controlled and set up in the MCAT “Motor Identif” tab, which is shown in Figure 24. To measure your own motor, follow these steps:

- Select your hardware board. Choose between the standard NXP hardware or use your own. If you use your own hardware, specify its scales (“I max” and “U DCB max” in the “Parameters” menu tab).
- If you don’t know the number of motor’s pole-pairs, use the number of pole-pair assistant and compute the number of motor rotor stops in one turn.
- If you use your own hardware for the first time, perform the power stage characterization.
- Enter the motor measurement parameters and start the measurement by pressing the “Measure electrical” or “Measure mechanical” buttons. You can observe which parameter is being measured in the “Status” bar.

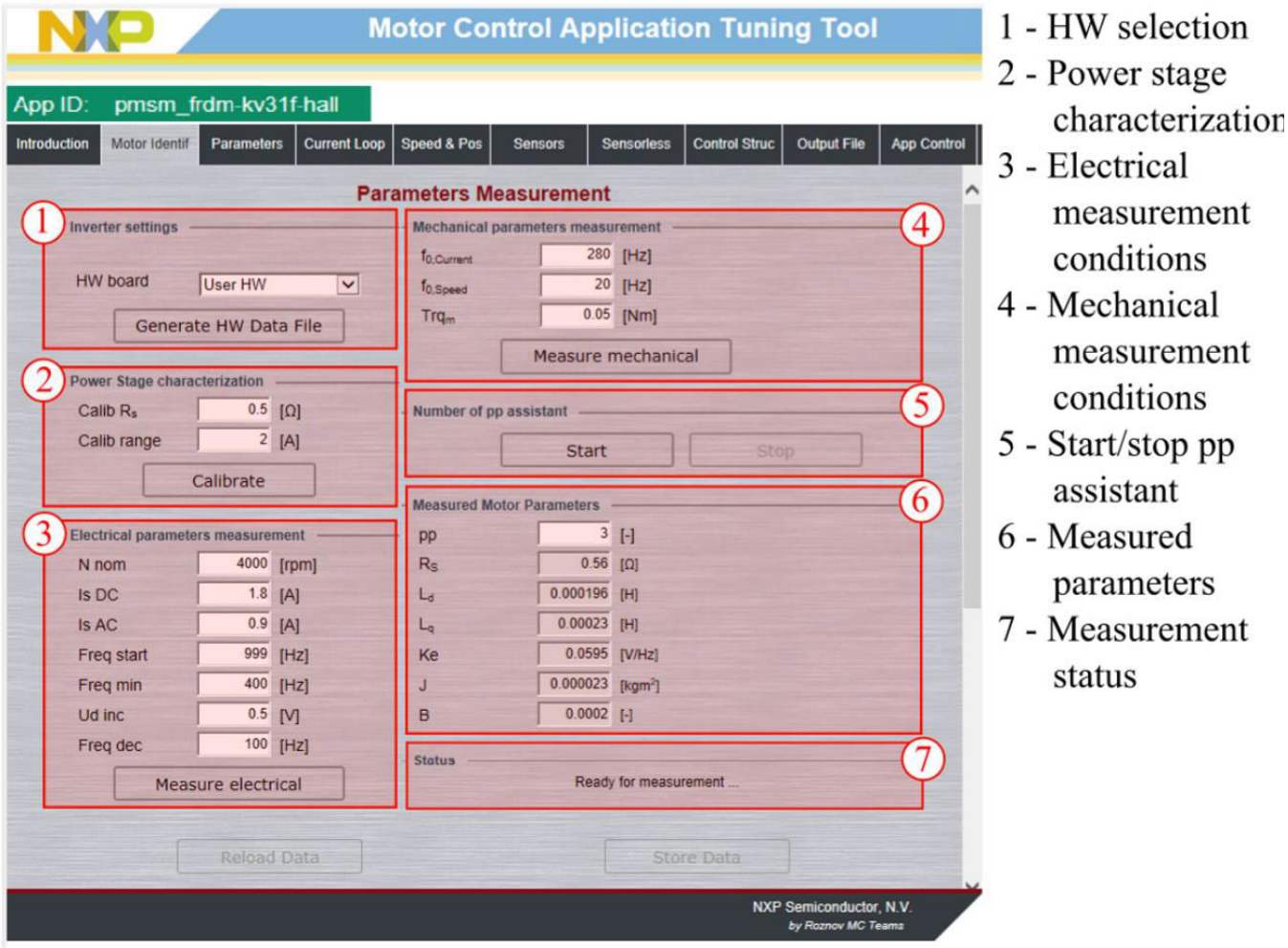

Figure 24. PMSM identification tab

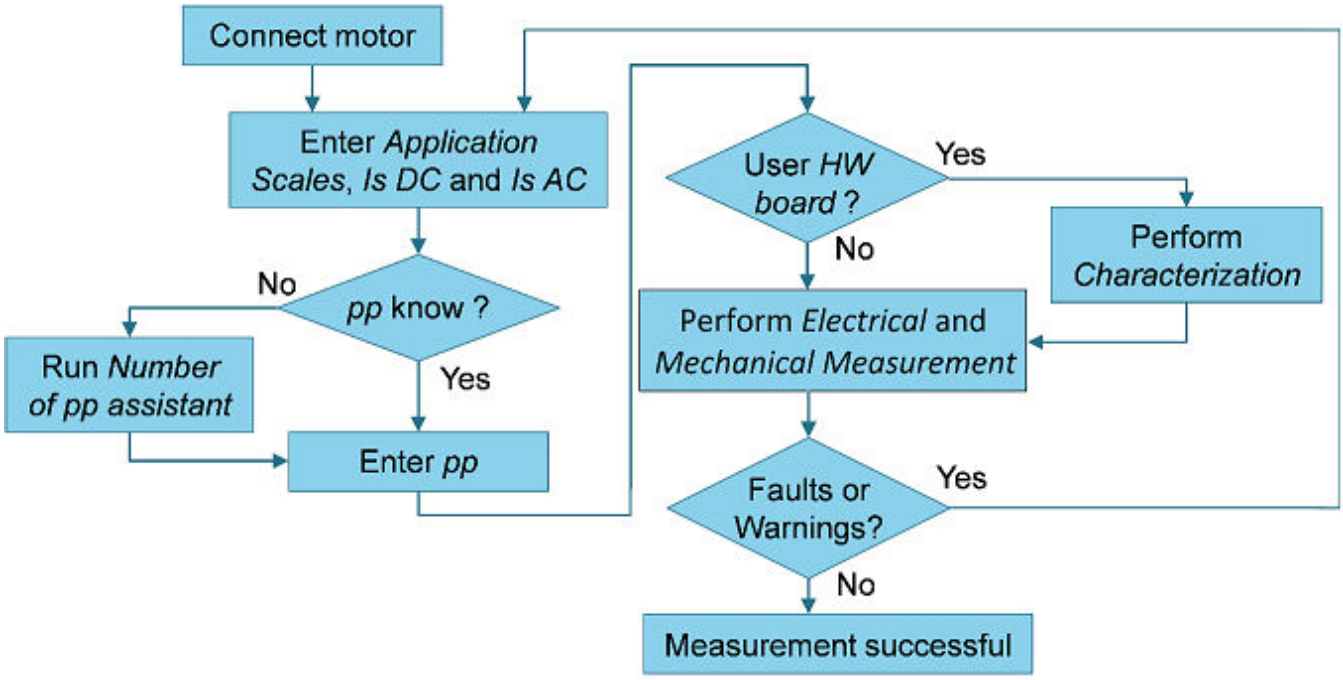

Figure 25. Measurement process diagram

During the measurement, faults and warnings may occur. Do not confuse these faults for the application faults, such as overcurrent, undervoltage, and so on. The list of these faults with their description and possible troubleshooting is shown in Table 6.

Table 6. Measurement faults and warnings

Fault no.	Fault description	Fault reason	Troubleshooting
1	Motor not connected	I_d > 50 mA cannot be reached with the available DC-bus voltage.	Check that the motor is connected.
2	R_s too high for calibration	The calibration cannot be reached with the available DC-bus voltage.	Use a motor with a lower R_s for the power stage characterization.
3	Current measurement I_s DC not reached	The user-defined I_s DC was not reached, so the measurement was taken with a lower I_s DC.	Raise the DC-bus voltage to reach the I_s DC or lower the I_s DC to avoid this warning.
4	Current amplitude measurement I_s AC not reached	The user-defined I_s AC was not reached, so the measurement was taken with a lower I_s AC.	Raise the DC-bus voltage or lower the F_{min} to reach the I_s AC or lower the I_s AC to avoid this warning.
5	Wrong characteristic data	The characteristic data, which is used for the voltage correction, does not correspond to the actual power stage.	Select the user hardware and perform the calibration.

Table continues on the next page...

Table 6. Measurement faults and warnings (continued)

Fault no.	Fault description	Fault reason	Troubleshooting
6	Mechanical measurement timeout	The mechanical measurement takes too long.	Repeat the measurement process with a different setup.

9.2 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 7 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 7. 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

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 not accessible in the “Basic” mode and are calculated using the motor parameters and hardware scales (see Table 8).

Table 8. 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

Table continues on the next page...

Table 8. Fault limits (continued)

Parameter	Units	Description	Typical range
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 not accessible in the “Basic” mode and are calculated using the motor parameters and hardware scales.

Table 9. Application scales

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

7. Check the alignment parameters—these fields are not accessible in the “Basic” mode and they 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.

9.3 Control structure modes

1. Select the scalar control by clicking the “DISABLED” button in the “Scalar Control” section. The button color changes to red and the text changes to “ENABLED”.
2. Turn the application switch on. The application state changes to “RUN”.
3. Set the required frequency value in the “Freq_req” field; for example, 15 Hz in the “Scalar Control” section. The motor starts running (Figure 26).

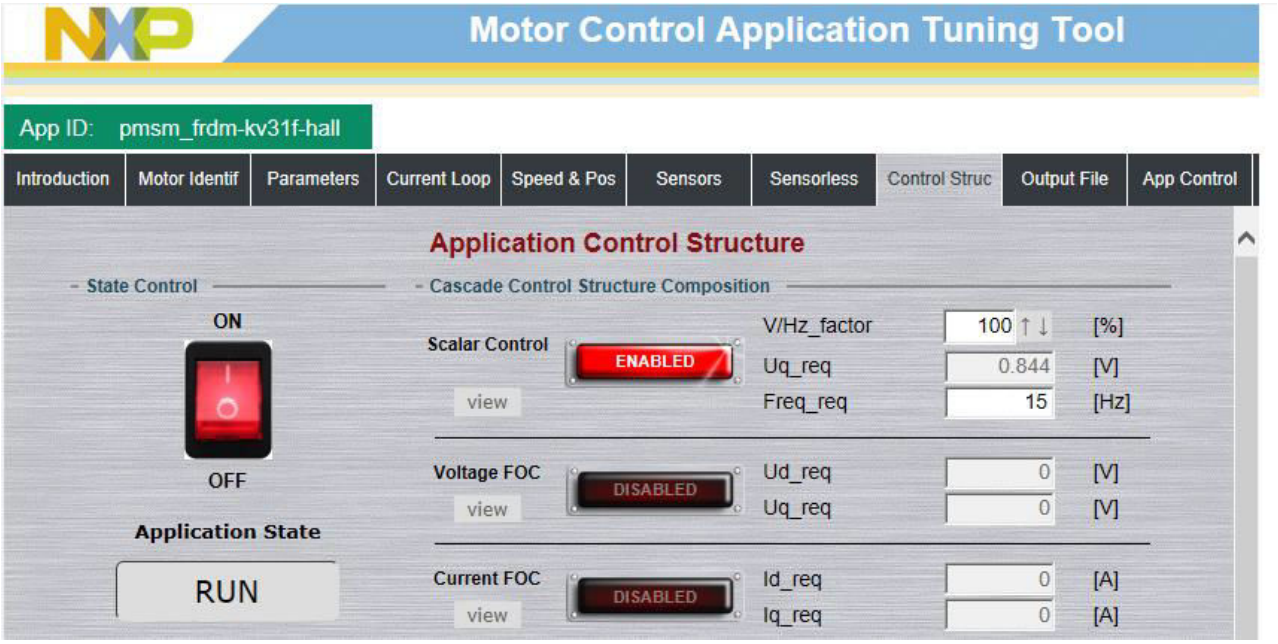

Figure 26. MCAT scalar control

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 or using the “UP/DOWN” buttons. The shape of the motor currents should be close to a sinusoidal shape (Figure 27).

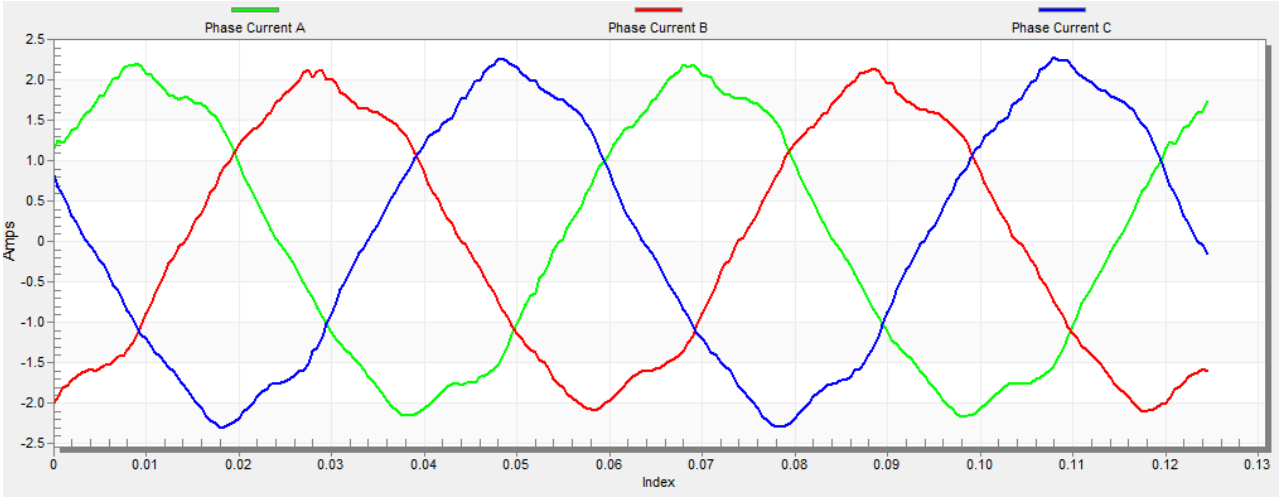

Figure 27. Phase currents

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 28) 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.

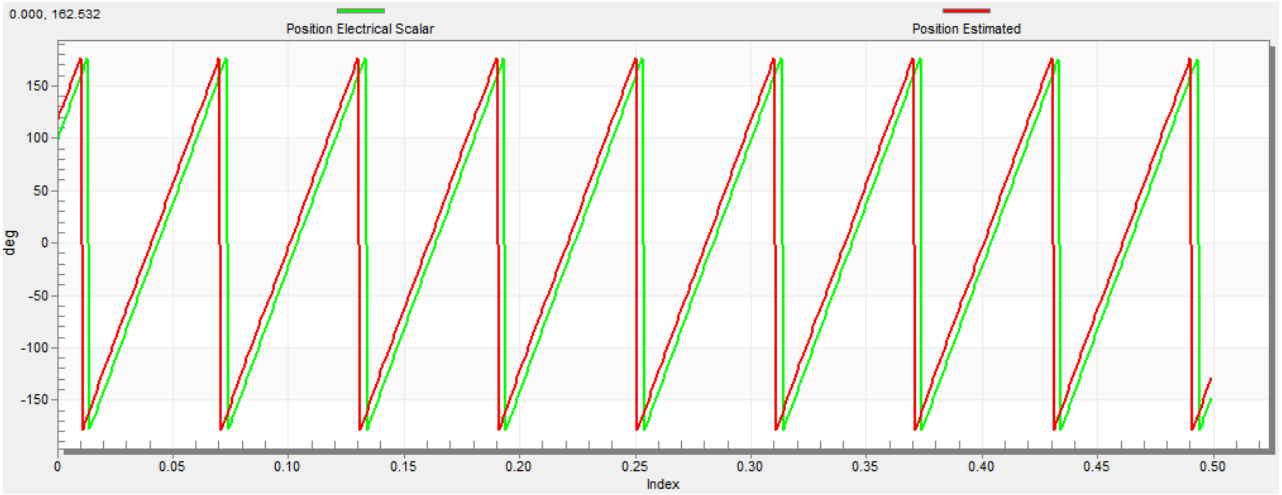

Figure 28. Generated and estimated positions

7. If an opposite speed direction is required, set a negative speed value into the “Freq_req” field.
8. The proper observer functionality and the measurement of analog quantities is expected at this step.
9. Enable the voltage FOC mode by clicking the “DISABLED” button in the “Voltage FOC” section while the main application switch is turned off.
10. Switch the main application switch on and set a non-zero value in the “Uq_req” field. The FOC algorithm uses the estimated position to run the motor.

9.4 Alignment tuning

For the alignment parameters, navigate to the “Tab” menu and select “Parameters”. The alignment procedure sets the rotor to an accurate initial position and enables you to apply full start-up torque to the motor. The rotor-alignment parameters are available for editing in the “Expert” mode. 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.

9.5 Current loop tuning

The parameters for the current D, Q, and PI controllers are fully calculated in the “Basic” mode 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 “Control Structure” tab, switch to “Current FOC”, set the “Iq_req” field to a very low value (for example 0.01), and set the required step in “Id_req”. 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 29).

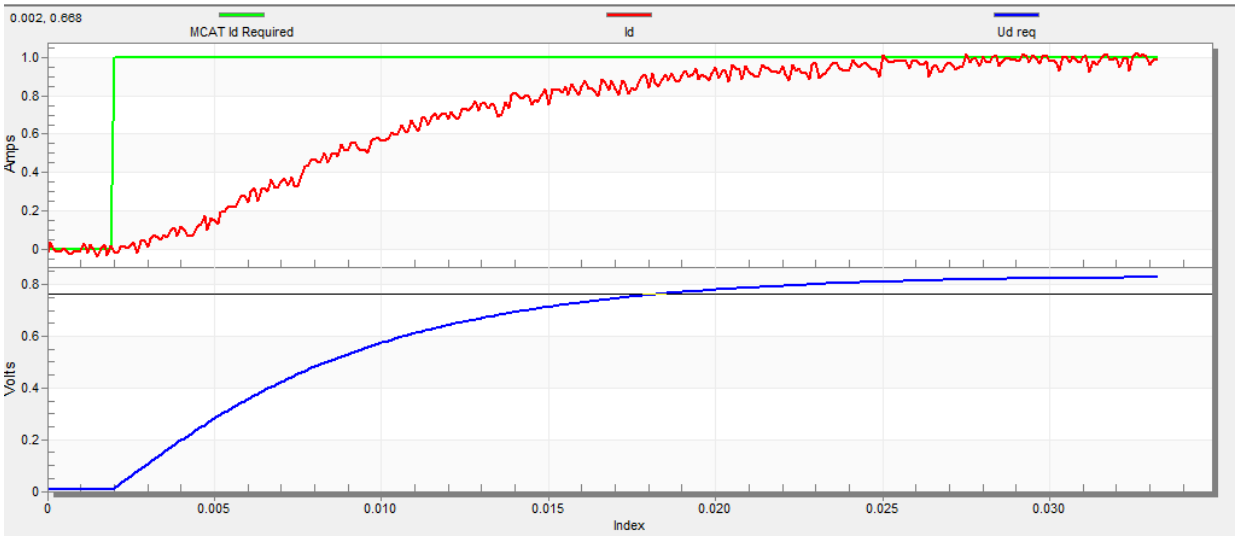

Figure 29. Slow step response of the Id current controller

   - The loop bandwidth (400 Hz) is optimal and the response time of the Id current is sufficient (see Figure 30).

Figure 30. 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 31).

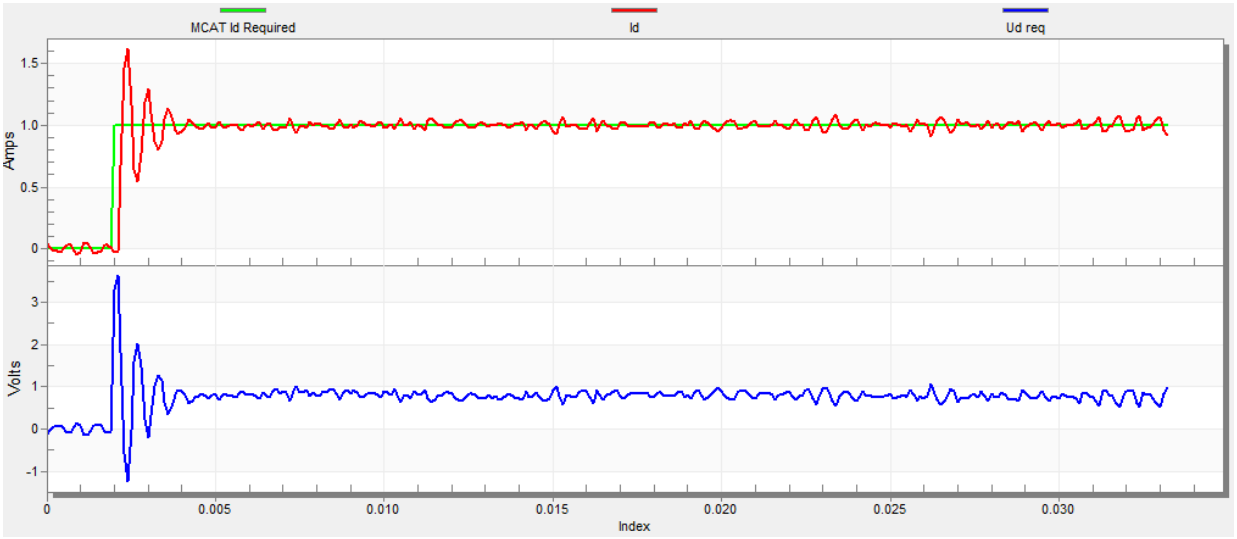

Figure 31. Fast step response of the Id current controller

9.6 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 & Pos” 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 32. 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.

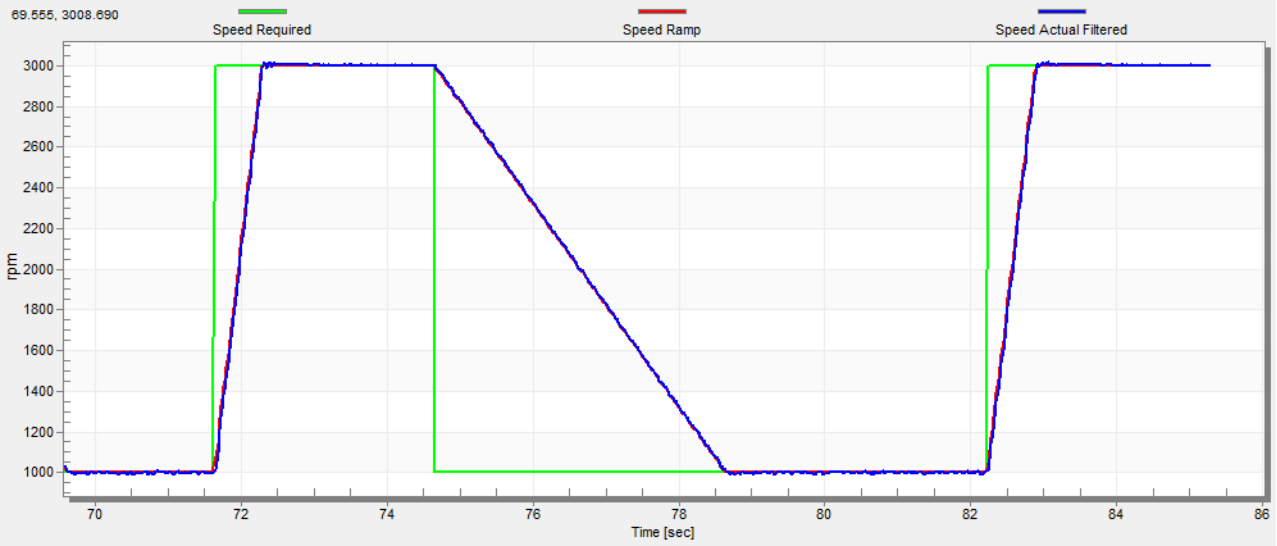

Figure 32. Speed profile

9.7 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 33.
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. Switch to the “Control Structure” tab, and enable the “Speed FOC”.
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 %.

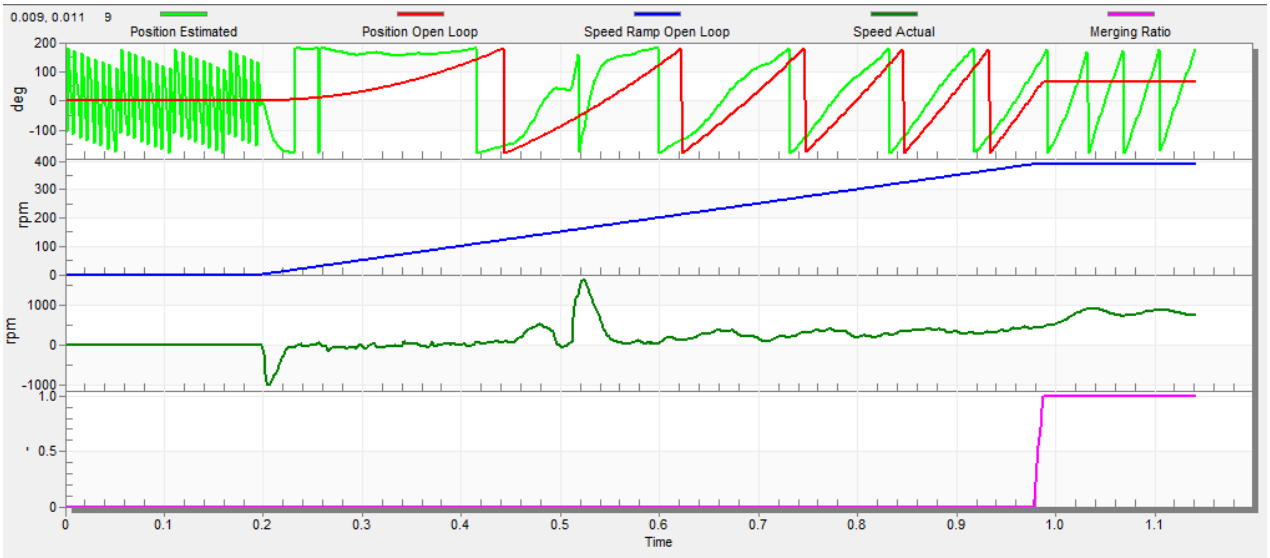

Figure 33. Motor startup

9.8 BEMF observer tuning

1. In the “Basic” mode, the parameters of the BEMF observer and the tracking observer are fully calculated using the motor parameters and no action is required. If the calculated loop parameters do not correspond to the optimal response, the bandwidth and attenuation parameters can be tuned.
2. Select the “Observer” recorder from the FreeMASTER project tree.
3. Set the required bandwidth and attenuation of the BEMF observer—the bandwidth is typically set to a value close to the current loop bandwidth.
4. 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).
5. Click the “Update Target” button to apply the changes to the MCU.
6. Check the observer response in the recorder.

9.9 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 34). 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 & Pos” 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).
   - Switch to the “Control Structure” tab and 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}.
- Switch back to the “Speed loop” tab.
- 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 34).

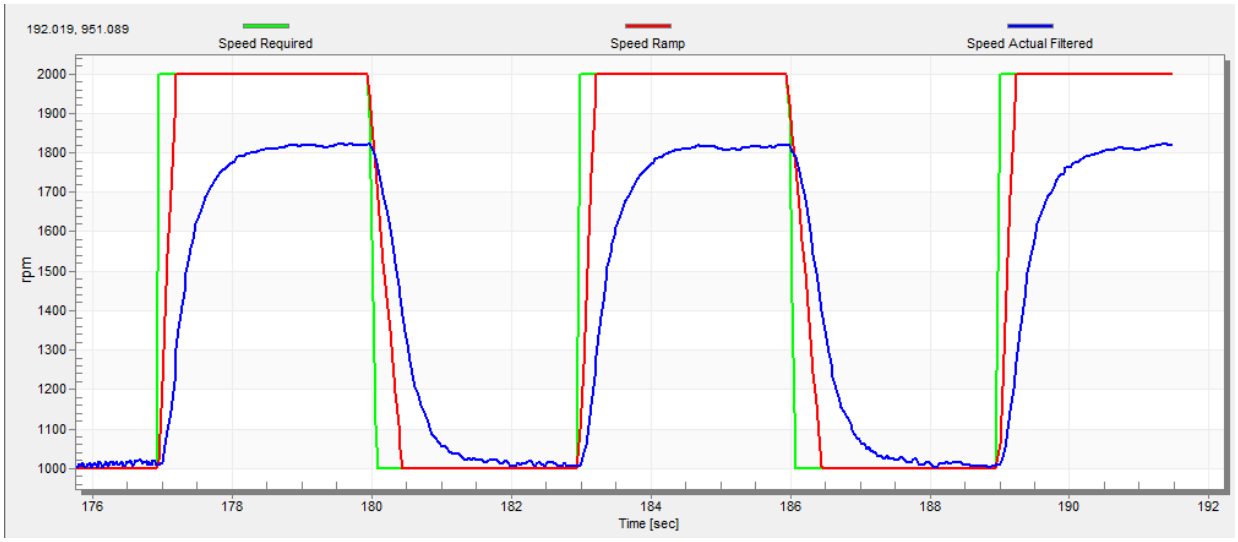

Figure 34. Speed controller response—SL_Ki value is low, Speed Ramp is not achieved

   - The “SL_Kp” value is low, the “Speed Actual Filtered” greatly overshoots, and the long settling time is unwanted (see Figure 35).

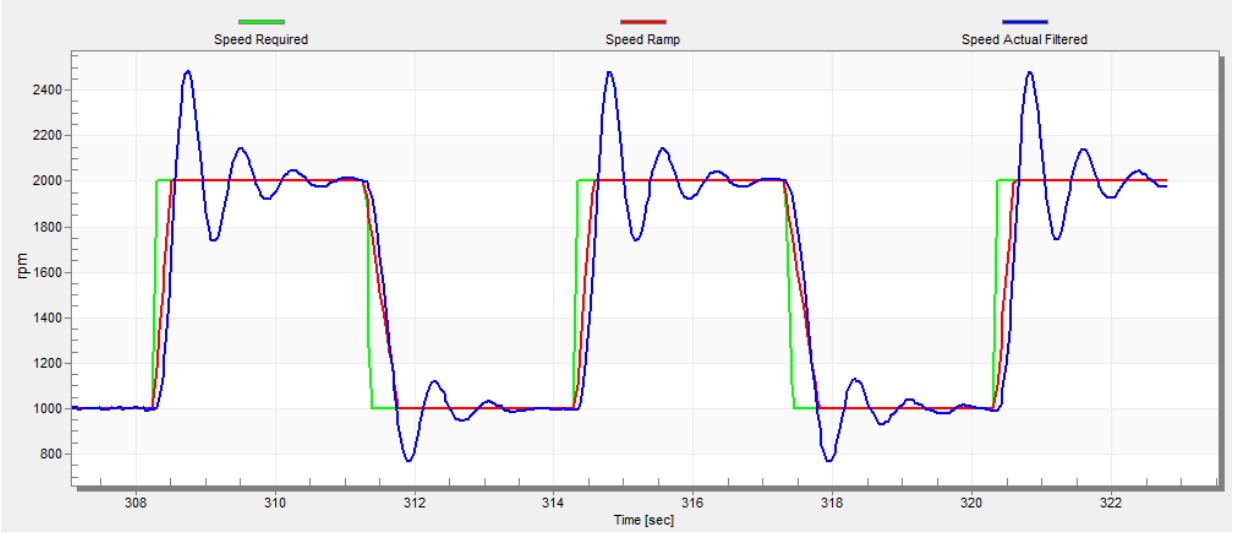

Figure 35. Speed controller response—SL_Kp value is low, Speed Actual Filtered greatly overshoots

- 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 36).

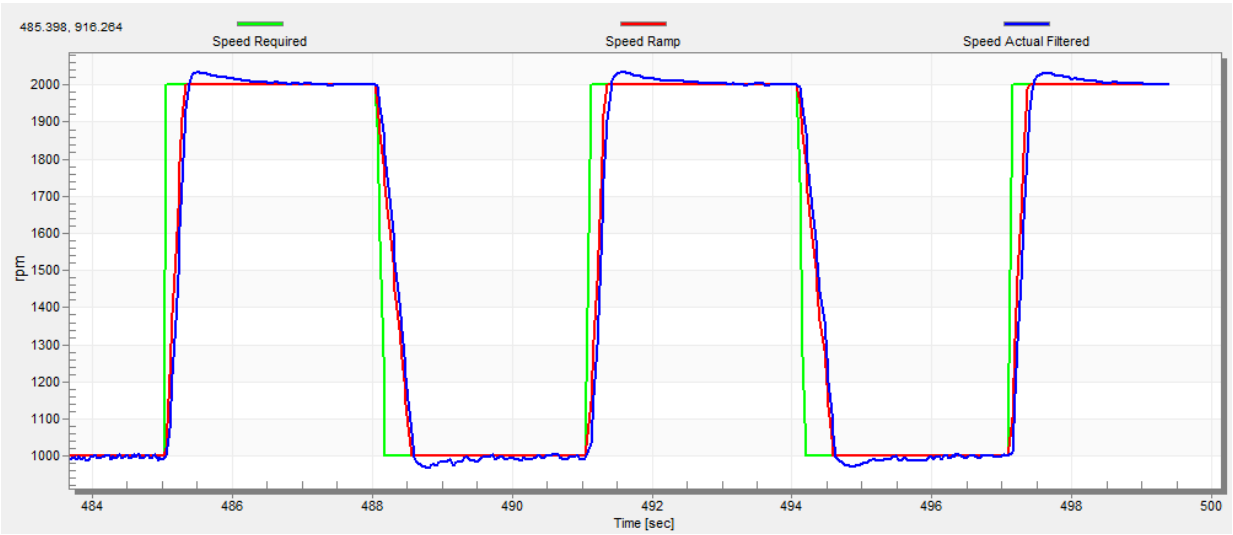

Figure 36. Speed controller response—speed loop response with a small overshoot

Chapter 10
Conclusion

This application note describes the implementation of the sensor and sensorless Field-Oriented Control of a 3-phase PMSM on the NXP LPC55S69 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 LPC55S69 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 using MCAT.

Chapter 11
Acronyms and abbreviations

Table 10.

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
LPUART	Low Power Universal Asynchronous Receiver/Transmitter
MCAT	Motor Control Application Tuning tool
MCDRV	Motor Control Peripheral Drivers
MCU	Microcontroller
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 12
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).

Chapter 13
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. LPC55S69-EVK
5. FRDM-MC-PMSM Freedome Development Platform
6. Get Started with the LPC55S69-EVK
7. SCTimer/PWM Cookbook (document AN11538)
8. MCUXpresso IDE - Importing MCUXpresso SDK
9. MCUXpresso SDK Builder (SDK examples in several IDEs) https://mcuxpresso.nxp.com/en/welcome

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

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

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

Date of release: 02/2020

Document identifier: 3PPMSMCLPCUG