AN12745

Sensorless PMSM Field-Oriented Control on DSC 56F837xx

Rev. 0 — 03/2020 Application Note

by: NXP Semiconductors

Contents

1 Introduction.....1
2 Development platforms.....1
3 MCU features and peripheral settings...3
4 Motor Control Peripheral Drivers (MCDRV).....8
5 Tuning and controlling the application.10
6 Conclusion.....29
7 Acronyms and abbreviations.....30
8 References.....30

1 Introduction

This application note describes the implementation of the sensorless motor control reference application software for a 3-phase Permanent Magnet Synchronous Motor (PMSM) on the MC56F837xx DSCs. The sensorless control software and the PMSM control theory in general are described in *Sensorless PMSM Field-Oriented Control* (document DRM148). The Freedom (FRDM-MC-LVPMSM) and High Voltage (HVP-MC3PH) power stages are used as hardware platforms 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 last part of this 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 are a simple and user-friendly way for algorithm tuning, software control, debugging, and diagnostics.

2 Development platforms

There are these two standard NXP power stages:

- FRDM-MC-LVPMSM
- HVP-MC3PH

2.1 FRDM-MC-LVPMSM

This evaluation board, in a shield form factor, effectively turns the NXP Freedom development board into a complete motor control reference design, compatible with the existing NXP Freedom development boards. The Freedom motor control headers are compatible with the Arduino™ R3 pin layout.

The FRDM-MC-LVPMSM has the power supply input voltage of 24-48 VDC with a reverse-polarity protection circuitry. An auxiliary power supply of 5.5 VDC supplies the FRDM MCU boards. The output current is up to 5 A RMS. The inverter is realized with a 3-phase bridge inverter (6 MOSFETs) and a 3-phase MOSFET gate driver. 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/position sensors (Encoder, Hall). The block diagram of a complete Freedom motor control development kit is in Figure 1.

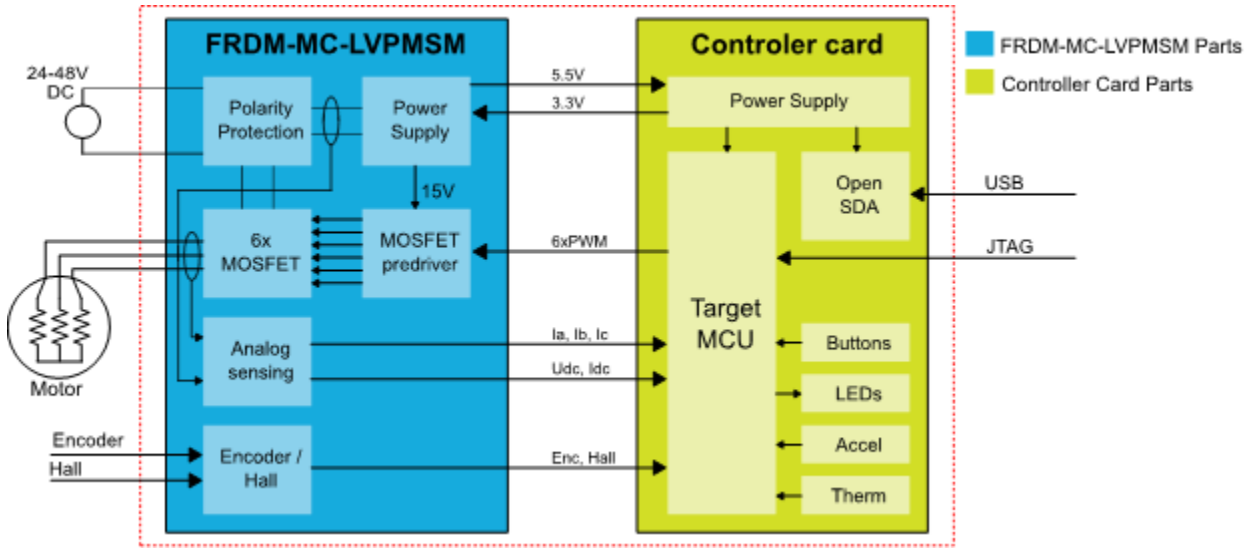

Figure 1. Freedom motor control development platform block diagram

The FRDM-MC-LVPMSM does not require any complicated setup and there is only one way to connect this shield board to the Freedom MCU board. See the *Freedom FRDM-MC-LVPMSM Development Platform User's Guide* (document FRDMLVPMSMUG) and the *PMSM sensorless application based on DSC MC56F837xx device* (document xxxx). For more information about the Freedom development platform, see nxp.com.

2.2 HVP-MC3PH

NXP's 3-phase high-voltage motor-control development platform (HVP) is a 115/230-V, 1-KW power stage that is an integral part of NXP's embedded motion-control series of development tools. It is supplied in the HVP-MC3PH kit. In combination with an HVP daughter card, it provides a ready-made software development platform for more than 1-horsepower motors. The block diagram of a complete high-voltage motor-control development kit is in Figure 2.

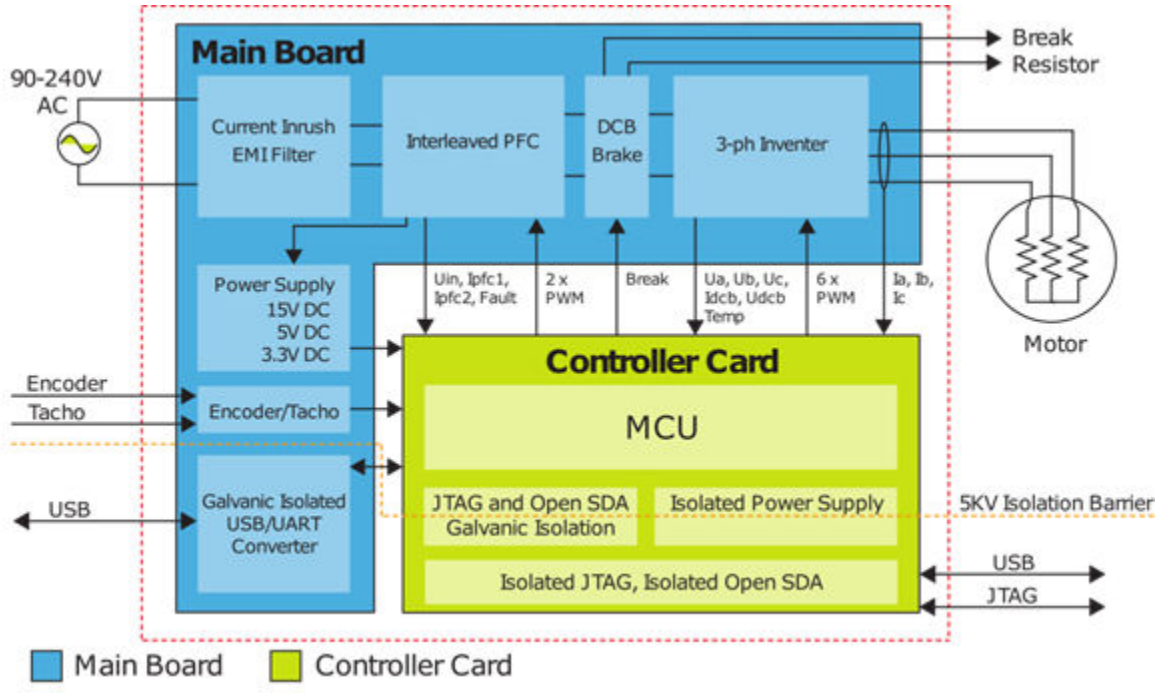

Figure 2. High-voltage motor-control development platform block diagram

The HVP-MC3PH power stage does not require a complicated setup and there is only one way to connect a daughter card to the HVP. It is strongly recommended to read the complete *Freescale High-Voltage Motor Control Platform User's Guide* (document HVPMC3PHUG). Due to high voltage, the HVP platform may represent a safety risk when not handled correctly. For more information about NXP's high-voltage motor-control development platform, see www.nxp.com.

3 MCU features and peripheral settings

This chapter describes the hardware-dependent part of code for all supported MCUs, which includes the peripheral initialization and explanation of the application timing.

NOTE

Both the high-voltage and low-voltage PMSM sensorless applications use the same peripheral settings, except for the dead time setting, hardware over-current fault setting, and other insignificant differences. The peripheral settings are described only for the high-voltage platform, but the differences in peripheral settings between the high-voltage and low-voltage platforms are listed in the related sections.

3.1 DSC MC56F837xx family

MC56F837xx is a low-power DSC MCU family, offering outstanding power consumption at run time in a compact 5 x 5 mm package with exceptional performance, precision, and control for high-efficiency digital power conversion and advanced motor control (MC56F837xx) applications. MC56F837xx includes advanced high-speed and high-accuracy peripherals, such as high-resolution Pulse Width Modulation (PWM) with 312-picosecond resolution, dual high-speed 12-bit Analog-to-Digital Converters (ADCs) with built-in PGA sampling of up to 1.25 Mega Samples Per Second (MSPS) at 12 bits. Faster application-specific control loops are driven via a 32-bit DSP core with single-cycle math computation, fractional arithmetic support, and parallel moves. For more information, see the *MC56F83xxx Reference Manual* (document MC56F83XXXRM).

The peripherals (whose setup is detailed later on in this chapter) used by the PMSM motor-control software on MC56F837xx are:

- 12-bit cyclic Analog-to-Digital Converter (ADC12) for the phase currents, DC-bus voltage, and IPM temperature measurement.

- eFlexPWM module (PWM) for the 6-channel PWM generation.
- Periodic Interrupt Timer (PIT) for the slow control loop timing.
- XBARA multiplexer for the over-current fault and ADC12 trigger routing.
- Serial Interface (QSCI0) for the FreeMASTER communication.
- General-Purpose Input/Output (GPIO) pins for the inrush relay and brake circuit control.

The application timing diagram is shown in Figure 3. All tasks are handled using these interrupt service routines:

- ADC_EOS_isr ()—level-two priority interrupt, triggered when the conversion of all enabled samples is completed by the ADCA. It handles the fast control loop of the FOC and the FreeMASTER recorder feature.
- PIT_ISR ()—level-one priority interrupt triggered by the PIT overflow. It handles the slow control loop of the FOC.

The fast and slow control loop ISRs are described in more detail in *Sensorless PMSM Field-Oriented Control* (document DRM148).

3.2 DSC56800EX Quick Start

The MCU pins, clocks, and used peripherals are configured using the graphical configuration tool, which is a component of DSC56800EX Quick Start tool. The MCU configuration is saved in the *appconfig.h* file that is included in the application source code. For more information, see the *DSC56800EX Quick Start User's Guide* (document DSC56800EXQSUG).

3.3 Hardware timing and synchronization

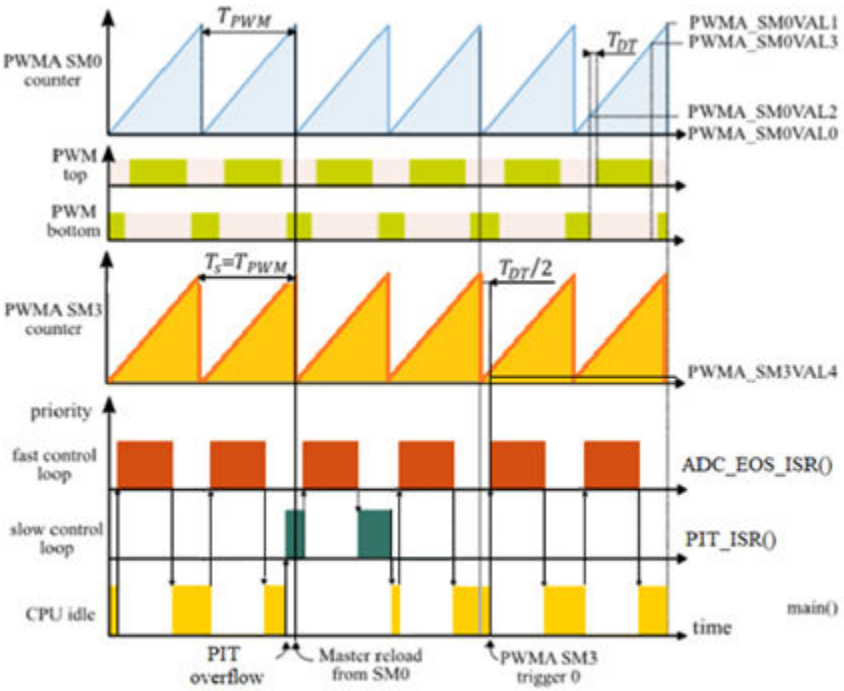

Figure 3. Example of application timing on MC56F837xx

The PWM sub-module 0 (SM0) timer internal counter counts from the PWM_SM0VAL0 value to the PWM_SM0VAL1 value with the *TPWM* period. The switching of the transistors on each motor phase is determined by the PWMA_SM[0..2]VAL2 and PWMA_SM[0..2]VAL3 register pair on PWMA SM0, SM1, and SM2. The dead time, which delays the rising edge of the transistor control signals by *TDT*, is inserted to avoid short circuit on the DC-bus.

The selection of the PWM switching frequency affects the switching power losses (a lower frequency is better) and audible noise (a higher frequency is better). This reference solution offers the possibility to easily increase the ratio between the FOC sampling period *Ts* and the PWM period *TPWM* (see MCDRV initialization and configuration"). The example in Figure 2 shows the case when *Ts* equals *TPWM*.

The ADCA and ADCB (because both ADC12s run in the triggered parallel mode) are triggered by the PWM SM0 trigger 0 signal, which is connected to the ADC via XBARA. The trigger is issued when the SM0 internal counter reaches the PWMA_SM0VAL4 value, which is set to *TDTI*/2 by default (this value ensures correct ADC sampling even at a very high duty cycle). The internal counter of SM0 is reloaded by the master reload trigger event from SM0. ADC12 converts a total of four samples at the beginning of sampling period *Ts*:

- The first two samples on the ADCA (channel 1 for phase A or 6 for phase C) and ADCB (channel 2 for phase B or 7 for phase C) are the samples of phase currents.
- The DC-bus voltage is sampled second by the ADCA channel 3.
- The IPM temperature is sampled second by the ADCB channel 0.

When all the samples are converted, the processing of the ADC_EOS_isr () high-priority ISR starts.

The CPU load and memory usage for the PMSM FOC reference software (see the *DSC PMSM Control Reference Application Package User's Guide*(document xxxx) for more details) is shown in Table 1. The results apply to the application built using the CodeWarrior 11.x IDE with the maximum speed optimization. The memory usage is calculated from the linker *.map* file, including the 2-KB FreeMASTER recorder buffer (allocated in RAM).

Table 1. MC56F837xx CPU and memory usage

-	MC56F837xx
CPU clock [MHz]	100
Fast Control Loop (%)	30,86
Slow Control Loop (%)	0,54
Total CPU load [%]	31,4
Flash usage [B]	23644
RAM usage [B]	5068

3.3.1 On-Chip Clock Synthesis (OCCS)

The MC56F837xx DSC uses the OCCS and SIM modules to configure and distribute the clock across the peripheral modules. The OCCS module provides several clock-source options for the MCU. The System Integration Module (SIM) provides system control and chip configuration. The OCCS module configuration is as follows:

- The 8-MHz clock from the 48-MHz/6 internal oscillator is used as the reference clock source.
- The PLL is used to generate the 100-MHz OCCS core output clock.
- The peripheral clock frequency is set to 100 MHz.

3.3.2 Periodic Interrupt Timer (PIT)

The PIT peripheral module is used for the slow control loop timing. The PIT module is configured as follows:

- The input clock is set to 195,312 kHz (1/512 of the peripheral clock frequency).
- The interrupt with a level-one priority is enabled on the counter reaching the modulo value (195).
- The modulo is set so that the overflow interrupt occurs at the slow control loop period (1,00352 ms).

3.3.3 12-bit cyclic Analog-to-Digital Converter (ADC12)

The ADC12 module is used to measure the phase currents, DC-bus voltage, the IPM temperature (a total of four samples are taken each sampling period). It consists of two converters (ADCA and ADCB).

The ADC12 module is configured as follows:

- The input clock is set to 20 MHz (1/5 of the peripheral clock frequency).
- The end-of-scan interrupt with a level 1 priority is enabled on the ADCA.
- The single-ended, 12-bit conversion with the hardware trigger from the PWMA is selected. The triggered parallel conversion is used on both ADCA and ADCB.
- Only the SAMPLE0, SAMPLE1, SAMPLE8, and SAMPLE9 samples are enabled.

3.3.4 Pulse Width Modulator A (PWMA)

The first three sub-modules of the eFlexPWM periphery PWMA are used to generate the 6-phase PWM for motor control with this setup:

- The input clock is set to *fPWMin* =100 MHz (fast peripheral clock frequency).
- The output PWM frequency is set to *fPWM =1/TPWM* =10 kHz . The PWMA_SM[0..2]INIT and PWMA_SM[0..2]VAL1 registers are used to define the PWM period and the PWMA_SM[0..2]VAL2 and PWMA_SM[0..2]VAL3 registers specify the current duty cycle.
- The counters at SM1 and SM2 are synchronized with the master sync signal from sub-module 0.
- The center-aligned, complementary PWM is generated only with a full cycle reload.
- A dead time of *TDT* =1.5 µs is inserted. This value is recommended by the manufacturer of the IPM used on the HVP-MC3PH board. The dead time counter modulo is set to PWMA_SM[0..2]DTCNT0 = PWMA_SM[0..2]DTCNT1 = *TDTfPWMin* =150.
- Channels A and B at SM0, SM1, and SM2 are disabled on faults number 0 active with automatic clearing (the PWM outputs are re-enabled at the first PWM reload after the fault disappears). Faults number 0 (connected to the IPM fault pin via GPIO, active in low) are enabled.

Sub-module 0 VAL4 is used for the ADC12 triggering with this setup:

- The trigger is issued when the PWMA_SM0VAL4 value is reached (*TDT*/2 by default).

The eFlexPWM module is a dedicated peripheral enabling the generation of 3-phase PWM signals connected to the IPM H-bridge driver. The three PWM submodules used in the application are configured using the Graphical Configuration Tool (GCT), as listed here:

- PWM_0:
  - IPBus clock source of 100 MHz.
  - Running frequency of 10 kHz with a 100-µs period.
  - INIT register—5000, VAL1 4999—13-bit resolution.
  - Complementary mode with 1.5-µs dead time.
  - PWM reload and synchronization signals, generated at every opportunity from this module.
  - Trigger 4, enabled to provide synchronization with the ADC module via XBAR.
  - High-side and low-side PWM_A and PWM_B outputs in positive (active high) polarity.
- PWM_1 and PWM_2:
  - PWM_0 clock source.
  - Running frequency of 10 kHz with 100-µs period.
  - INIT register—5000, VAL1 4999—13-bit resolution.

  - Complementary mode with 1.5-µs dead time.
  - PWM reload and synchronization signals, generated at every opportunity from this module.
  - High-side and low-side PWM_A and PWM_B outputs in positive (active high) polarity.
- PWM FAULT:
  - The fault 0 pin with a low fault level is connected via XBAR to a fixed 10,5-A hardware over-current protection of the power module.
  - The fault 1 pin with a low fault level is connected via XBAR to a comparator output that creates adjustable over-current protection. The comparator compares the DC-bus current with the adjustable DAC level.
  - The fault input filter is disabled.

Differences between the high-voltage platform and the low-voltage PMSM platform:

- The low-voltage PMSM platform uses a dead time of 0,5 µs (DTCNT0 = DTCNT1 = *TDTfPWMin* = 50).
- The low-voltage PMSM platform does not use a fixed over hardware over-current protection, but only adjustable over-current protection formed by the comparator that compares the DC-bus current with the adjustable level of the DAC. The comparator output is routed to the fault 0 pin with a low fault level.

3.3.5 Inter-Peripheral Crossbar Switch A (XBARA)

The XBARA module is used to route the over-current fault signal from the CMP1 and the IPM fault pin to the PWMA and to route the trigger signal from the PWMA to the ADC12. The XBARA module is set up as follows:

- The XBARA input IN6 (GPIO_C16/GPIO_F0) is connected to output OUT29 (PWMA_FAULT0).
- The XBARA input IN14 (CMPC_OUT) is connected to output OUT30 (PWMA_FAULT1).
- The XBARA input IN20 (PWMA0_MUX_TRIG0/PWMB0_OUT_TRIG0 signal) is connected to output OUT12 (ADCA_TRIG).

The differences between the high-voltage platform and the low-voltage PMSM platform are as follows:

- The low-voltage PMSM platform uses only adjustable over-current protection routed to the fault 0 pin via the XBAR as follows:
  - The XBARA input IN14 (CMPC_OUT) is connected to output OUT29 (PWMA_FAULT0).
  - The XBARA output OUT30 is not used.

3.3.6 High-Speed Comparator C (HSCMP_C)

The high-speed comparator module is used for adjustable over-current fault detection. The output of comparator C is connected to the PWMA fault 1 input. The HSCMP_C module is configured as follows:

- The negative comparator input is connected to the CMPC_IN0 (GPIO_B3) - DC Bus current signal.
- The positive comparator input is connected to the 8-bit DAC. The DAC reference output voltage level can be set so that the over-current fault is activated when the DC-bus current ranges from 0,063 A to 7,937 A with a resolution of 63 mA.
- The comparator output polarity is set to normal. The output high happens when the positive input is higher than the negative input.

The differences between the high-voltage platform and the low-voltage PMSM platform are as follows:

- The comparator C output is connected to the PWMA fault 0 input.
- The negative comparator input is connected to CMPC_IN3 (GPIO_B2).
- Because of a slightly different current scale, the over-current fault is activated when the DC-bus current ranges from 0,065 A to 8,185 A with a resolution of 65 mA.

3.3.7 Universal Asynchronous Receiver and Transmitter (SCI0)

The SCI0 module is used for the FreeMASTER communication between the MCU board and the PC. The module configuration is as follows:

- The baud rate is set to 19,200 Bd.
- Both the receiver and the transmitter are enabled.
- The other settings are set to default.

3.3.8 General-Purpose Input/Output (GPIO)

These GPIO pins are used:

- Inrush relay control on GPIOF7.
- Braking circuit control on GPIOF6.
- LED state indication on GPIOC0.

NOTE

The low-voltage PMSM application does not use the inrush relay control and the braking circuit control.

4 Motor Control Peripheral Drivers (MCDRV)

The MCDRV represent a simple way of peripheral initialization and access for the purposes of 3-phase ACIM or PMSM control. The features provided by the MCDRV library are the 3-phase PWM generation and 3-phase current measurements, as well as the measurements of the DC-bus voltage and the IPM temperature (or one general user-defined auxiliary quantity). The principles of the 3-phase current measurement and PWM generation using the Space Vector Modulation (SVM) technique are described in *Sensorless PMSM Field-Oriented Control* (document DRM148).

The MCDRV are divided into two parts:

- The first part is the peripheral initialization module consisting of *mcdrv_<platform>-<device>.c* and *mcdrv_<platform>-<device>.h* files, which is unique for each supported device. The header file includes all MCDRV setup options including the ADC channel assignment. The source file then contains functions for the initialization of all peripherals used for motor control. This module is more closely described in MCDRV initialization.
- The second part consists of peripheral driver library modules for each supported periphery. Generally, all the ADC and PWM periphery drivers share the same API within its class. This makes the higher-level code platform independent, because the peripheral driver function calls were replaced by universally named macros. The list of supported peripherals and the API of their drivers is in MCDRV API.

4.1 MCDRV initialization

The MCDRV initialization module consists of MCU peripheral initialization functions, as well as all the definitions that can be specified by the user. The functions are in the device-specific *mcdrv_<platform>-<device>.c* source file and *mcdrv_<platform>-<device>.h* header file. Out of all functions in the MCDRV initialization module, it is only necessary to call the *MCDRV_Init_M1*() function during MCU startup and before any other MCDRV functions. All peripherals used by a given device for motor-control purposes are initialized within this function.

The *mcdrv_<platform>-<device>.h* header file offers 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_MODULO – the value of this definition sets the PWM frequency. The value in this macro can be calculated as M1_PWM_MODULO = *f*PWMin/ *f*PWM, where *f*PWMin is the input frequency of the PWM periphery timer (see the section

about your peripheral initialization on your device for the default value set by the MCDRV) and *f*PWM is the desired PWM frequency.

- M1_FOC_PERIOD – this macro allows you to call the fast loop interrupt every 1st, 2nd, 3rd, ar nth PWM reload. This is convenient when the PWM frequency must be higher than the maximal fast loop interrupt.
- M1_PWM_PAIR_PH[A..C] – these macros allow for a simple assignment of physical motor phases to the PWM periphery channels or sub-modules. The order of motor phases can be altered this way.
- ADC_NO_CHAN – this macro is used to specify the unassigned ADC channel. It is recommended not to change this number.
- M1_ADC[0,1]_PH_[A..C] – these macros assign the ADC channels for the phase current measurement. The general rule is that at least one phase current must be measurable on both ADC converters and the remaining two 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 on the 3-phase current measurement, see *Sensorless PMSM Field-Oriented Control* (document DRM148). The unassigned ADC channels are set to the ADC_NO_CHAN value.
- M1_ADC[0,1]_UDCB and M1_ADC[0,1]_AUX – these defines are used to select the ADC channel for the measurement of the DC-bus voltage and one user-defined auxiliary quantity, which is not used directly for motor control (the IPM temperature is measured by default). The rule for the ADC channel assignment is that the DC-bus voltage and auxiliary quantity must be measurable on different ADC converters. If this rule is broken, a preprocessor error is issued.
- M1_MCDRV_ADCIO_UDCB_MEAS – this define should contain the name of the 16-bit fractional variable in which to store the DC-bus voltage measurements.
- M1_MCDRV_ADCIO_IABC_MEAS – this define should contain the name of the *GMCLIB_3COOR_T_F16* structure variable in which to store the phase current measurement results. The GMCLIB_3COOR_T_F16 datatype is defined in the Real Time Control Embedded Software Motor Control and Power Conversion Libraries (RTCESL). For more information on RTCESL, visit www.nxp.com/rtcesl.
- M1_MCDRV_ADCIO_SVMSECTOR – this macro defines the name of the unsigned integer variable which contains the current SVM sector number.
- M1_MCDRV_ADCIO_AUX_MEAS – this define stores the name of the 16-bit fractional variable in which to store the auxiliary-quantity measured values.
- M1_MCDRV_PWMIO_DUTY – this define should contain the name of the *GMCLIB_3COOR_T_F16* structure variable in which to define the required phase PWM duty cycles. The GMCLIB_3COOR_T_F16 datatype is defined in RTCESL.

4.2 MCDRV API

The ADC and PWM motor control drivers share the same API within its class. To ensure device independency of the MCDRV API, all driver functions are accessible through universally named macros in the *mcdrv_<platform>-<device>.h* files.

The available API of the ADC MCDRV is:

- *M1_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 it 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. This 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 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 average filters is by default set to eight samples. 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. It should be called after a sufficient number of *M1_MCDRV_CURR_3PH_CALIB*() calls. This function always returns *true*.
- *bool_t M1_MCDRV_GET*(*MCDRV_ADC_T**) – this function reads and calculates the actual values of the 3-phase currents, DC-bus voltage, and auxiliary quantity. The obtained measurements are stored in the M1_MCDRV_ADCIO_IABC_MEAS, M1_MCDRV_ADCIO_UDCB_MEAS, and M1_MCDRV_ADCIO_AUX_MEAS variables. This function always returns *true*.

The API for the PWM MCDRV is:

- *M1_MCDRV_PWM_T* – MCDRV PWM structure data type.
- *bool_t M1_MCDRV_PWM_PERIPH_INIT* (*M1_MCDRV_PWM_T**) – this function is by default called during the PWM periphery initialization procedure invoked by the *MCDRV_Init_M1*() function. This function always returns *true*.
- *bool_t M1_MCDRV_PWM3PH_SET*(*M1_MCDRV_PWM_T**) – this function updates the PWM phase duty cycles based on the required values stored in the M1_MCDRV_PWMIO_DUTY variable. This function always returns *true*.
- *bool_t M1_MCDRV_PWM3PH_EN*(*M1_MCDRV_PWM_T**) – calling this function enables all PWM channels. This function always returns *true*.
- *bool_t M1_MCDRV_PWM3PH_DIS* (*M1_MCDRV_PWM_T**) – calling this function disables all PWM channels. This function always returns *true*.
- *bool_t M1_MCDRV_PWM3PH_FAULT_GET*(*M1_MCDRV_PWM_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*.

5 Tuning and controlling the application

This section describes the tools and recommended procedures to control the sensorless PMSM Field-Oriented Control (FOC) application. The application contains the embedded-side driver of the FreeMASTER real-time debug monitor and data visualization tool to communicate with a PC. It supports non-intrusive monitoring, as well as the modification of target variables in real-time, which is very useful for algorithm tuning. Besides the target-side driver, the FreeMASTER tool requires the installation of the PC application as well. For more information, see www.nxp.com/freemaster.

The PMSM sensorless FOC application can be easily controlled and tuned using the Motor Control Application Tuning (MCAT) page for PMSM. The MCAT for PMSM is a user-friendly modular page, which runs within FreeMASTER. To launch it, simply execute the *.pmp* file located next to your project. See the user guide for your version of the PMSM sensorless application for more information. Figure 4 shows the MCAT for PMSM welcome page. The tool consists of a tab menu (point 1), the information about the connected board (point 2), and the workspace (point 3). Each tab 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 variable watch window are predefined in the FreeMASTER project file to further simplify motor parameter tuning and debugging. The basic and expert tuning modes are available. Selecting the expert mode gives you the access to modify all parameters and fields available in the MCAT. The basic mode is recommended for inexperienced users. When FreeMASTER is not connected to the target, the *App Id* line shows *offline*. When the communication with the target MCU with a correct software is established, the *App Id* line displays correct MCU and platform and all stored parameters for a given MCU are loaded.

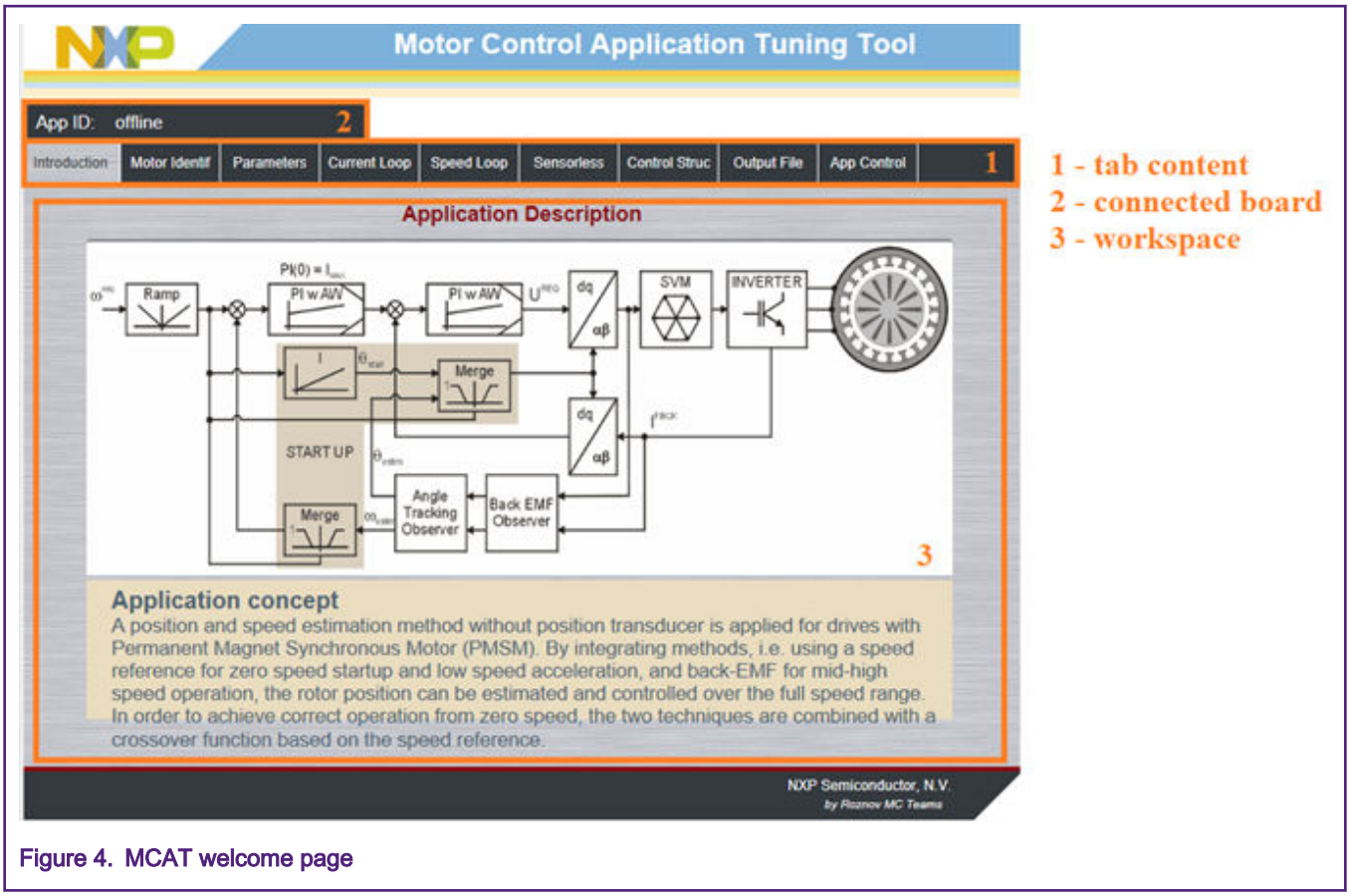

Figure 4. MCAT welcome page

In the default configuration, the following tabs are available:

- *Introduction* – welcome tab with the PMSM sensorless FOC diagram and a short description of the application.
- *Motor Identif* – PMSM semi-automated parameter measurement control tab. The PMSM parameter identification is described later on in this document.
- *Parameters* – this tab allows you to modify the motor parameters, specification of the hardware and application scales, and fault limits.
- *Current Loop* – current loop PI controller gains and output limits.
- *Speed loop* – this tab contains fields to specify the speed controller proportional and integral gains, as well as the output limits and the parameters of the speed ramp.
- *Sensorless* – this tab enables you to tune the parameters for the BEMF observer, tracking observer, and open-loop startup.
- *Control Struc* – this application control tab enables you to select and control a PMSM using different techniques (Scalar – Volt/Hertz control, Voltage FOC, Current FOC, and Speed FOC). The application state is shown in this tab.
- *Output file* – this tab enables you to view all calculated constants that are required by the PMSM 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 project rebuild.
- *Control page* – this tab contains graphical elements such as the speed gauge, DC-bus voltage measurement bar, and variety of switches that enable simple, quick, and user-friendly application control. The fault clearing and the demo mode, which sets various predefined required speeds over time, can be also controlled 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 them to 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 ACIM and how to appropriately tune the application.

5.1 PMSM parameter identification

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 stator resistance *R*s, direct inductance *L*d, quadrature inductance *L*q, and BEMF constant *K*e.

5.1.1 Power stage characterization

Each inverter introduces a 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 in Figure 5. The power stage characterization is a part of the MCAT and can be controlled from the *Motor Identif* tab. To perform the characterization, connect a motor with a known stator resistance *R*s and set this value in the *Calib Rs* field. Then specify the *Calibration Range*, which is the range of the stator current *i*s, in which the measurement of the *U*error is performed. Afterwards, start the characterization by clicking the *Calibrate* button. The characterization gradually performs 65 *i*sd current steps, from *i*s = -*I*s,calib to *i*s = *I*s,calib, each taking 300 ms. Be aware that the whole process takes about 20 seconds and that 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 maximum current *I*s,calib. It is recommended to use a motor with a low *R*s for characterization purposes.

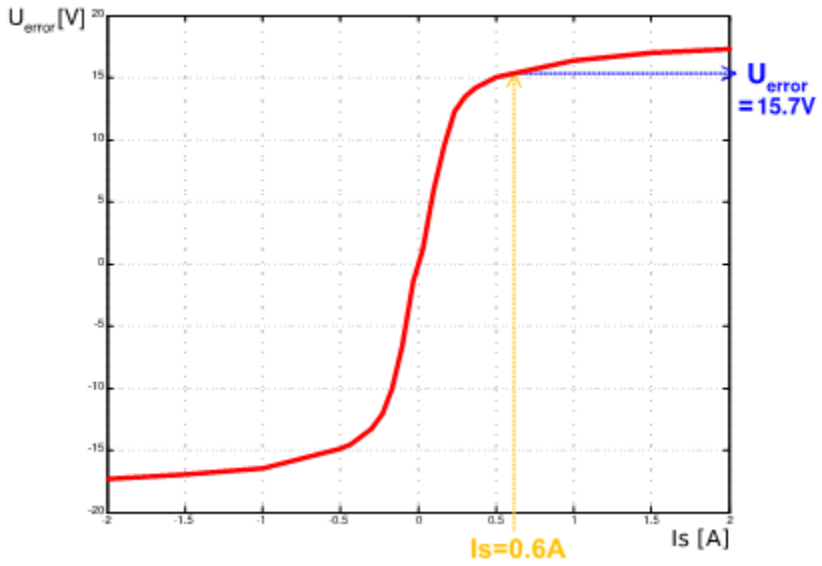

Figure 5. Example power stage characteristic

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

5.1.2 Stator resistance measurement

Stator resistance *R*s is measured with the DC current *I*phN value, which is applied to the motor for 1200 ms. The DC voltage *U*DC is held using the current controllers. Their parameters are selected conservatively, so that the stability is assured. The stator resistance *R*s is calculated using the Ohm's law as:

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

5.1.3 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 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 6.

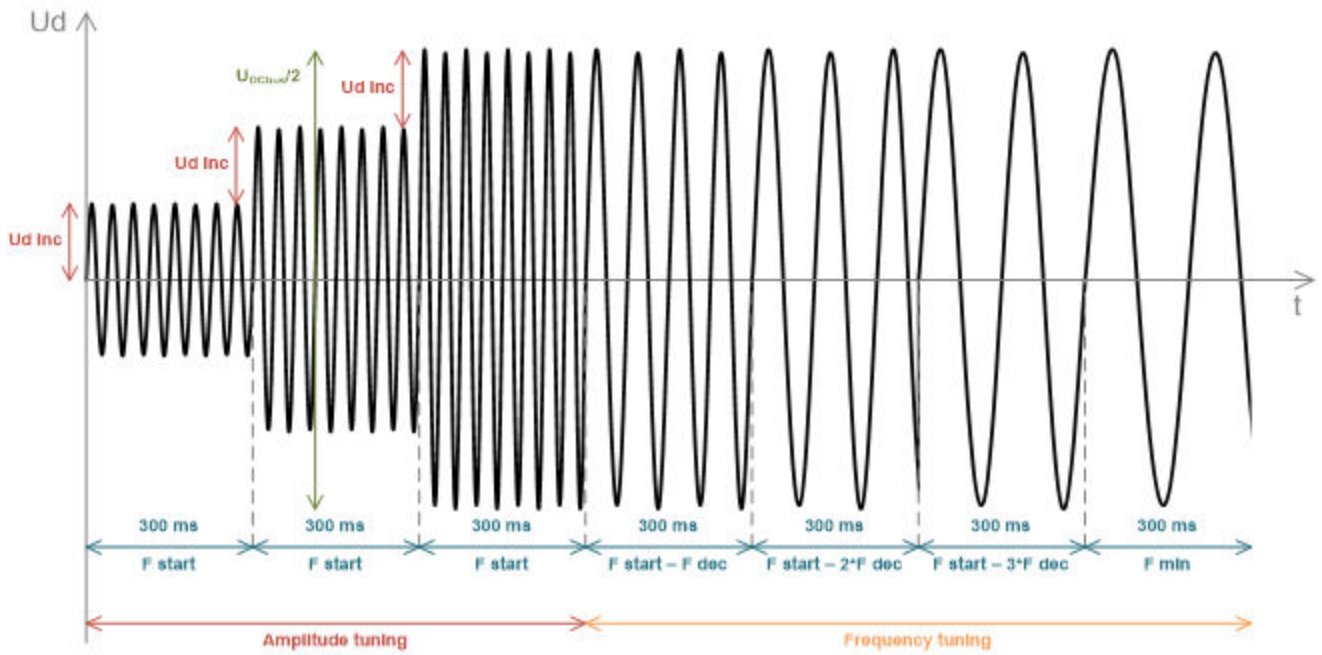

Figure 6. Tuning Ls measuring signal

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 Ls is calculated from the total impedance of the RL circuit.

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

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

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

The direct inductance (*L*d) and quadrature inductance (*L*q) measurements are completed in the same way as *L*S. Before the *L*d and *L*q measurement, a 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.

5.1.4 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 be spinning. *I*d is controlled through *Id 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:

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

When *K*e is being measured, you must visually check to determine whether the motor is spinning properly. If the motor is not spinning properly, perform the following steps:

- Ensure that the number of *pp* is correct. The required speed for the *K*e measurement is also calculated from *pp*. Therefore, an inaccuracy in pp causes inaccuracy in the resultant *K*e.
- Increase *Id meas* to produce higher torque when spinning during the open-loop.
- Decrease *N nom* to decrease the required speed for the *K*e measurement.

5.1.5 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 *pp* assistant performs one electrical revolution, stops for a few seconds, and then repeats. Because the *pp* value is a 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 an alignment occurs during the first revolution, which affects the number of stops. During the *pp* measurement, the current loop is enabled and the *Id* current is controlled to *Id 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 *Id meas*, and restart the assistant.

5.1.6 PMSM electrical parameter measurement process

The motor identification process can be controlled and set up from the MCAT *Motor Identif* tab, which is shown in Figure 7. To measure your own motor, follow these steps (depicted also in Figure 7):

- Select your hardware board. You can choose between a standard NXP hardware or your own hardware. If you use your own hardware, specify its scales (I max, U DCB max, Fast Loop Period) too.

- If you do not know the motor number of pole-pairs, use the number of pole-pair assistant described in Number of pole-pair assistant.
- If you are using your own hardware for the first time, perform the power stage characterization described in Power stage characterization.
- Enter the motor measurement parameters (depending on the *Basic/Expert* modes) and start the measurement by pressing the *Measure* button. You can observe which parameter is being measured in the *Status* bar.

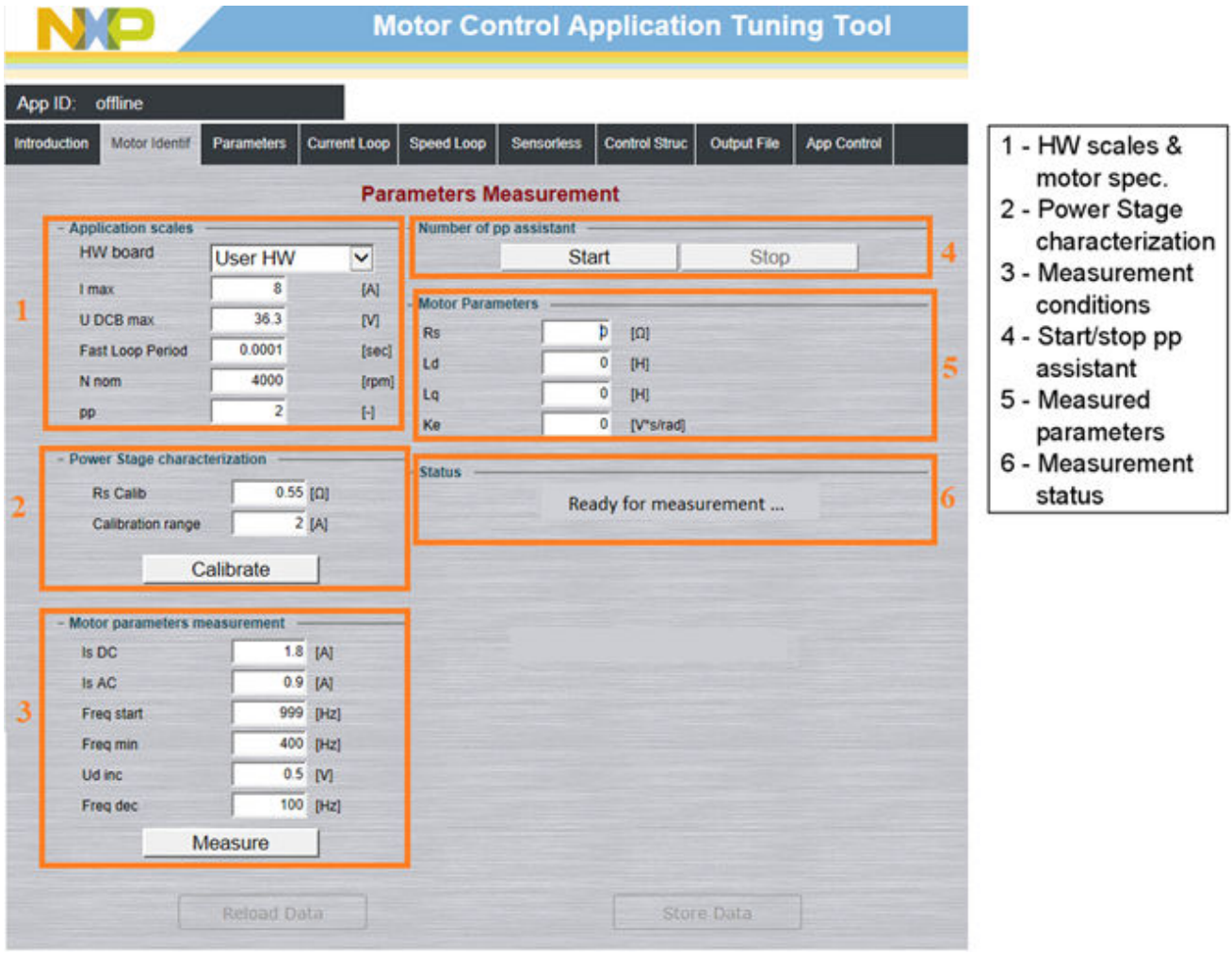

Figure 7. PMSM identification tab

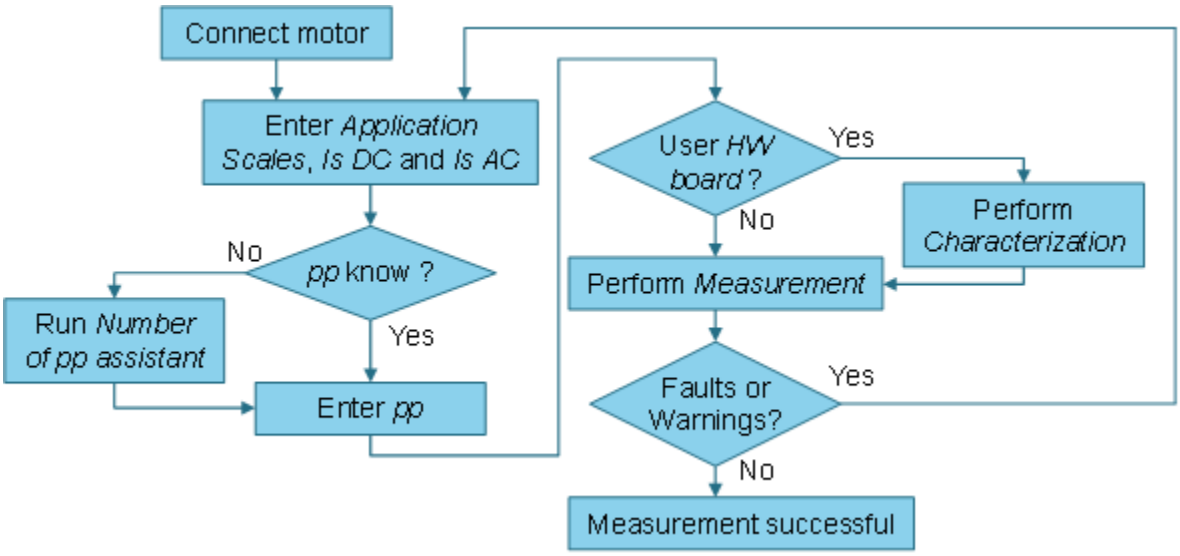

Figure 8. Measurement process diagram

During the measurement, measurement faults and warnings may occur. Don't 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 in Table 2.

Table 2. Measurement faults and warnings

Fault No.	Fault Description	Fault Reason	Troubleshooting
01	Motor not connected	Id > 50mA cannot be reached with the available DC-bus voltage.	Check that the motor is connected.
02	Rs too high for calibration	Calibration I cannot be reached with the available DC-bus voltage.	Use a motor with a lower Rs for the power stage characterization.
03	Current measurement *Is DC* not reached	User-defined *Is DC* was not reached, so the measurement was taken with a lower *Is DC*.	Raise the DC bus voltage to reach the *Is DC* or lower the *Is DC* to avoid this warning.
04	Current amplitude measurement *Is AC* not reached	User-defined *Is AC* was not reached, so the measurement was taken with a lower *Is AC*.	Raise the DC bus voltage or lower the *F min* to reach the *Is AC* or lower the *Is AC* to avoid this warning.
05	Wrong characteristic data	Characteristic data, which is used for voltage correction, does not correspond to the actual power stage.	Select the user hardware and perform the calibration.

5.2 PMSM sensorless application control and tuning using MCAT

FreeMASTER enabled with the MCAT page can be used to completely control and easily tune the ACIM sensorless FOC application. The MCAT for PMSM sub-module tabs, which were listed in Tuning and controlling the application, are described here.

5.2.1 Application control using MCAT

The application can be controlled through *Control Struc* tab, which is shown in Figure 9. The state control area on the left side of the screen shows the current application state and allows turning the main application switch *on* or *off* (turning a running application *off* disables all PWM outputs). The *Cascade Control Structure* area is on the right-hand side of the screen. Here you can select between the scalar and FOC controls using the appropriate buttons. The selected parts of the FOC cascade structure can be enabled by choosing the *Voltage FOC*, *Current FOC*, or full *Speed FOC*. This is useful for application tuning and debugging.

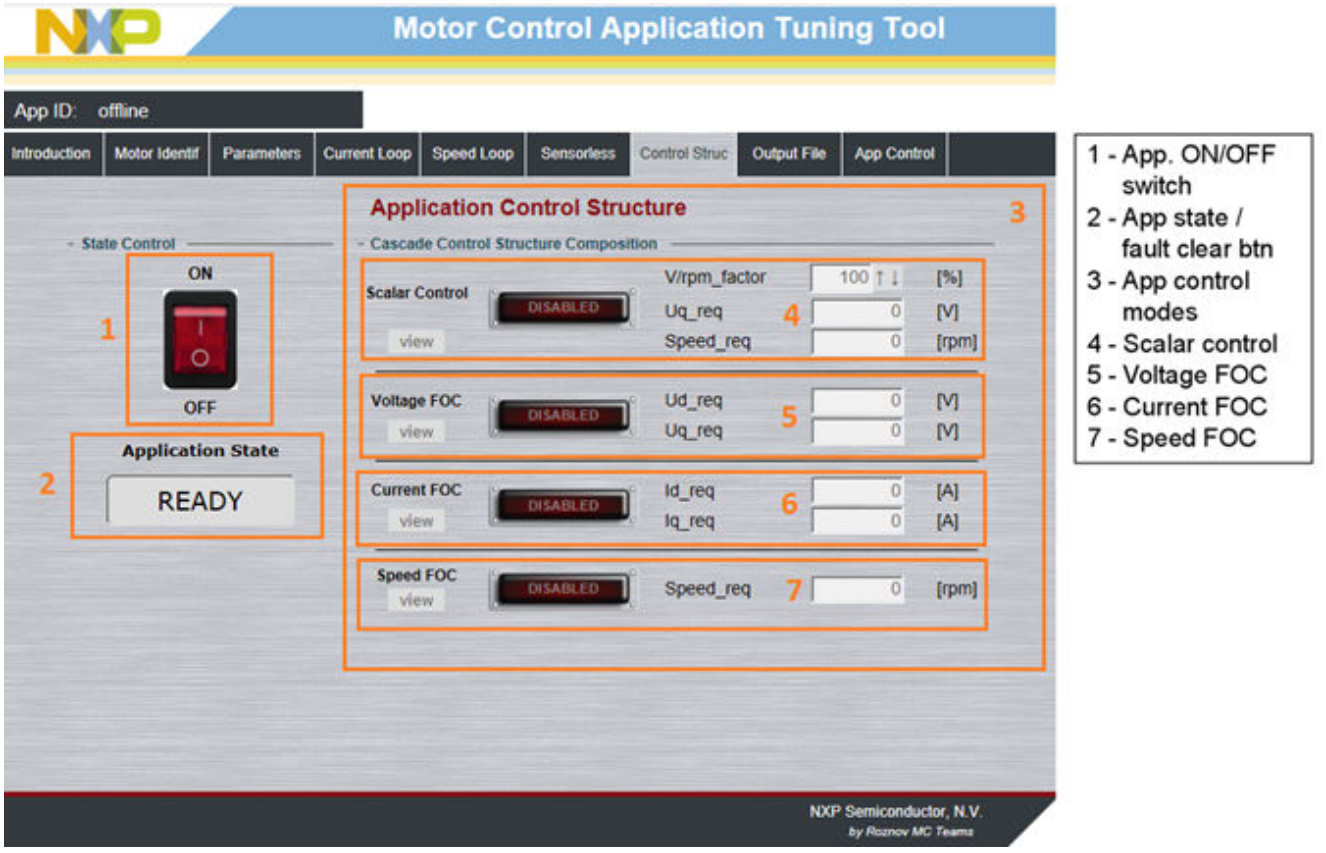

Figure 9. MCAT for PMSM control page

The *Scalar control* diagram is in Figure 10. It is the simplest type of motor-control techniques. The ratio between the magnitude of the stator voltage and the frequency (frequency information is hidden in the *Speed_req* value) must be kept at the nominal ratio, so the control method is sometimes called Volt per Hertz or V/Hz. Therefore, pay attention when entering the required voltage and speed in the expert tuning mode. The ratio is kept constant in the basic mode and the only required input is the required speed. The position estimation BEMF observer and the tracking observer algorithms run in the background even if the estimated position information is not directly used. This is useful for the BEMF observer tuning. See *Sensorless PMSM Field-Oriented Control* (document DRM148).

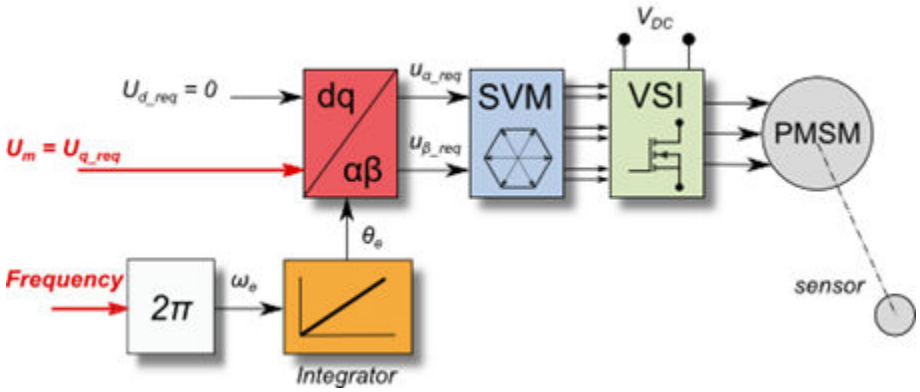

Figure 10. Scalar control mode

The block diagram of the so-called *Voltage FOC* is in Figure 11. Unlike the *Scalar control*, the position feedback is closed using the BEMF observer and the stator voltage magnitude is independent from the motor speed. Both the *d*-axis and *q*-axis stator voltages can be specified using the *Ud_req* and *Uq_req* fields. This control method is useful for the BEMF observer functionality check.

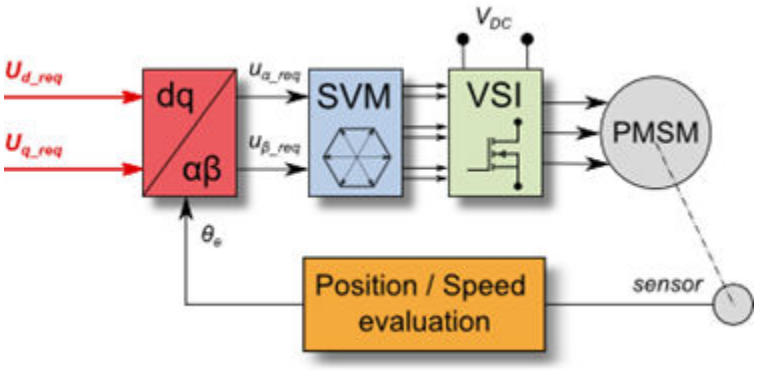

Figure 11. Voltage FOC control mode

The *Current FOC* or torque control requires the rotor position feedback, as well as the currents transformed into the d-q reference frame. There are two reference variables (*Id_req* and *Iq_req*) available for motor control, as shown in Figure 12. The *d*-axis current component *i*sd_req is responsible for the rotor flux control, while the *q*-axis current component of the current *i*sq_req generates torque and the motor runs by its application. By changing the polarity of the *i*sq_req current, the motor changes the rotation direction. Supposing that the BEMF observer is tuned correctly, the current PI controllers can be tuned using the *Current FOC* control structure.

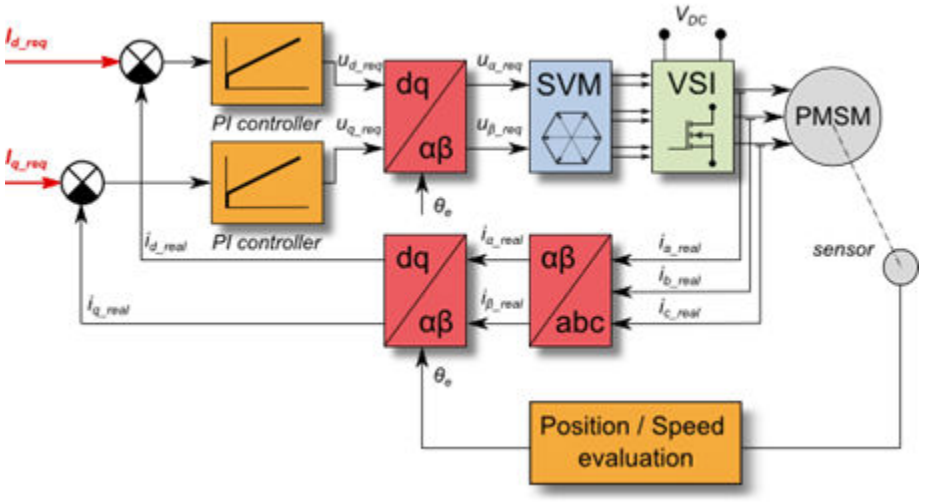

Figure 12. Current (torque) control mode

The speed PMSM sensorless FOC (its diagram is shown in Figure 13) is activated by enabling the *Speed FOC* control structure. The required speed can be entered in the *Speed_req* field. The *d*-axis current reference is kept at zero during the entire FOC operation. This control scheme is used for the speed PI controller design, which is the final stage of the PMSM sensorless application tuning.

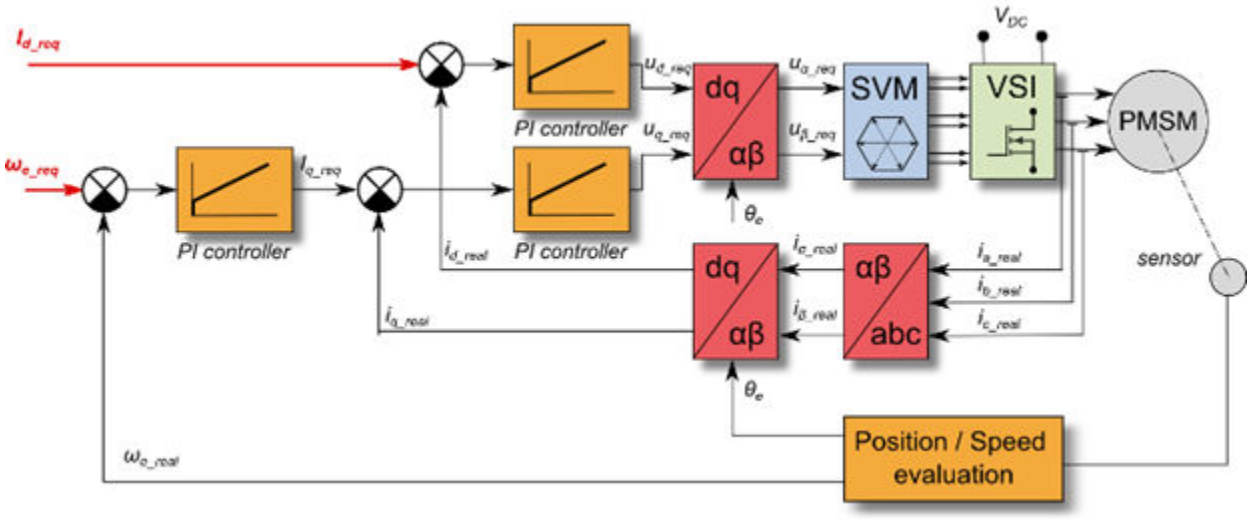

Figure 13. Speed FOC control mode

5.2.2 PMSM sensorless application tuning using MCAT

This chapter describes how to run your motor in several steps. It is highly recommended to go through all the steps carefully to eliminate any possible issues during the tuning process. Figure 14 depicts a typical PMSM sensorless control tuning process. The details of each tuning phase are described in the following sections.

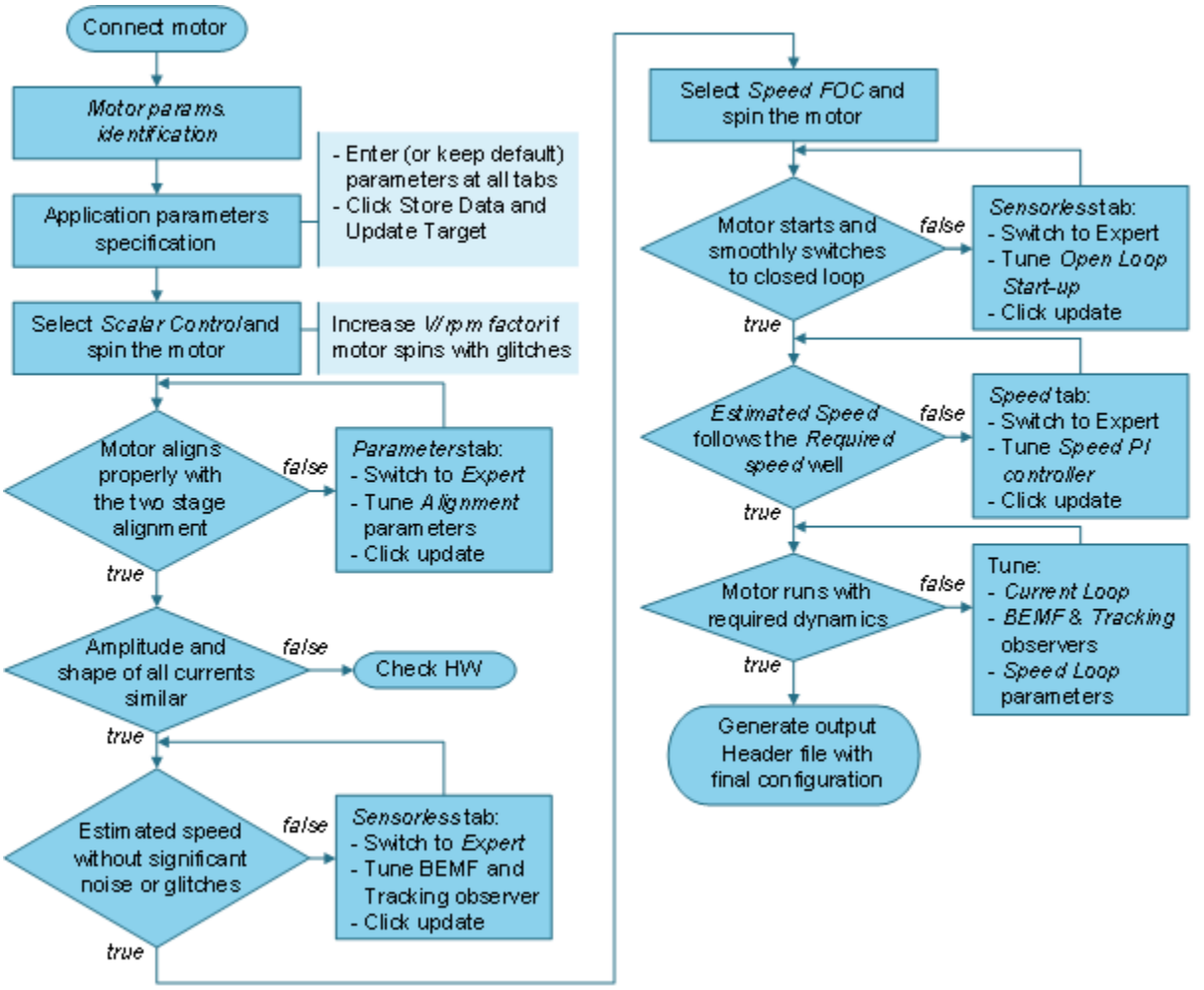

Figure 14. Running a new PMSM

5.2.3 Initial configuration settings and updates

- Open the PMSM sensorless control application FreeMASTER project containing the dedicated MCAT plug-in module.
- Select the "Basic" mode—recommended for users who are not experienced in the motor-control theory. The number of required input parameters is reduced.
- Select the "Parameters" tab.
- Leave the measured motor parameters empty or specify the parameters manually. 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 3 are accessible in the "Basic" and "Expert" modes. Motor inertia J expresses the overall system inertia that is very often difficult to obtain. Additional methods to identify the drive inertia are in non-NXP publications (for example; in IEEE). 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 3. MCAT motor parameters

Parameter	Units	Description	Typical range
pp	[-]	Motor pole pairs	1 - 10

Table continues on the next page...

Table 3. MCAT motor parameters (continued)

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.m2]	System inertia	0.000001 - 1
Iph nom	[A]	Motor nominal phase current	0.5 – 8
Uph nom	[V]	Motor nominal phase voltage	10 - 300
N nom	[rpm]	Motor nominal speed	1000 - 2000

- Set the hardware scales—the modification of these two fields is not required when using a reference to the standard Power Stage Board. These scales express the maximum measurable current and voltage analog quantities.
- Check the fault limits—these fields are not accessible in the "Basic" mode and they are calculated using the motor parameters and hardware scales (see Table 4).

Table 4. Fault limits

Parameter	Units	Description	Typical Range
U DCB trip	[V]	Voltage value when the external braking resistor switch is turned on	U DCB Over ~ U DCB max
U DCB under	[V]	Trigger value when the under-voltage fault is detected	0 ~ U DCB Over
U DCB over	[V]	Trigger value when the over-voltage fault is detected	U DCB Under ~ U max
N over	[rpm]	Trigger value when the over-speed fault is detected	N nom ~ N max
N min	[rpm]	Minimal actual speed value for sensorless control	(0.05~0.2) *N max

- Check the application scales—these fields are not accessible in the "Basic" mode and they are calculated using the motor parameters and hardware scales.

Table 5. Application scales

Parameter Name	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	—

- 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 rotor alignment and its duration.
- Click the “Store Data” button to save the modified parameters to the inner file.

5.2.4 Control structure modes

- 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.
- Turn the application switch on. The application state should change to RUN.
- Set the required speed value in the "Speed_req" field to 500 rpm in the "Scalar Control" section. The motor should start running, see Figure 15.

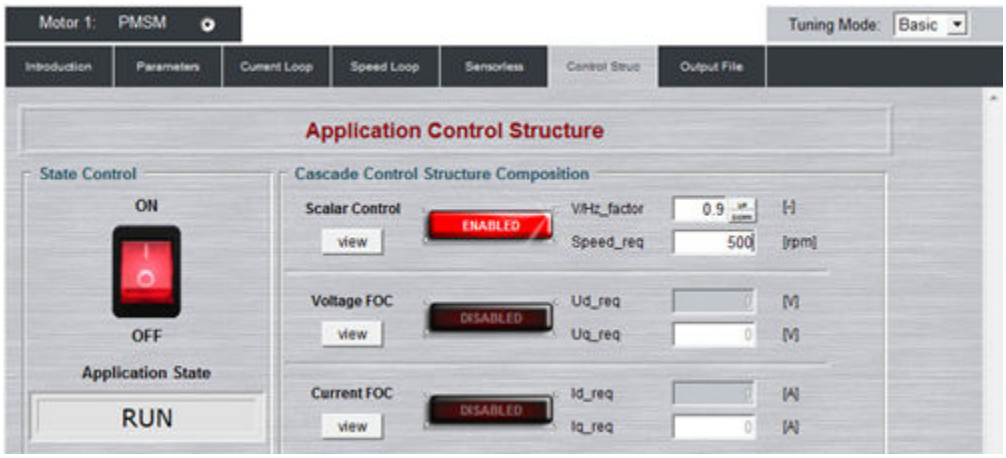

Figure 15. MCAT scalar control

- Select the “Phase Currents” recorder from the "Scalar & Voltage Control" FreeMASTER project tree.
- 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 the sinusoidal shape (see Figure 16).

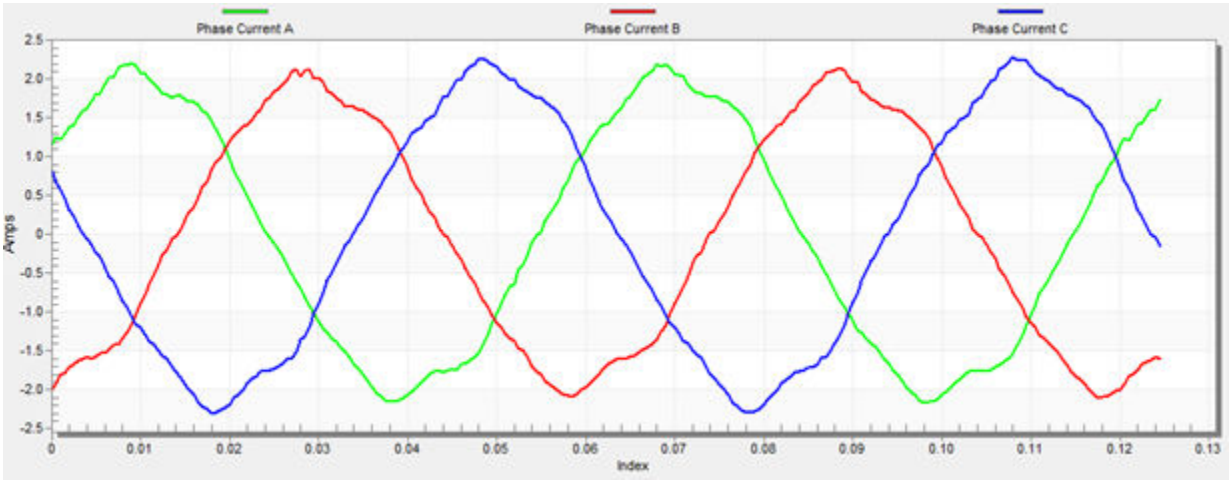

Figure 16. Phase currents

- 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 17) 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.

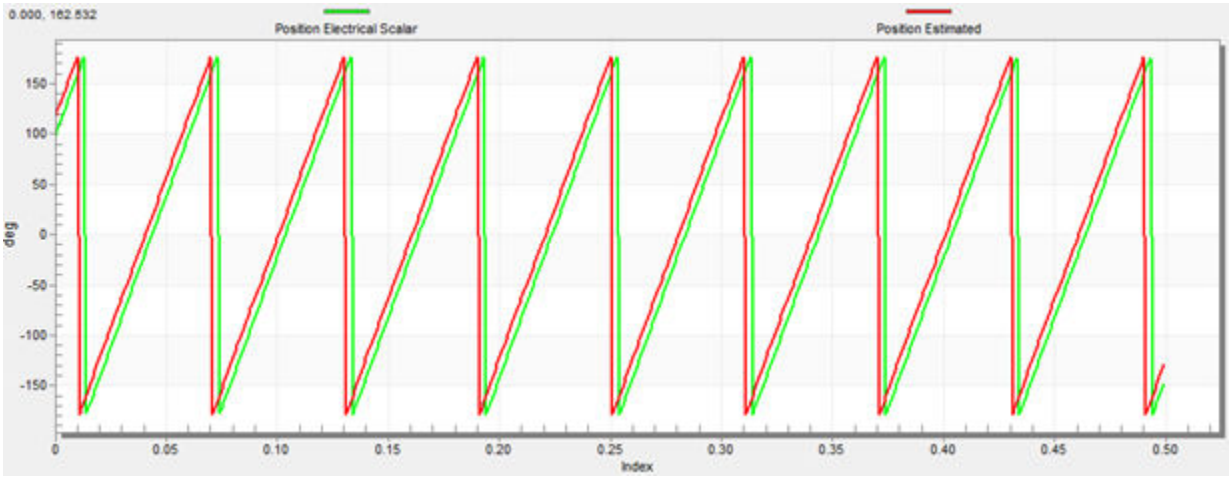

Figure 17. Generated and estimated positions

- If an opposite speed direction is required, set a negative speed value in the "Speed_req" field.
- At this step, proper observer functionality and measurement of analog quantities is expected.
- Enable the “Voltage FOC” mode by clicking the DISABLED button in the "Voltage FOC" section while the main application switch is turned off.
- 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.

5.2.5 Alignment tuning

The alignment procedure sets the rotor to an accurate initial position and applies full start-up torque to the motor. 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 oscillations before startup.

- The alignment voltage is a value applied to the d-axis during alignment. Increase this value for a higher shaft load.
- The alignment duration expresses the time when the alignment routine is called. Tune this parameter to have the rotor without oscillations or movement at the end of the alignment process.

5.2.6 Current loop tuning

The parameters for the current D,Q 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.

- Switch the tuning mode to “Expert”.
- 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.
- Select the "Current Controller" Id recorder.
- Select the "Control Structure" tab, switch to "Current FOC", set "Iq_req" 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 (see Figure 18).
- Tune the loop bandwidth and attenuation until the required response is achieved. 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 (see Figure 18).

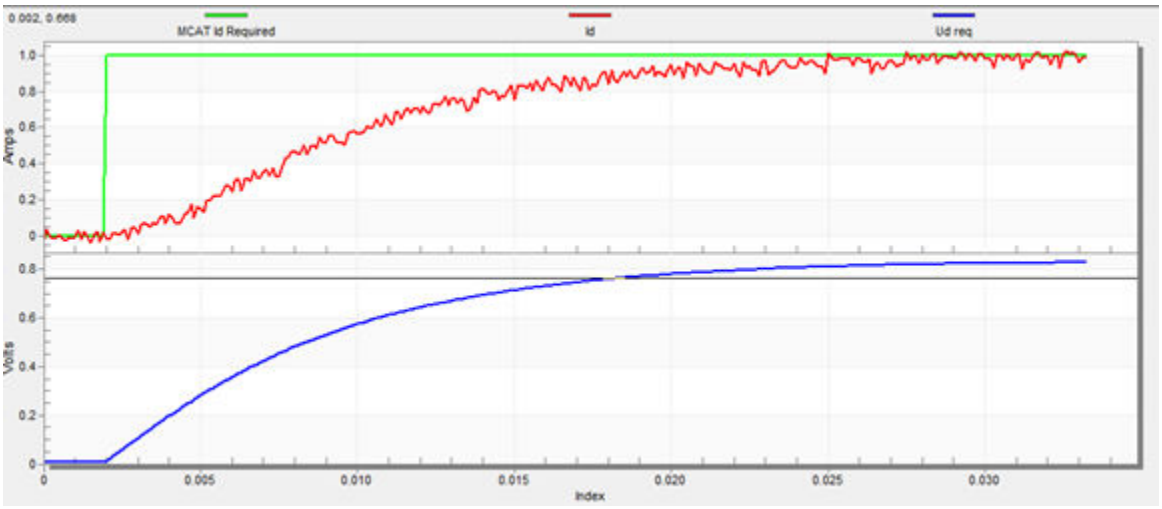

Figure 18. Slow step response of Id current controller

— The loop bandwidth (400 Hz) is optimal and the response time of the Id current is sufficient (see Figure 19).

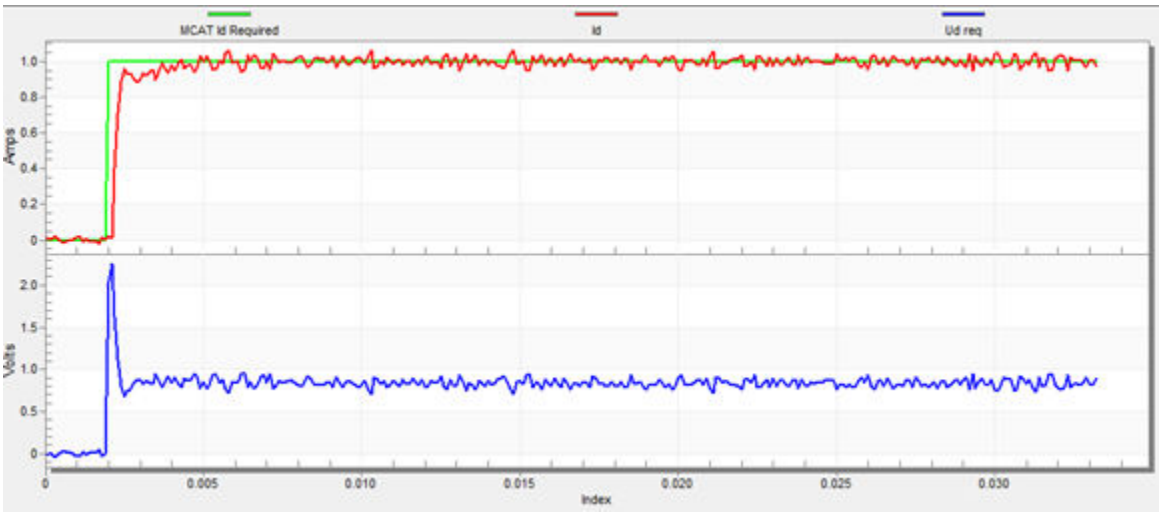

Figure 19. Optimal step response of Id current controller

— 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 20).

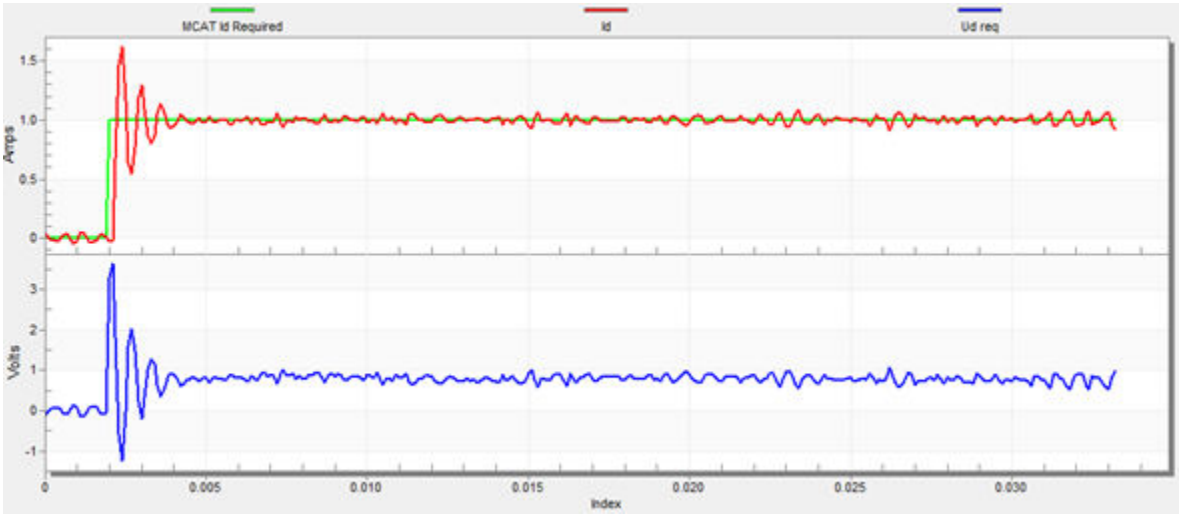

Figure 20. Fast step response of Id current controller

5.2.7 Actual speed filter

The estimated speed from the BEMF observer is fed into the speed PI controller through the IIR filter. The filter cut-off frequency can be modified in the "Expert" mode in the "Speed loop" tab. The speed loop sample time is typically several milliseconds, so the actual speed filter cut-off frequency is mostly in the range from 5 Hz to 100 Hz.

The filter output can be tracked in the Speed scope. The modified filter cut-off frequency value must be applied to the MCU by clicking the “Update Target” button.

5.2.8 Speed ramp tuning

The speed command is applied to the speed controller through a speed ramp. The ramp function contains two increments (up and down) that express the motor acceleration and deceleration per second. If the increments are very high, it can cause an over-current fault during acceleration and an over-voltage fault during deceleration. You can see whether the "Speed Actual Filtered" waveform shape equals the "Speed Ramp" profile in the "Speed" scope.

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 21. The ramp down increment is set to 500 rpm/sec, while the up increment is set to 3,000 rpm/sec.

The start-up ramp increment is located in the "Sensorless" tab and its value is usually higher than that for the speed loop ramp.

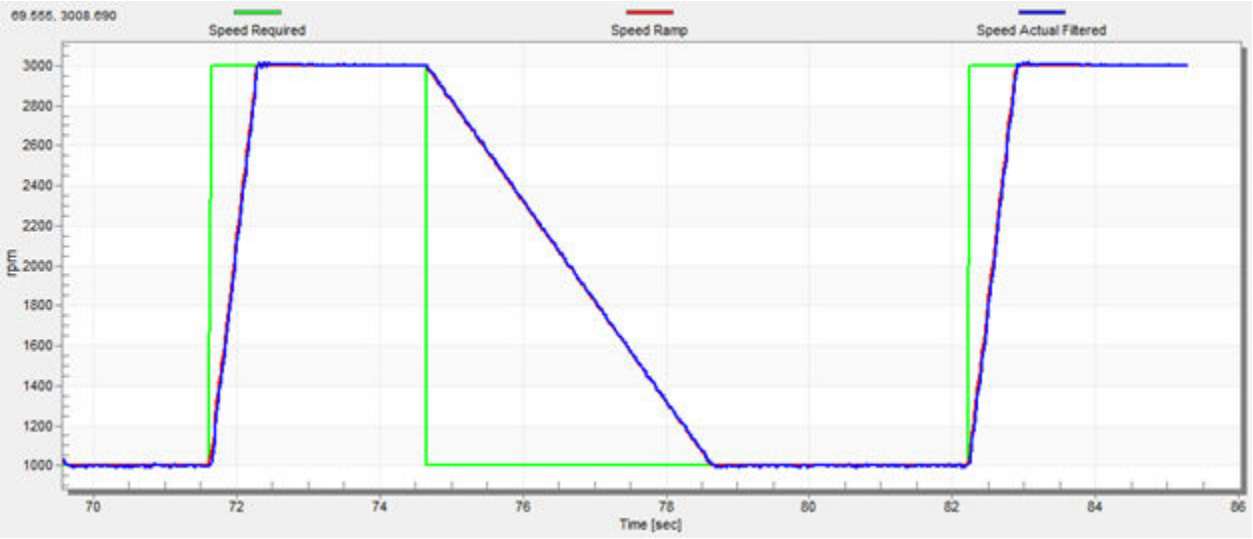

Figure 21. Speed profile

5.2.9 Open-loop startup

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 apart from the scalar control. Setting optimal values achieves a proper motor startup. An example start-up state of low-dynamic drives (fans, pumps) is shown in Figure 22.

- Select the "Startup" recorder from the FreeMASTER Project Tree.
- Set the start-up ramp increment typically to a higher value than the speed loop ramp increment.
- 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.
- Set the required merging speed - the threshold when the open-loop and estimated positions start to merge, mostly in the range from 5 % to 10 % of the nominal speed.
- Set the merging coefficient - the position merging process duration; 100 % corresponds to a half of the electrical revolution. The higher the value, the faster the merge is done. Values close to 1 % are set for drives where a high start-up torque and a smooth transition between the open loop and the closed loop are required.
- Click the “Update Target" button to apply the changes to the MCU.
- Switch to the "Control Structure" tab and enable the speed FOC.
- Set the required speed higher than the merging speed.
- Check the start-up response in the recorder.
- Tune the start-up parameters until the optimal response is achieved.
- If the rotor does not start to run, increase the start-up current.
- If the merging process fails (the rotor is stuck or not moving), decrease the start-up ramp increment, increase the merging speed, and set the merging coefficient to 5 %.

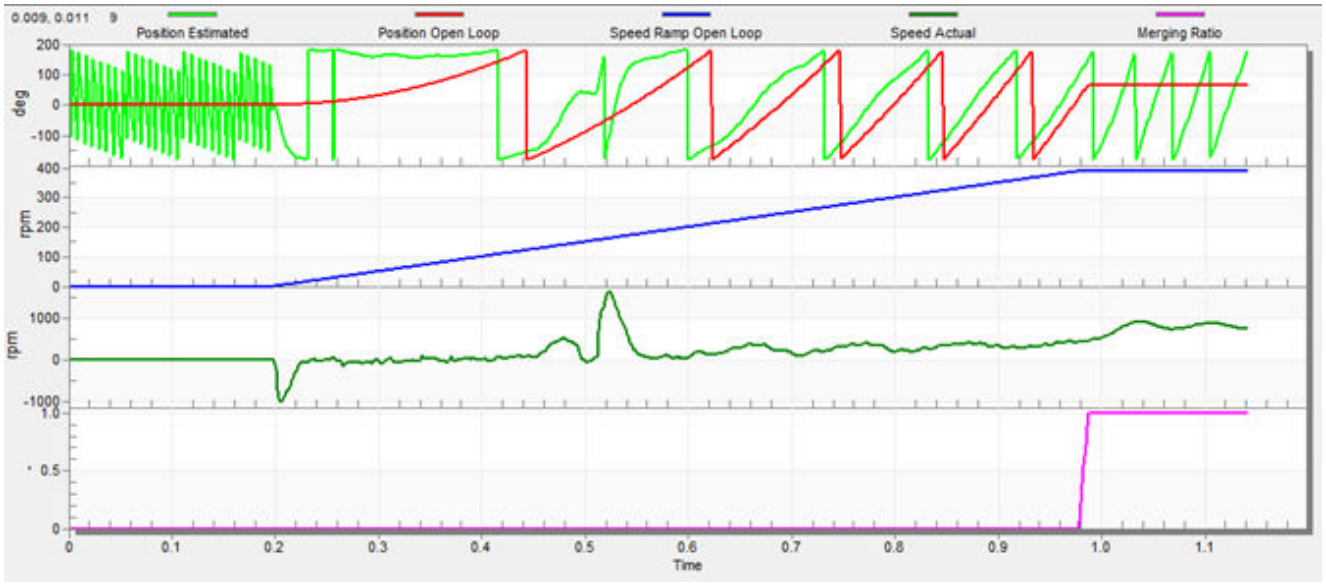

Figure 22. Motor startup

5.2.10 BEMF observer tuning

The "BEMF Observer" and Tracking Observer" parameters 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 optimal response, the bandwidth and attenuation parameters can be tuned.

- Switch the tuning mode to "Expert".
- Select the "Observer recorder" from the FreeMASTER Project Tree.
- Set the required bandwidth and attenuation of the BEMF observer—the bandwidth is typically set to a value close to the "Current Loop" bandwidth.
- Set the required bandwidth and attenuation of the tracking observer—the bandwidth is typically set in the range from 10 Hz to 20 Hz for most low-dynamic drives (fans, pumps).
- Click the "Update Target" button to apply the changes to the MCU.
- Check the observer response in the recorder.

5.2.11 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 these mechanical constants are available, the PI controller constants can be tuned using loop bandwidth and attenuation. In practice, the values of the motor and the load inertias and frictions are very often unknown and it is quite difficult to obtain them. Therefore, a manual tuning of the P and the I portions of the speed controllers is available to obtain the required speed response. See the example response in Figure 23. There are dozens of approaches for tuning the PI controller constants. The following steps provide one of these examples to tune the speed PI controller for a PM synchronous motor.

- Select the "Speed Controller" from the FreeMASTER Project Tree.
- Select the "Speed loop" tab.
- Check the "Manual Constant Tuning" option, which means that the bandwidth and attenuation fields are disabled and *SL_Kp* and *SL_Ki* are enabled.
- Tune the proportional gain:
  - Set the *SL_Ki* integral gain to zero.

- Set the speed ramp to 1000 rpm/sec or higher.
- Switch to the "Control Structure" tab and run the motor at a convenient speed (roughly 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 properly to the required value, without oscillations or excessive overshoot:
  - If *SL_Kp* is set low, the system response is slow.
  - If *SL_Kp* is set high, the system response is tighter.
  - When *SL_Ki* = 0, the system will probably not achieve the required speed.
- Click the "Update Target" button to apply the changes to the MCU.

- Tune the integral gain:
  - Increase *SL_Ki* slowly to minimize the difference between the required and actual speeds to zero.
  - Adjust *SL_Ki* in such a way that no oscillation nor large overshoot of the actual speed value is visible while the required speed step is applied.
  - Click the "Update Target" button to apply the changes to the MCU.
- Tune the loop bandwidth and attenuation until the required response is received. The example waveforms are shown in the following figures with the correct and incorrect settings of the current loop parameters:
  - The *SL_Ki* value is low and the *Speed Actual Filtered* does not achieve the *Speed Ramp* (see Figure 23).

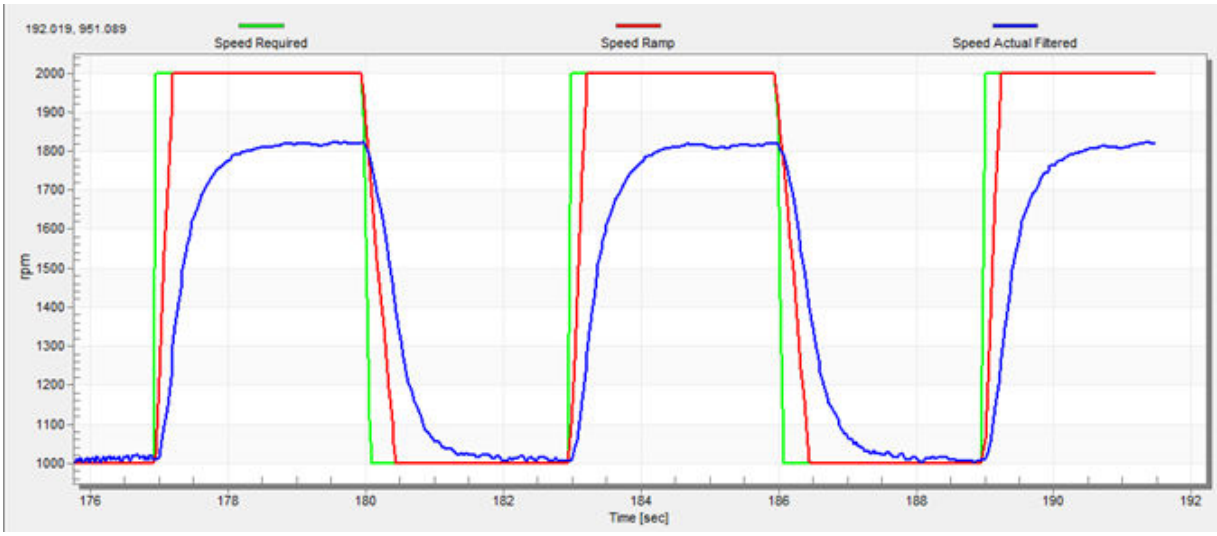

Figure 23. Speed controller response—SL_Ki value is low, speed ramp is not achieved

  - The *SL_Kp* value is low, the *Speed Actual Filtered* greatly overshoots, and the long settling time is not wanted (see Figure 24).

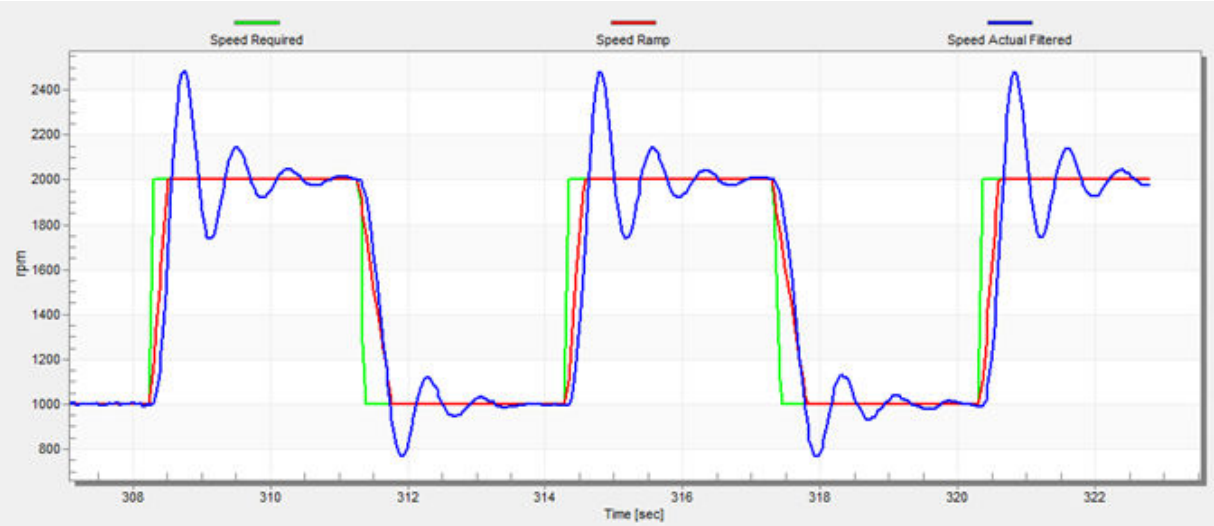

Figure 24. Speed controller response—SL_Kp value is low, speed actual filtered greatly overshoots

— The speed loop response has a small overshoot and the *Speed Actual Filtered* settling time is sufficient. Such response is considered optimal (see Figure 25).

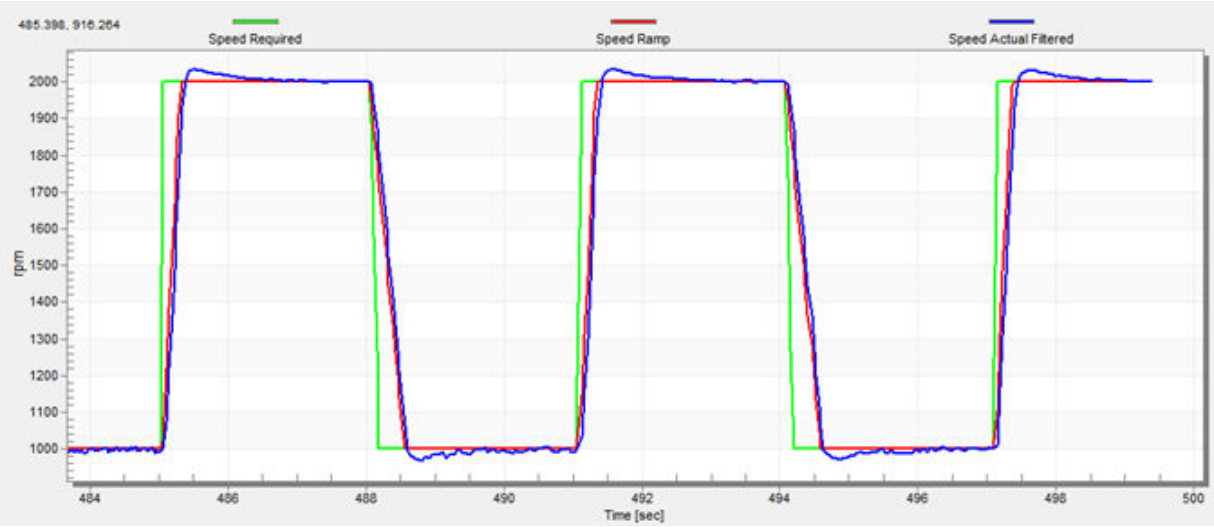

Figure 25. Speed controller response—speed loop response with small overshoot

5.2.12 MCAT output file generation

When you successfully finish the tuning of the application and you want to store all calculated parameters to the embedded application, navigate to the "Output File" tab. There is a list of all definitions generated by the MCAT. Clicking the "Generate Configuration File" button overwrites the old version of the *m1_pmsm_appconfig.h* file, which contains those definitions. For a proper motor parameter file generation, provide a correct path to the file. To change the path, navigate to the right-hand corner of the MCAT screen, until a symbol with screwdriver and wrench appears. When you click this symbol, the *Application Settings Page* appears. In the *Project Path Selection* area, modify the path to the *m1_pmsm_appconfig.h* file.

6 Conclusion

This application note describes the implementation of a sensorless field-oriented control of a 3-phase PMSM on the DSC MC56F837xx devices and NXP high-voltage and Freedom development platforms. The hardware-dependent part of the

sensorless control software, including a detailed peripheral setup, Motor Control (MC) peripheral drivers, and application timing are described in MCU features and peripheral settings. The motor parameters identification theory and the identification algorithms themselves are described in PMSM parameter identification. The last part of document introduces and explains the user interface represented by the Motor Control Application Tuning (MCAT) tool based on the FreeMASTER communication interface.

7 Acronyms and abbreviations

Table 6. Acronyms and abbreviations

Acronym	Meaning
AC	Alternating Current
ADC	Analog-to-Digital Converter
AN	Application Note
CPU	Central Processing Unit
CMP	Comparator
DC	Direct Current
DRM	Design Reference Manual
FOC	Field-Oriented Control
FTM	FlexTimer Module
GPIO	General-Purpose Input/Output
I/O	Input/Output interfaces between a computer system and the external world. The CPU reads the input to sense the level of an external signal and writes to the output to change the level of the external signal.
MCAT	Motor Control Application Tuning tool
MCDRV	Motor Control peripheral Drivers
MCU	Microcontroller
PDB	Programmable Delay Block
PI	Proportional Integral controller
PMSM	Permanent Magnet Synchronous Machine
PWM	Pulse-Width Modulation.
UART	Universal Asynchronous Receiver/Transmitter

8 References

The following references are available on www.nxp.com:

1. *Sensorless PMSM Field Oriented Control* (document DRM148)
2. *MC56F83xxx Reference Manual* (document MC56F83XXXRM)
3. *Freedom FRDM-MC-LVPMSM Development Platform User's Guide* (document FRDMLVPMSMUG)
4. *Freescale High-Voltage Motor Control Platform User's Guide* (document HVPMC3PHUG)
5. *Motor Control Application Tuning (MCAT) Tool for 3-Phase PMSM* (document AN4642)

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.

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 03/2020
Document identifier: AN12745