NXP Semiconductors
User's Guide

Document identifier: ACIMKV46DEMOUG
Rev. 0, 05/2020

ACIM KV46 Demo

Contents

Chapter 1
Introduction

This user's guide provides a step-by-step guide on how to build and download the SDK package with the AC Induction Machine (ACIM) sensorless application, open and flash demo software into MCU, and implement the sensorless field-oriented control software for a three-phase ACIM. It includes the machine parameters identification algorithm on 32-bit Kinetis MCUs. The sensorless control software and the ACIM control theory in general is described in *Sensorless ACIM Field-Oriented Control* (document DRM150). The motor parameter identification theory and algorithms are described in ACIM parameter identification. The hardware-dependent part of the sensorless control software, which includes the peripheral setup and the Motor Control Peripheral Drivers (MCDRV), is described as well. The last part of the document describes the user interface represented by the Motor Control Application Tuning (MCAT) page based on the FreeMASTER run-time debugging tool. These tools represent a simple and user-friendly way of machine parameter identification, algorithm tuning, software control, debugging, and diagnostics.

Chapter 2
Hardware setup

This section describes the default supported hardware configurations consisting of the HVP-MC3PH power stage, supported daughter card, and default induction motor.

2.1 HVP-MC3PH power stage

The ACIM reference solution package is available only for the 3-phase High-Voltage Motor-Control Platform (HVP), which is a 115/230-VAC, 1-kW power stage and a part of the HVP-MC3PH kit. In combination with one of the supported controller cards based on a Kinetis MCU, it provides a software development platform for more than one horse-power high-voltage motors. The block diagram of the complete High-Voltage Motor-Control Platform with the controller card is shown in Figure 1.

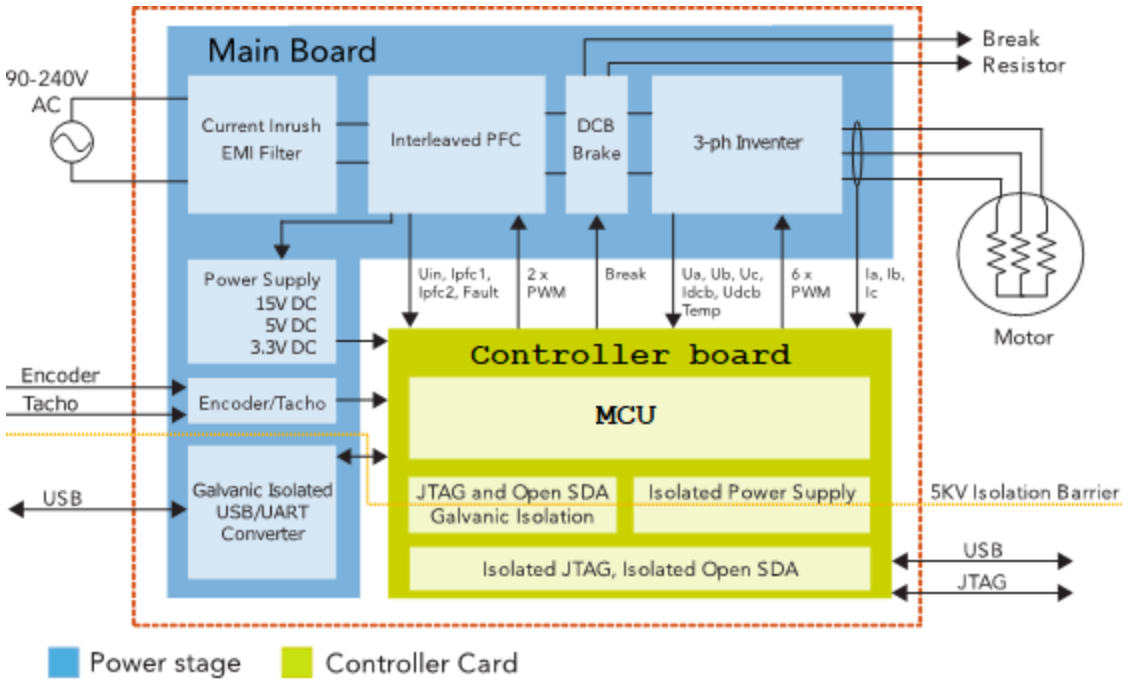

Figure 1. High-Voltage Motor-Control Platform block diagram

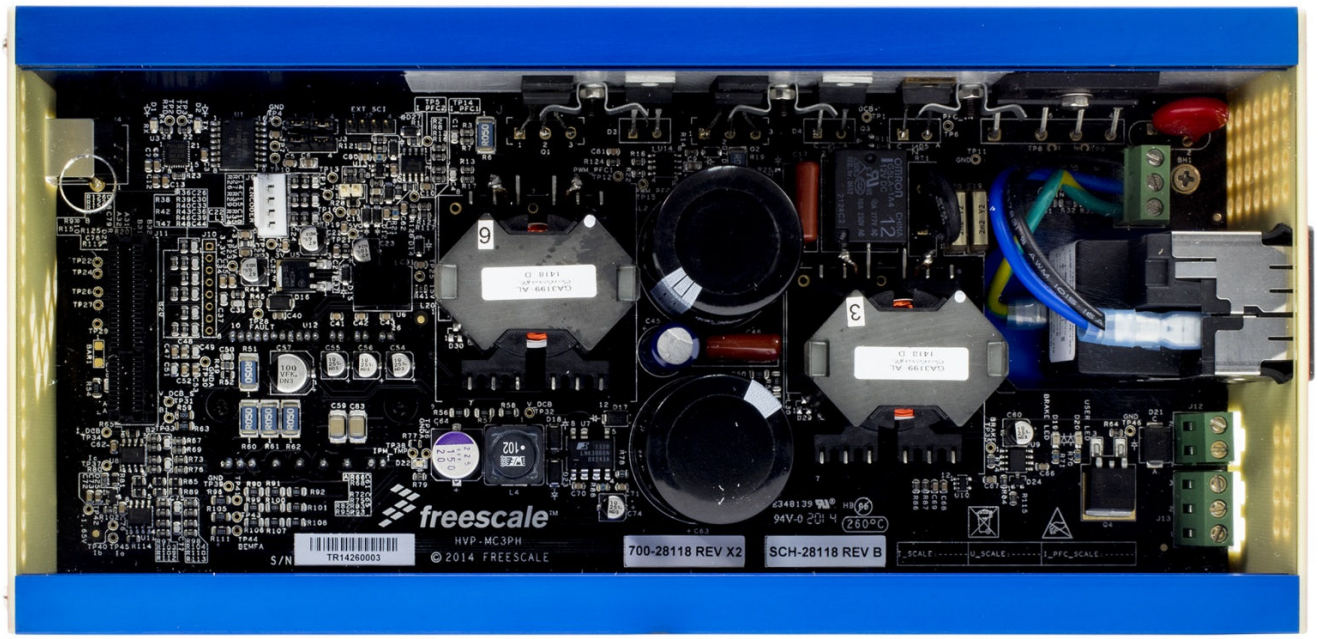

Figure 2. HVP-MC3PH High-Voltage Platform

The HVP power stage setup is easy and straightforward. See *Freescale High-Voltage Motor-Control Platform User's Guide* (document HVPMC3PHUG) for more information about the HVP setup.

NOTE

Due to the presence of high voltage, the HVP platform represents a safety risk when not used properly. For more information about the High-Voltage Motor-Control Platform, see www.nxp.com.

2.2 Default AC induction motor

The default induction motor (for which the application is pre-tuned) is Elektrim 0.33HP. The motor parameters provided by the manufacturer are listed in Table 1.

Table 1. Elektrim 0.33HP motor parameters

Characteristic	Symbol	Value	Units
Nominal voltage	U_N	230/400	V
Nominal frequency	f_N	50	Hz
Nominal current	I_N	1.5/0.85	A
Number of pole pairs	pp	2	-

2.3 HVP-KV46F150M daughter card

The HVP-KV46F150M daughter card features a Kinetis KV46F MCU, which is based on the Arm® Cortex®-CM4F core with a single-precision floating-point unit, running at 168 MHz and containing up to 256 KB of flash memory (see *KV4x Reference Manual* (document KV4XP100M168RM)). This daughter card is developed for use in motor-control applications, together with the High-Voltage Platform power stage. It contains OpenSDA, which is NXP's USB-based open-source hardware embedded serial and debug adapter and bootloader.

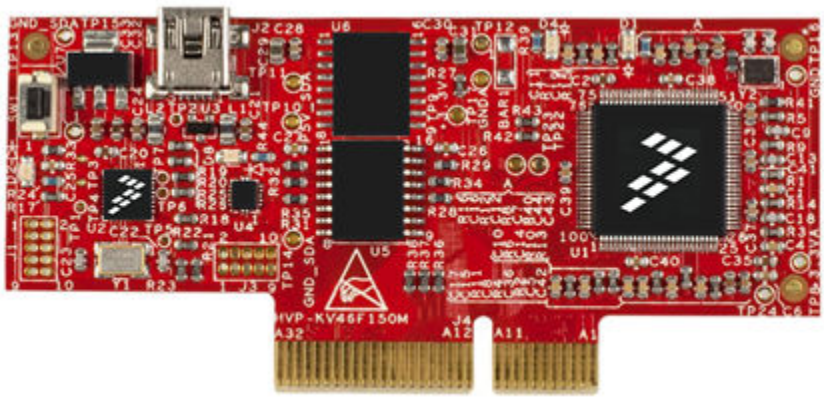

Figure 3. HVP-KV58F220M daughter card

2.4 High-Voltage Platform assembling

1. Make sure the HVP-MC3PH power stage is disconnected from the power source and the capacitors are not charged (no LED is lit).
2. Insert one of the supported daughter cards to the HVP-MC3PH main board (connector J11).
3. Connect the ACIM motor phase wires into the screw terminals on the board (MOTOR connector J13).
4. Place the protective plastic cover on top of the power stage to ensure safety.
5. Connect the USB cable to the OpenSDA mini USB connector.
6. Connect the power supply to the power connector and switch the power stage on.

Chapter 3
Demo setup

This section describes how to run the demo software.

3.1 Running the demo software

Downloading the demo software into the controller board:

1. Assembly the NXP hardware according to the instructions in High-Voltage Platform assembling.
2. Open the downloaded ACIM application in the IDE tool for which the software was downloaded (or choose your favorite IDE tool if you downloaded the ACIM application for all supported IDE tools).
3. Flash the project into the target device via the OpenSDA debug interface, as described in the *Getting Started with MCUXpresso SDK* document.

NOTE

The *Getting Started with MCUXpresso SDK* document is included in the downloaded SDK package.

Running the motor:

1. Open the FreeMASTER project and establish the communication between the MCU and the PC according to the instructions in Remote control using FreeMASTER.
2. Set up the required motor speed using the "Control Struc" tab (Figure 18).

Stopping the motor:

1. Click the “ON/OFF” button in the "Control Struc" tab (Figure 18).
2. Set the required speed to zero in the "Control Struc" tab (Figure 18).
3. In case of emergency, turn off the power supply.

Clearing the fault:

To clear the fault, remove the fault source (for example under-voltage) and click the fault button in the "Control Struc" tab, as shown in Figure 4.

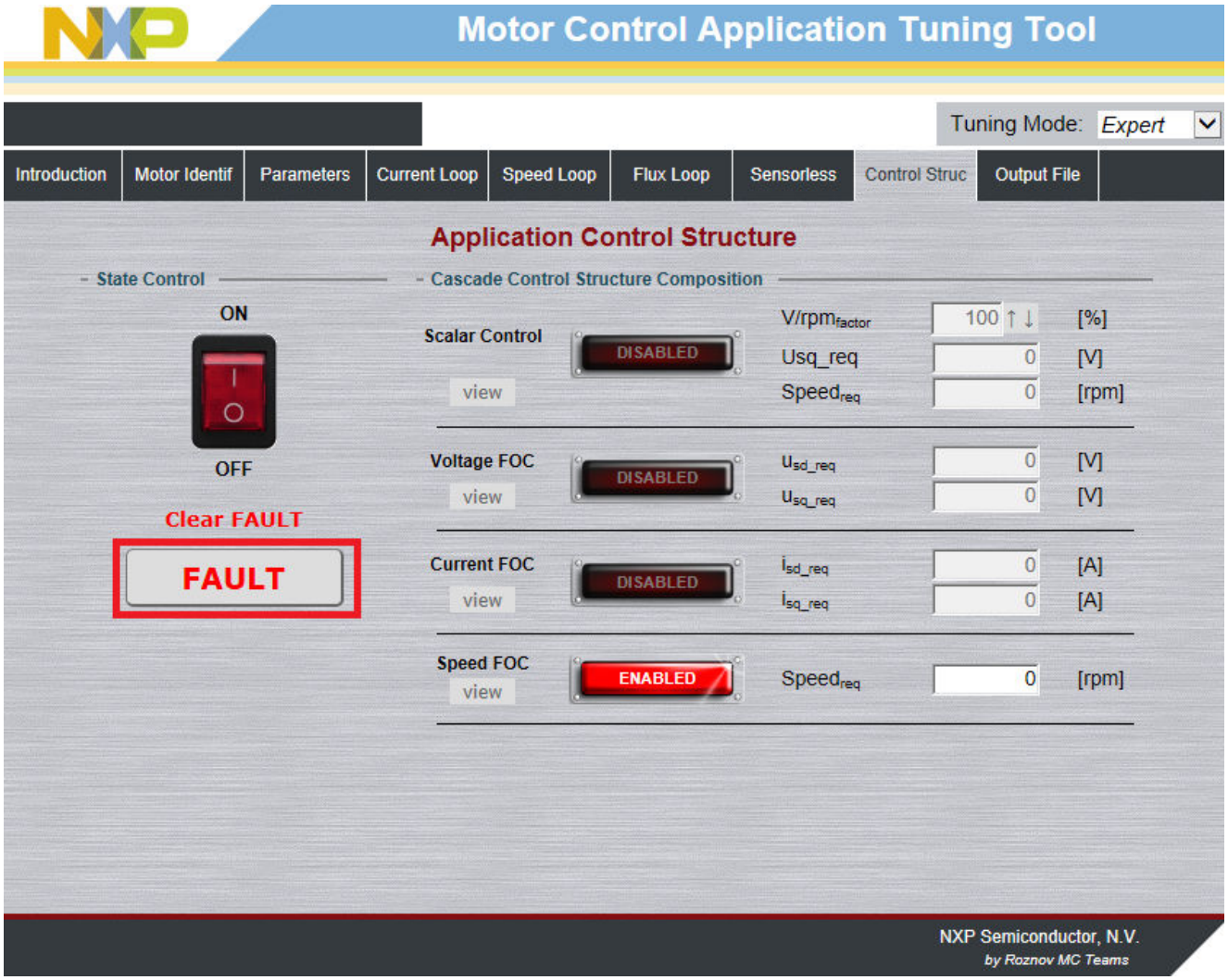

Figure 4. Fault clearing

Chapter 4
MCU peripheral settings

This section focuses on the hardware-dependent part of code, which includes the peripheral initialization and explanation of the application timing.

4.1 MKV46 family

The MKV46F family of Kinetis MCUs is a high-performance solution built around the Arm® Cortex®-M4F core running at 168 MHz with a floating-point unit, up to 256 KB of flash, and 32 KB of RAM. The MKV46F MCUs are targeted mainly at motor-control applications. Advanced peripherals, such as the high-resolution pulse-width modulation modules with a total of 30 PWM channels and the high-speed dual 12-bit Analog-to-Digital Converters (ADCs) make these devices ideal for multi-motor systems. For more information, see the *KV4x Reference Manual* (document KV4XP100M168RM).

The HVP-KV46F150M controller card is based on the MKV46F256VLL15 MCU. The controller card is equipped with the open-standard serial and debug USB-based interface (OpenSDA). For more information about the HVP-KV46F150M controller card, see the *HVP-KV46F150M User's Guide* (document HVPKV46F150MUG).

The peripherals (whose setup is described in more detail later on in this chapter) used by the ACIM motor-control software on MKV46F are:

- 12-bit cyclic Analog-to-Digital Converter (ADC12) for the phase currents, DC-bus voltage, and IPM temperature measurement.
- eFlexPWM module (PWMA) for 6-channel PWM generation.
- FlexTimer Module 1 (FTM1) for the slow control loop timing.
- XBARA multiplexer for the over-current fault and ADC12 trigger routing.
- Multi-purpose Clock Generator (MCG) and System Integration Module (SIM) for the MCU clock setup and distribution.
- Universal Asynchronous Receiver and Transmitter UART1 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 5. All tasks are handled using these interrupt service routines:

- ADCA_IRQHandler()—level-one priority interrupt triggered when the conversion of all enabled samples is completed by ADCA. It handles the fast control loop of FOC and the FreeMASTER recorder feature.
- FTM1_IRQHandler()—level-two priority interrupt triggered by the overflow of FTM1. It handles the slow control loop of FOC.

The fast and slow control loop ISRs are described in more detail in *Sensorless ACIM Field-Oriented Control* (documentDRM150).

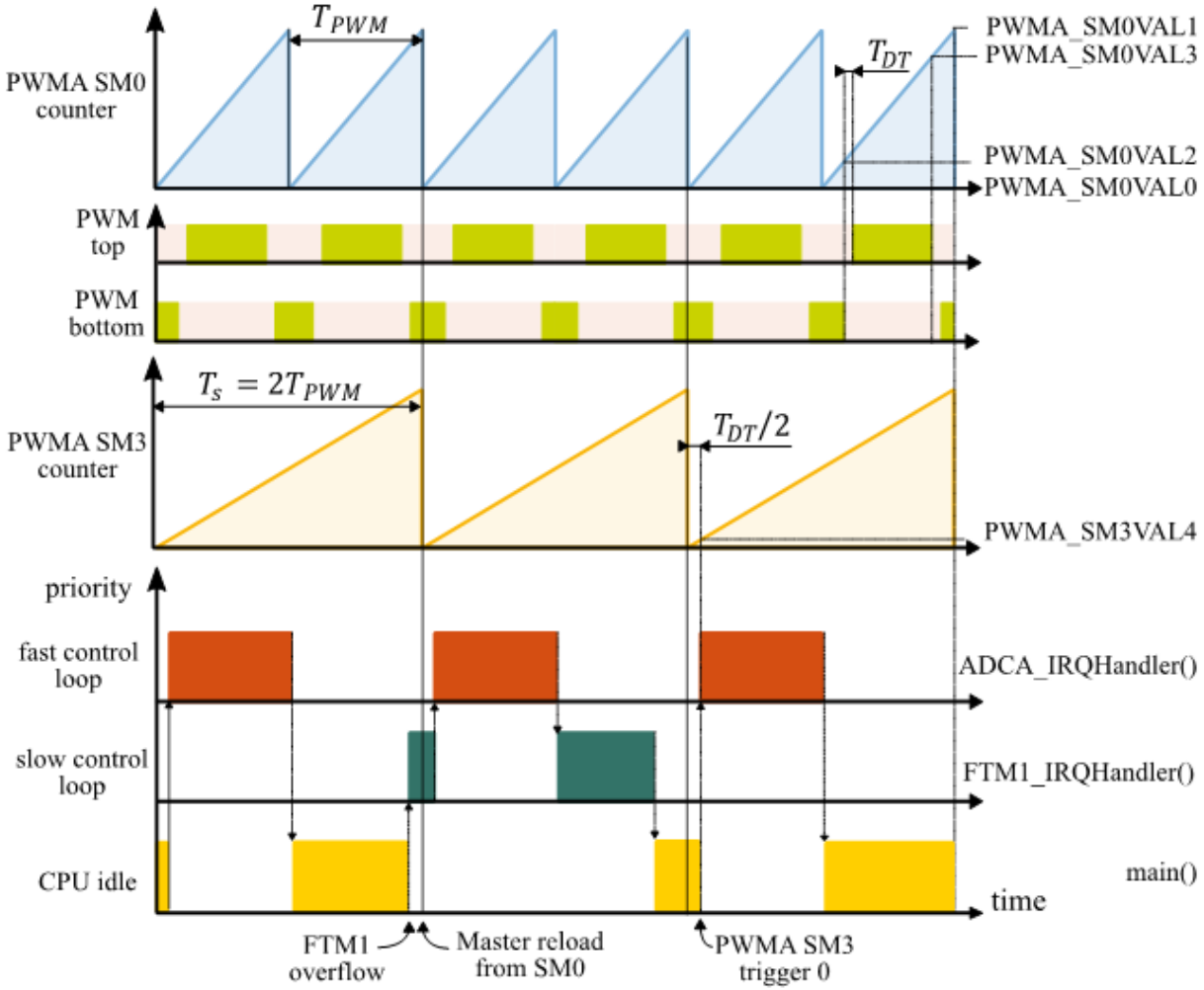

Figure 5. Example of application timing on KV46F

The PWMA Sub-Module 0 (SM0) timer internal counter counts from the PWMA_SM0VAL0 value to the PWMA_SM0VAL1 value with the T_{PWM} 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 T_{DT}, is inserted to avoid a 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 therefore offers the possibility to easily increase the ratio between the FOC sampling period T_S and the PWM period T_{PWM} (see MCDRV configuration). The example in Figure 5 shows the case when T_S is double the T_{PWM}.

The ADCA and ADCB (because both ADC12s run in the triggered parallel mode) are triggered by the PWMA SM3 trigger 0 signal, which is connected to the ADC via XBARA. The trigger is issued when the SM3 internal counter reaches the PWMA_SM3VAL4 value, which is set to $T_{DT}/2$ by default (this value ensures a correct ADC sampling even at a very high duty cycle). The internal counter of SM3 is reloaded by the master reload trigger event from SM0. You can set SM3 to ignore up to first 15 trigger opportunities, which allows to define the sampling period T_S to the PWM period T_{PWM} ratio. ADC12 converts a total of four samples at the beginning of sampling period T_S:

- The first two samples on ADCA (channel 2 for phase A or 3 for phase C) and ADCB (channel 2 for phase B or 3 for phase C) are the samples of phase currents.

- The DC-bus voltage is sampled second by the ADCA channel 1.
- The IPM temperature is sampled second by the ADCB channel 1.

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

The CPU load and memory usage for the ACIM sensorless application software is in Table 2. The results apply to the demonstration application built using the IAR® Embedded Workbench® IDE with the maximum speed optimization. The memory usage is calculated from the linker *.map* file, including the 8-KB FreeMASTER recorder buffer (allocated in RAM) and the 6.1-KB FreeMASTER TSA (Target-Side Addressing) table (allocated in flash). The CPU load was measured using the SysTick timer and calculated according to:

$$CPU\ load = \frac{cycles_{\mathrm{slow}}}{f_{\mathrm{CPU}} T_{\mathrm{Sslow}}} + \frac{cycles_{\mathrm{fast}}}{f_{\mathrm{CPU}} T_{\mathrm{S}}}$$

where $cycles_{slow}$ and $cycles_{fast}$ are the numbers of the CPU cycles measured in the fast and slow loops. $T_{Sslow} = 1\ ms$ is the slow loop sampling period and $T_S = 100\ \mu s$ is the fast loop sampling period.

Table 2. KV46F CPU and memory usage

-	KV46
CPU clock [MHz]	148
Fast Control Loop [cycles] (%)	4098 (27.7%)
Slow Control Loop [cycles] (%)	5010 (3.4%)
Total CPU load [%]	31.1%
Flash usage [B]	26 458
RAM usage [B]	9993

4.1.1 Multi-purpose Clock Generator (MCG) and System Integration Module (SIM)

The MKV46F MCU uses the MCG and SIM modules to configure and distribute the clock across the peripheral modules. The MCG (Multi-purpose Clock Generator) module provides several clock-source options for the MCU. The SIM (System Integration Module) module provides system control and chip configuration. The MCG module configuration is as follows:

- The 8-MHz external oscillator is used as the reference clock source.
- The PLL is used to generate the 148-MHz MCG output clock (MCG_C5[PRDIV] = 0 and MCG_C6[VDIV] = 0x15).

The SIM module is configured as follows:

- The clock is enabled for all peripheral modules used.
- The system clock frequency is 148 MHz (divider SIM_CLKDIV1[OUTDIV1] = 0).
- The fast peripheral clock frequency is 74 MHz (divider SIM_CLKDIV1[OUTDIV2] = 1).
- The flash clock frequency is 24.67 MHz (divider SIM_CLKDIV1[OUTDIV4] = 5).

The MCU clock is set using the Kinetis Software Development Kit (KSDK) version 2.0 clock setup procedure. For more information about Kinetis SDK, see www.nxp.com/KSDK.

4.1.2 FlexTimer (FTM1)

The FTM1 peripheral module is used for the slow control loop timing. The FTM1 module is configured as follows:

- The input clock is set to 4.625 MHz (1/16 of the fast peripheral clock frequency).
- The interrupt with a level-two priority is enabled on the counter reaching the modulo value.

- The modulo is set so that the overflow interrupt occurs with the slow control loop period.

4.1.3 12-bit cyclic Analog-to-Digital Converter (ADC12)

The ADC12 module is used to measure the phase currents, the DC-bus voltage, and 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 24.67 MHz (1/6 of the fast peripheral clock frequency).
- The end-of-scan interrupt with a level-one priority is enabled on ADCA.
- A single-ended, 12-bit conversion with the hardware trigger from PWMA is selected. The triggered parallel conversion is used on both the ADCA and ADCB.
- Only the SAMPLE0, SAMPLE1, SAMPLE8, and SAMPLE9 samples are enabled.

4.1.4 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 f_{PWMin}=*74 MHz* (fast peripheral clock frequency).
- The output PWM frequency is set to $f_{PWM}=1/T_{PWM}$= *10 kHz*. The PWM frequency setup is described in MCDRV configuration. 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.
- A center-aligned, complementary PWM is generated only with a full cycle reload.
- A dead time of T_{DT} *= 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 *=* $T_{DT}f_{PWMin}$ *= 111*.
- Channels A and B at SM0, SM1, and SM2 are disabled on faults number 0 or 1 active, with automatic clearing (the PWM outputs are re-enabled at the first PWM reload after the fault disappears). Fault number 1 (connected to the IPM fault pin via GPIO, active in low) is enabled.

Sub-module 3 is used for the ADC12 triggering with this setup:

- The trigger is issued when the PWMA_SM0VAL4 value is reached ($T_{DT}/2$ by default).
- The 74-MHz fast peripheral input clock is divided by two.
- It is reloaded by the master reload event on the sub-module 0 at every second opportunity. This can be selected by the M1_FOC_FREQ_VS_PWM_FREQ macro (see MCDRV configuration).

4.1.5 Inter-peripheral switch A (XBARA)

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

- The XBARA input IN7 (XBARA_IN7 signal) is connected to output OUT30 (PWMA_FAULT1).
- The XBARA input IN26 (PWMA3_TRG0 signal) is connected to output OUT12 (ADCA_TRIG).

4.1.6 Universal Asynchronous Receiver and Transmitter (UART1)

The UART1 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 115200 bit/s.
- Both the receiver and transmitter are enabled.

- The other settings are set to default.

4.1.7 General-Purpose Input/Output (GPIO)

The following GPIO pins are used:

- Inrush relay control on PTC13
- Braking circuit control on PTC16
- LED state indication on PTB19

Chapter 5
Motor-Control Peripheral Drivers

The Motor-Control Peripheral Drivers (MCDRV) are a simple way of peripheral initialization and access for a 3-phase ACIM control. The features provided by the MCDRV library include 3-phase PWM generation using Space Vector Modulation (SVM) and measurement of the 3-phase current, DC-bus voltage, and IPM temperature (or one general user-defined auxiliary quantity). The principles of the 3-phase current measurement and SVM are described in *Sensorless ACIM Field-Oriented Control* (document DRM150).

The MCDRV consists of two parts:

- The first part is the peripheral configuration module, which is unique for each supported device. The header file includes all MCDRV setup options, including the ADC channel assignment. This part is described in MCDRV configuration.
- The second part consists of the peripheral driver library modules for each supported periphery. All the ADC and PWM periphery drivers share the same API within their class. This enables the higher-level code to be platform-independent, because the peripheral driver function calls are replaced by universally-named macros. The list of supported peripherals and APIs of their drivers is in MCDRV application interface.

5.1 MCDRV configuration

The *mcdrv_hvp-<device>.h* header file provides several options that you can define:

- M1_MCDRV_ADC—this macro specifies the ADC periphery used.
- M1_MCDRV_PWM3PH—this macro specifies the PWM periphery used.
- M1_MCDRV_TMR_SLOWLOOP—this macro specifies the timer for the slow control loop timing.
- M1_PWM_FREQ—the value of this definition sets the PWM frequency in Hz.
- M1_FOC_FREQ_VS_PWM_FREQ—enables you to select a ratio between the sampling period and the PWM period (where M1_FOC_FREQ_VS_PWM_FREQ). This is convenient when the PWM frequency must be higher than the maximum fast-loop interrupt length due to the CPU performance restrictions.
- M1_SLOW_LOOP_FREQ—the value of this definition sets the slow loop period frequency in Hz.
- M1_PWM_PAIR_PH[A..C]—these macros enable a simple assignment of the physical motor phases to the PWM periphery channels or sub-modules. You may alter the order of the motor phases this way. Only the values of 0, 1, and 2 can be assigned to these macros.
- M1_BRAKE_[SET, CLEAR]—DC-bus brake circuit control macro.
- M1_ADC[0,1]_PH_[A..C]—these macros serve to assign the ADC channels for the phase-current measurement (the unassigned ADC channels are set to the ADC_NO_CHAN value). The general rule is that at least one of the phase currents must be measurable on both ADC converters and the remaining two phase currents must be measurable on different ADC converters. If this rule is broken, a pre-processor error is issued. The reason for this rule is that to ensure a proper ADC measurement in a wide range of the PWM duty cycle, the selection of the phase-current pair to measure depends on the current SVM sector. For more information about the 3-phase current measurement, see *Sensorless ACIM Field-Oriented Control* (document DRM150).
- ADC[0,1]_UDCB and 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 the auxiliary quantity must be measurable on different ADC converters, so that the measurement can be done simultaneously. If this rule is broken, a pre-processor error is issued during the software build.

5.2 MCDRV application interface

The ADC and PWM motor-control drivers share the same API within their class. To ensure device independency on the MCDRV API, all driver functions are accessible through universally-named macros in the *mcdrv_hvp-<device>.h* file.

5.2.1 ADC control API description

The initialization macros are used to assign I/O variables (for example; to store the measurement results to the variables in your application). These macros are defined:

- M1_SET_PTR_I_ABC(*var*)—assigns a pointer to the GMCLIB_3COOR_T_F16 structurevariable *var*, in which you want to store the phase current measurement results. The GMCLIB_3COOR_T_F16 datatype is defined in the Real-Time Control Embedded Software Libraries (RTCESL). For more information, see www.nxp.com/rtcesl.
- M1_SET_PTR_U_DC_BUS(*var*)—assigns a pointer to the 16-bit fractional variable *var*, in which you want to store the DC-bus voltage measurements.
- M1_SET_PTR_AUX_CHAN(*var*)—assigns a pointer to the 16-bit fractional variable *var*, in which you want to store the auxiliary quantity measured values.
- M1_SET_PTR_SECTOR(*var*)—assigns a pointer to the 16-bit unsigned integer variable *var* that contains the number of the current SVM sector.

NOTE

These macros must be executed before calling any MCDRV ADC functions. Otherwise, your application goes to a hard fault.

These functions are available:

- *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 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 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 it 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, the DC-bus voltage, and the auxiliary quantity and stores them in the variables defined by the user in the initialization macros (see above). This function always returns *true*.

5.2.2 PWM control API description

The initialization macros are used to assign the I/O variables (for example; to set the required duty cycles from your application). These macros are defined:

- M1_SET_PTR_DUTY(*var*)—sets the pointer to the GMCLIB_3COOR_T_F16 structure variable *var*, in which you define the required phase PWM duty cycles. The GMCLIB_3COOR_T_F16 datatype is defined in RTCESL.

NOTE

This macro must be executed before calling any MCDRV PWM functions. Otherwise, your application goes to a hard fault.

These functions are available:

- *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 variable defined by the user in the initialization macros (see above). 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 and automatically clears the state of the over-current fault flags. This function returns *true* when an over-current event occurs. Otherwise, it returns *false*.

Chapter 6
FreeMASTER user interface

This section provides information about the tools and recommended procedures to control the sensorless ACIM 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.

6.1 Remote control using FreeMASTER

The remote operation is provided by FreeMASTER via the USB interface. FreeMASTER 2.0 (or higher) is required for the application to operate properly. Download the up-to-date version of FreeMASTER at www.nxp.com/freemaster.

Perform these steps to control an ACIM motor using FreeMASTER:

1. Open the FreeMASTER file located in *pack_motor_(board)\middleware\motor_control\freemaster\acim.pmp*. All projects use the TSA by default, so it is not necessary to select a symbol file for FreeMASTER (see FreeMASTER TSA and user variables addition to FreeMASTER watch).
    - Click the communication button (the green GO! button in the top left-hand corner, as shown in Figure 6) to establish the communication.

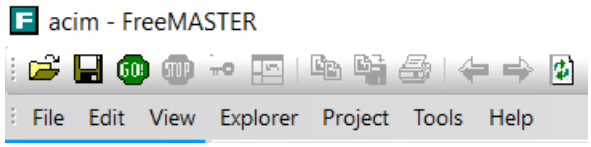

Figure 6. Green GO! button placed in top left-hand corner

    - ACIM Control Reference Solution Package, User's Guide, Rev. 3, 01/2017 NXP Semiconductors 23—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" (see Figure 7). Otherwise, a FreeMASTER warning pop-up window appears.

Figure 7. Example FreeMASTER communication established successfully

2. Control the AC induction motor using the control page or MCAT.

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

1. Go to the "Project→Options→Comm" tab and make sure that "OpenSDA" is set in the "Port" option and the communication speed is set to 115200 bit/s.

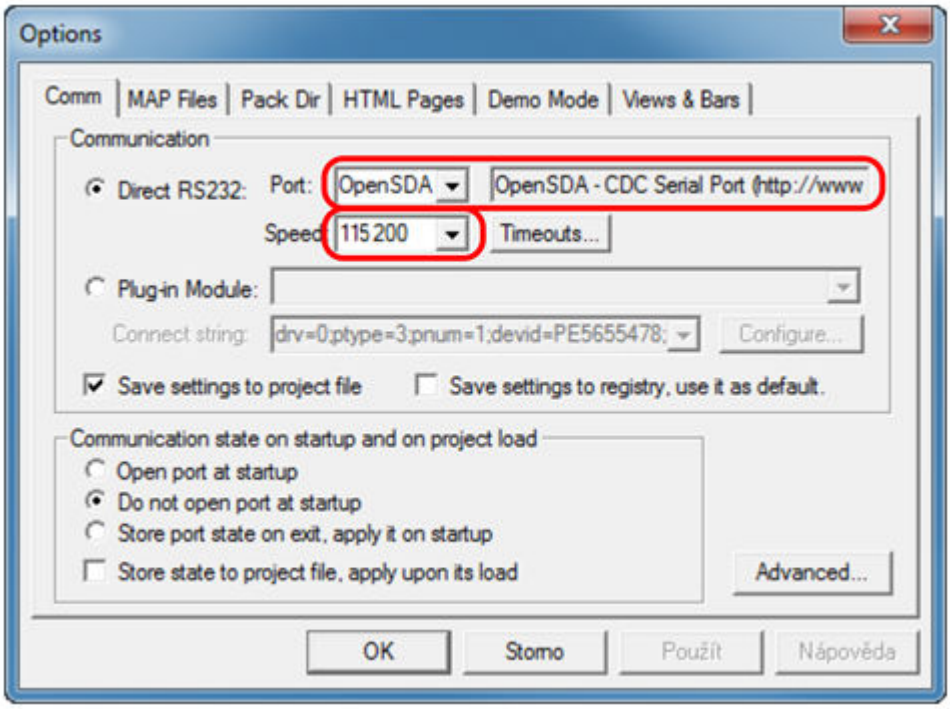

Figure 8. FreeMASTER communication setup window

2. If “OpenSDA-CDC Serial Port” is not printed out in the message box next to the “Port” dropdown menu, unplug and then plug in the USB cable and reopen the FreeMASTER project.
3. 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.

6.1.1 FreeMASTER TSA and user variables addition to FreeMASTER watch

By default, all projects use TSA (Target Side Addressing). This means that the information about the variables’ address and size are stored in the MCU flash memory. Only the variables necessary for the MCAT functionality are stored in the TSA. Only these variables are visible in FreeMASTER. If you want to monitor your own variables, provide a symbol file that contains the information about the addresses of all variables in the project to FreeMASTER. The symbol files are generated during the build process to the *\boards\(board_name)\demo_apps\mc_acim\<compiler>/<debug or release>* folder. For more information about the TSA, see *FreeMASTER Serial Communication Driver* (document FMSTERSCIDRVUG).

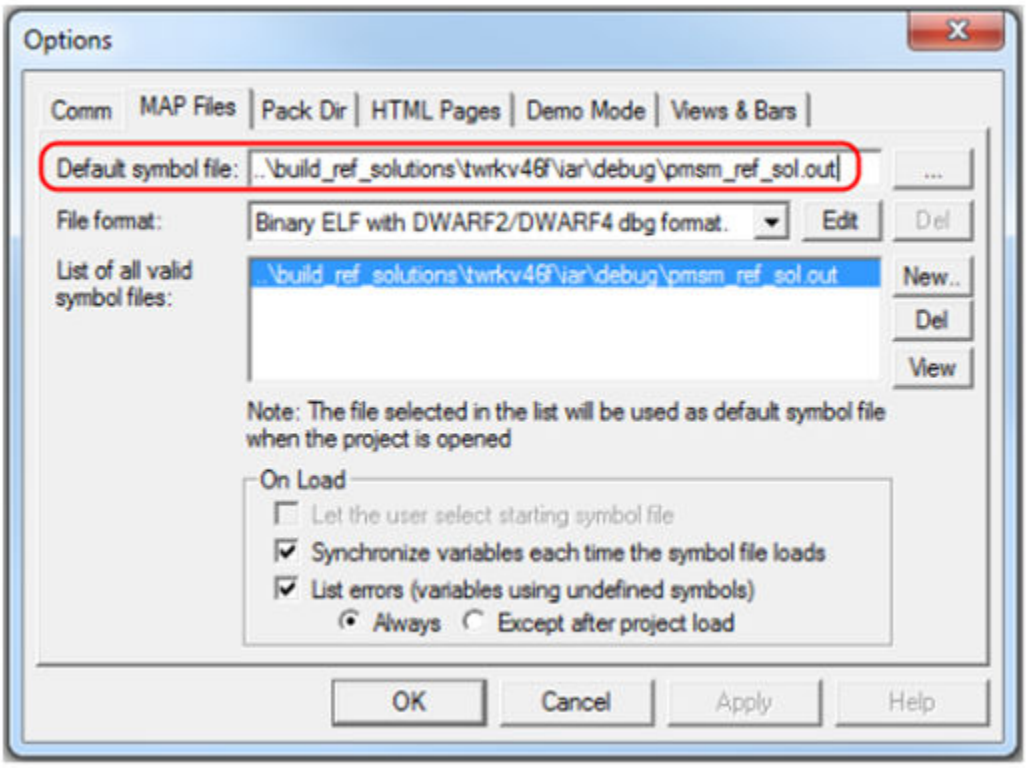

Figure 9. FreeMASTER MAP Files tab

Chapter 7
Tuning and controlling the application

This section provides information about the tools and recommended procedures for controlling the sensorless MCRSP for ACIM application. As the primary means of communication, the application contains an embedded-side driver of the FreeMASTER real-time debug monitor and data visualization tool for communication with the PC. FreeMASTER supports non-intrusive monitoring, as well as modifying of the target variables in real time, which is very useful for algorithm tuning. Besides the target-side driver, FreeMASTER requires installing the PC application as well. For more information, see www.nxp.com/freemaster.

The ACIM sensorless FOC application can be easily tuned using the Motor Control Application Tuning (MCAT) page for ACIM. The MCAT for ACIM is a user-friendly modular page, which runs within the FreeMASTER PC application. To launch it, execute the *.pmp* file. When the communication with the MCU side of the application is established, the MCU platform is detected and a proper MCAT setup is used. Without a connection, many features are disabled and the pertinent files are generated next to the *.pmp* file. See FreeMASTER user interface. Figure 10. shows the MCAT for ACIM welcome page. The tool consists of the tab menu (point one), the tuning experience level selector (point two), the detected platform (point three), and the tab content itself (point four). Each tab represents one sub-module, which enables you to tune and control different aspects of the application:

- *Introduction*—welcome page with the ACIM sensorless FOC diagram and a short description of the application.
- *Motor Identif*—ACIM semi-automated parameter-measurement control page. The ACIM parameter identification is described in ACIM parameter identification.
- *Parameters*—this page enables you to modify the motor parameters, the specification of the hardware and application scales, and the fault limits. For more information, see Input Application Parameters tab.
- *Current Loop*—specify the current loop PI controller gains, output limits, and default *d*-axis stator current reference here. For more information, see Current loop tuning.
- *Speed loop*—this tab contains fields to specify the speed controller proportional and integral gains, as well as the output limits, the parameters of the speed ramp, and the startup procedure. For more information, see Speed loop tuning.
- *Flux loop*—this tab is used to set up the *d*-axis current control, which includes the Max-Torque Per Ampere (MTPA) and Field-Weakening (FW) algorithm settings. For more information, see Flux loop tuning.
- *Sensorless*—this page enables you to tune the parameters of the Rotor Flux Observer (RFO) for the rotor flux position estimator and the Model-Reference Adaptive System (MRAS) speed observer. For more information, see Sensorless rotor flux position and speed estimation.
- *Control Struc*—the application control page enables you to choose between the scalar control (also known as Volt per Hertz or V/Hz) and FOC, where you can disable parts of the FOC cascade structure for tuning purposes. This tab enables you to set the required speed, the stator currents, and the stator voltage. It also provides information about the application state. For more information, see Application control using MCAT.
- *Output file*—this tab enables you to view all the calculated constants that are required by the ACIM sensorless FOC control algorithms and to generate a new *m1_acim_appconfig.h* application configuration header file. For more information, see MCAT output file generation.
- *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.

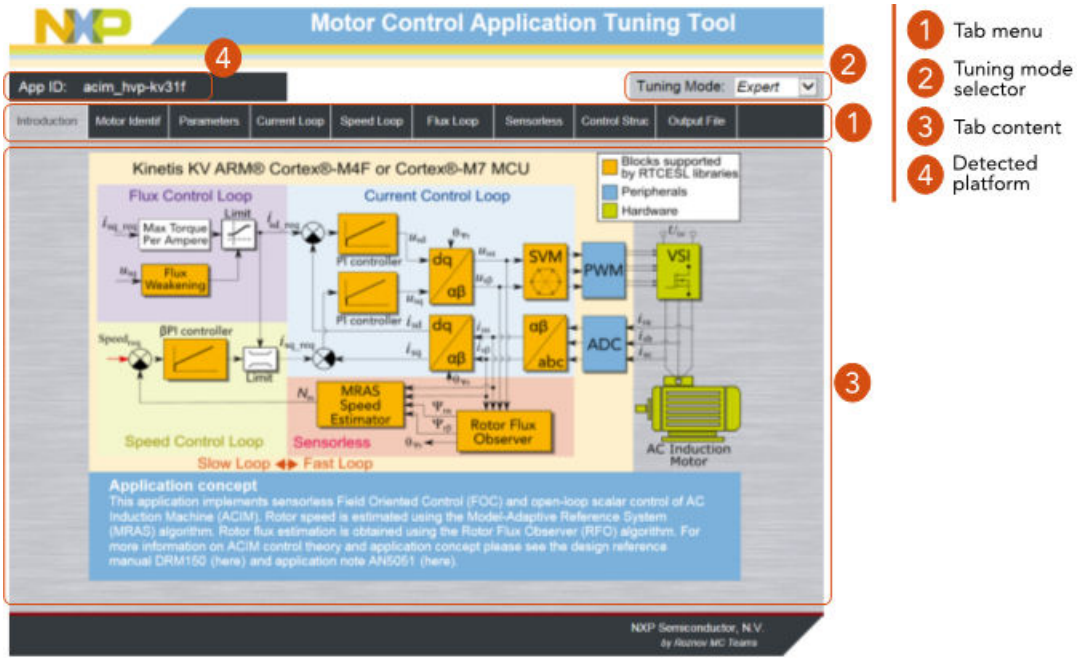

Figure 10. MCAT for ACIM welcome page

Most of the 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 *Store Data* (or *Reload Data*) button. The data stored using the *Store Data* button is automatically loaded the next time the MCAT is launched and the MCU communication is established. For more information about the application states, see *Sensorless ACIM Field-Oriented Control* (document DRM150).

The *basic* and *expert* tuning modes are available. Selecting the latter one grants you the access to modify all parameters and fields available in the MCAT. Using the *expert* mode is not recommended for inexperienced users. When the MCAT operates in the offline mode, the *App Id* line reads *offline*. When the communication with the target MCU is established using a correct software, the *App Id* line displays the correct platform name and all stored parameters for the given MCU are loaded.

Besides the MCAT page for ACIM, several scopes, recorders, and variables in the variable watch window are pre-defined in the FreeMASTER project file to further simplify the motor parameter tuning and debugging.

The following sections provide simple instructions on how to identify the parameters of the connected ACIM, and how to tune the application.

7.1 ACIM parameter identification

Because the model-based control methods of the ACIM drives are the most effective and usable, obtaining an accurate model of a motor is an important part of the drive design and control. The machine parameters required by the FOC can be classified as either electrical or mechanical parameters.

For the electrical parameters, it is necessary to know the values of stator resistance R_S, magnetizing inductance L_m, leakage stator inductance L_{sl}, leakage rotor inductance L_{rl}, and rotor resistance R_r. An equivalent steady-state circuit for one phase of an induction motor is shown in Figure 11. While the stator resistance R_S can be obtained by a simple DC measurement, the other parameters require a more complex approach. The most common identification methods of the ACIM parameters are based on the no-load and blocked-rotor tests. The ACIM sensorless control software contains parameter-identification algorithms that employ these methods as well. These algorithms also enable you to perform the power stage characterization, which allows to compensate for the inverter nonlinearity.

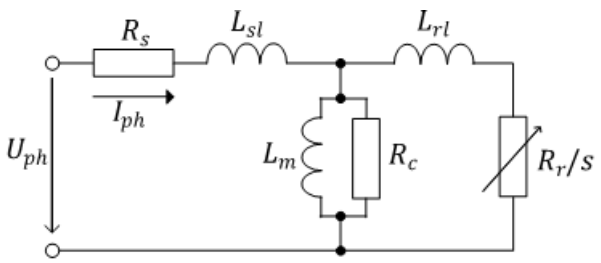

Figure 11. ACIM equivalent circuit

There are two mechanical parameters that are important for the speed-loop controller tuning. They are the moment of inertia J and the viscous friction B, which characterize the mechanical equation:

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

where ω_m is the mechanical angular speed, T is the torque generated by the machine, and T_{load} is the loading torque. To identify the moment of inertia and the viscous friction, both the speed and the torque on the shaft must be estimated or measured. The identification is usually conducted during acceleration or deceleration with a known torque, because the moment of inertia can be only detected when the speed is changing.

This section explains the motor-identification theory, as well as the implementation of these algorithms in the ACIM sensorless control software, including the MCAT identification page description.

7.1.1 Power stage characterization

All VSIs introduce non-linear error voltage U_{err} to their output. This parasitic effect is caused by the current-clamping effect, the dead time, and the transistor voltage drop. It depends on phase current i_{ph}, DC-bus voltage U_{DCbus}, and dead time T_{DT}. The error voltage U_{err} dependency on the phase current is measured during the power stage characterization process. An example of the inverter voltage error characteristic is shown in Figure 12. Such characteristic is then used by the motor-control application to linearize the output voltage. This is especially important in the case of sensorless control.

The power stage characterization can be done through the MCAT *Motor Identif* tab (see Parameter measurement process). To perform the characterization, connect a motor with a known stator resistance and set this value in the *Calib Rs* field. The other parameter that you must specify is the calibration range $I_{ph,cal}$ of the stator phase current i_{ph} in the *Calib range* field. The range must be set so that the non-linearity of the error voltage (the knee of the curve in Figure 12) is captured. Start the characterization by pressing the *Calibrate* button. A total of 65 points are measured in the range ($-I_{ph,cal}$,$I_{ph,cal}$). Each measurement takes 300 ms, so the process takes about 20 s and the motor must withstand this load. The acquired characterization data can be saved to a file using the *Generate Calibration Data File* button (point two in Figure 16).

The power stage characterization is necessary only for non-NXP hardware boards. If you use NXP power stages with the sensorless ACIM application, you can omit the characterization process, because the calibration data file is already generated.

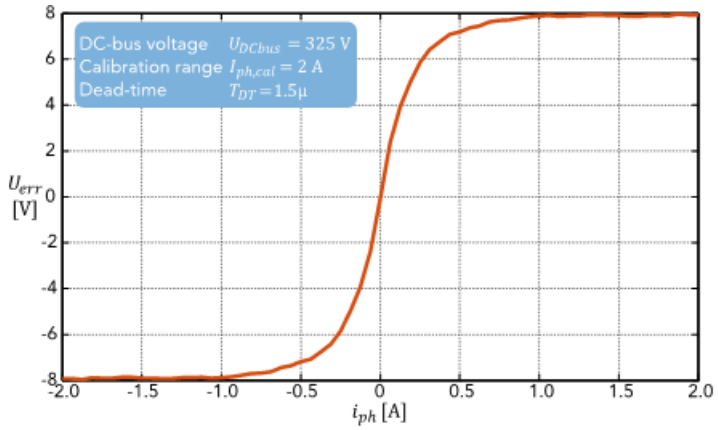

Figure 12. Example power stage characteristic

7.1.2 Stator resistance measurement

The stator resistance R_S is measured with the DC current value $I_{ph,DC}$ (equal to the nominal stator current amplitude by default), which is applied to the motor for 1200 ms. The DC phase voltage $U_{ph,DC}$ is kept using the current controllers. The current controller parameters are selected conservatively, so that stability is assured. The stator resistance R_S is calculated using the Ohm's law:

$$R_s = \frac{U_{ph,DC}}{I_{ph,DC}} \quad [\Omega]$$

7.1.3 No-load test

The main goal of the no-load test is to determine the parameters of the transverse branch of the equivalent circuit (the core-loss resistance R_C and the magnetizing reactance $X_m = j\omega L_m$ X_m=jωL_m). The no-load conditions mean that the motor runs at the rated frequency f_N, phase voltage U_N, and without an external load. The machine rotates at an almost synchronous speed and only little power is drawn from the power supply. The slip is close to zero, which means that the impedance of the rotor loop in the equivalent circuit is very high and you can ignore the entire rotor loop, as shown in Figure 13. The results of the no-load test are the no-load phase input active power P_{ph0}, the no-load phase input reactive power Q_{ph0}, and the no-load phase current I_{ph0}.

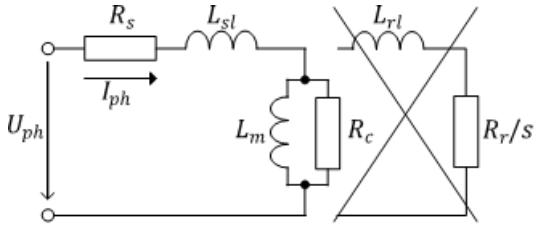

Figure 13. Equivalent circuit for no-load test

7.1.4 Blocked rotor test

The blocked-rotor test provides information about the parameters of the longitudinal branch of the equivalent circuit, such as the stator and rotor resistances and the stator and rotor reactances $X_S = j\omega L_S$ and $X_r = j\omega L_r$. When the rotor of an ACIM locks up, the slip is equal to one and the rotor resistance R_r is much lower than the core resistance R_C. You can ignore the transverse branch of the equivalent circuit. The unity slip means that all energy supplied to the motor is converted to heat. The measurement is therefore made at a reduced supply voltage U_{phL} so that the steady-state phase current reaches only the rated value I_{phN}. The

measurement is carried out as quickly as possible to prevent errors caused by the rotor and stator windings heating. The results of a blocked-rotor test are the load phase voltage U_{phL}, the load phase input active power P_{phL}, and the load phase input reactive power Q_{phL}.

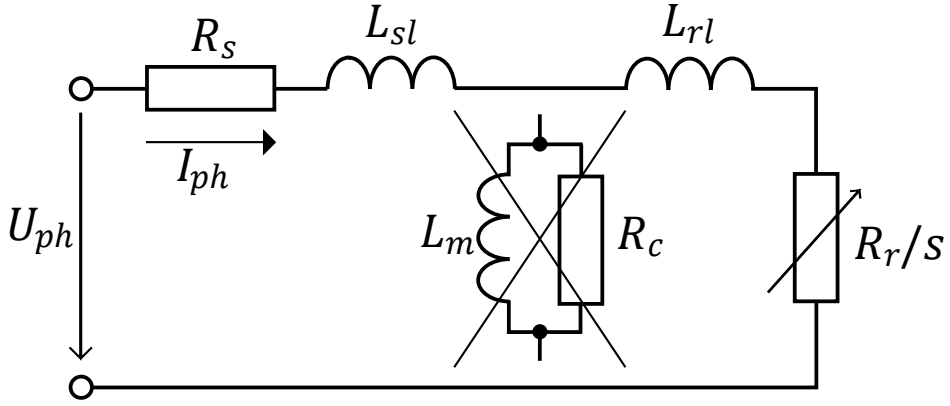

Figure 14. Equivalent circuit for blocked-rotor test

7.1.5 Calculation of electrical parameters

The no-load test results are used to calculate the combined stator and magnetic circuit resistance R_0 and the combined stator and magnetic circuit reactance X_0 as follows:

$$R_0 = \frac{P_{ph0}}{I_{ph0}^2} \quad [\Omega]$$

$$X_0 = \frac{Q_{ph0}}{I_{ph0}^2} \quad [\Omega]$$

The blocked-rotor test results are used to calculate the total resistance R_L and the total reactance X_L:

$$R_L = \frac{P_{phL}}{I_{phL}^2} \quad [\Omega]$$

$$X_L = \frac{Q_{phL}}{I_{phL}^2} \quad [\Omega]$$

The stator and rotor leakage reactances X_S and X_r are considered equal and calculated as:

$$X_s = X_r = \frac{X_L}{2} \quad [\Omega]$$

The magnetizing reactance is:

$$X_m = X_0 - X_s \quad [\Omega]$$

This leads to the magnetizing inductance:

$$L_m = \frac{X_m}{2\pi f_N} \quad [\mathrm{H}]$$

The stator and rotor single phase leakage inductances L_{Sl} and L_{rl} are calculated as:

$$L_{sl} = \frac{X_s}{2\pi f_N} \quad [\mathrm{H}]$$

$$L_{rl} = \frac{X_r}{2\pi f_N} \quad [\mathrm{H}]$$

The stator and rotor single-phase inductances L_S and L_r are calculated as:

$$L_s = L_{sl} + L_m \quad [\mathrm{H}]$$

$$L_r = L_{rl} + L_m \quad [\mathrm{H}]$$

The last parameter needed is the rotor resistance R_r referred to the stator, which is calculated as:

$$R_r = (R_L - R_s)\left(\frac{X_r + X_m}{X_m}\right)^2 \quad [\Omega]$$

7.1.6 Mechanical parameter measurement and calculation

As explained in ACIM parameter identification, it is necessary to know the moment of inertia J and the viscous friction B to tune the speed controller loop. The parameters can be identified using equation Eq. 2 during the speed acceleration test, with a known generated and loading torque.

NOTE

If using a sensorless algorithm, the mechanical parameters estimation is affected by the accuracy of the speed and torque estimations.

The ACIM identification software uses the torque profile, as shown in Figure 15. The loading torque is (for the purpose of simplicity) said to be zero during the whole measurement and 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 the nominal speed in time t_1. These integrals are calculated during the time period from t_0 (speed estimation is accurate enough) to t_1:

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

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

During the second phase, the rotor decelerates freely with no generated torque, only by friction. This allows 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 equation Eq. 2 :

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

The moment of inertia is as follows:

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

The viscous friction is then derived from the relation between the mechanical time constant and the moment of inertia.

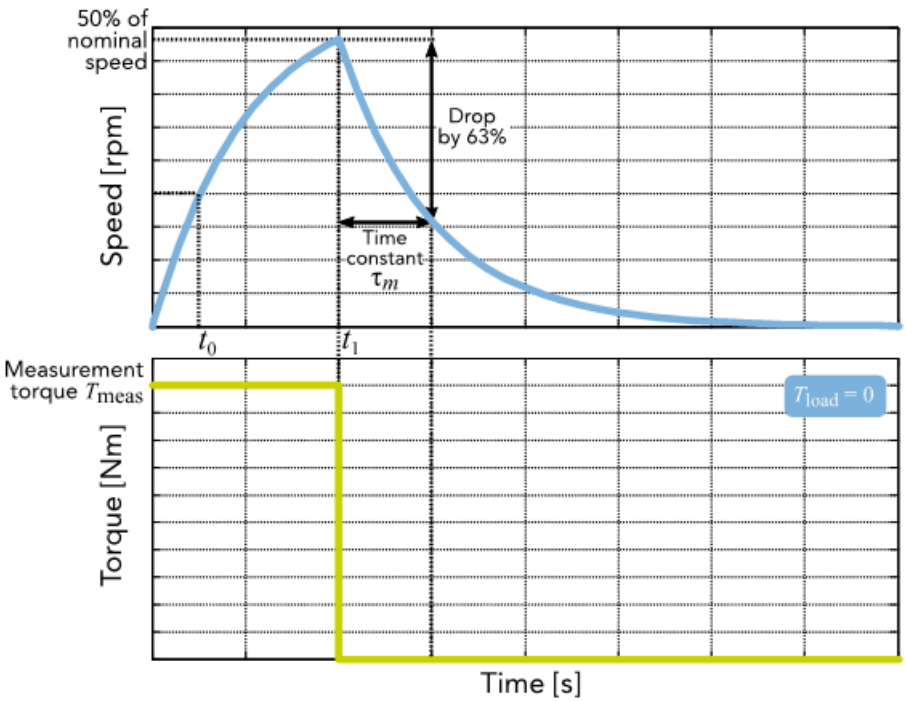

Figure 15. Mechanical parameter measurement process

7.1.7 Parameter measurement process

You can control and set up the motor identification process using the MCAT *Motor Identif* tab, which is shown in Figure 16. After filling in the motor label information (point three) and selecting the mechanical parameter measurement torque and current and speed loop bandwidth (point four), start the measurement by clicking the *Measure* button (point five). A flowchart of the measurement process is shown in Figure 18. When the measurement is complete, the results appear on the right side of the screen (point six). To apply the measured FOC algorithm parameters, click the *Apply parameters on target* button. If the results are satisfactory, click the *Store data in MCAT* button to update the motor parameters in MCAT and continue with fine-tuning the application in the other MCAT tabs. Otherwise, you can return to the previous parameters using the *Restore parameters in target* button.

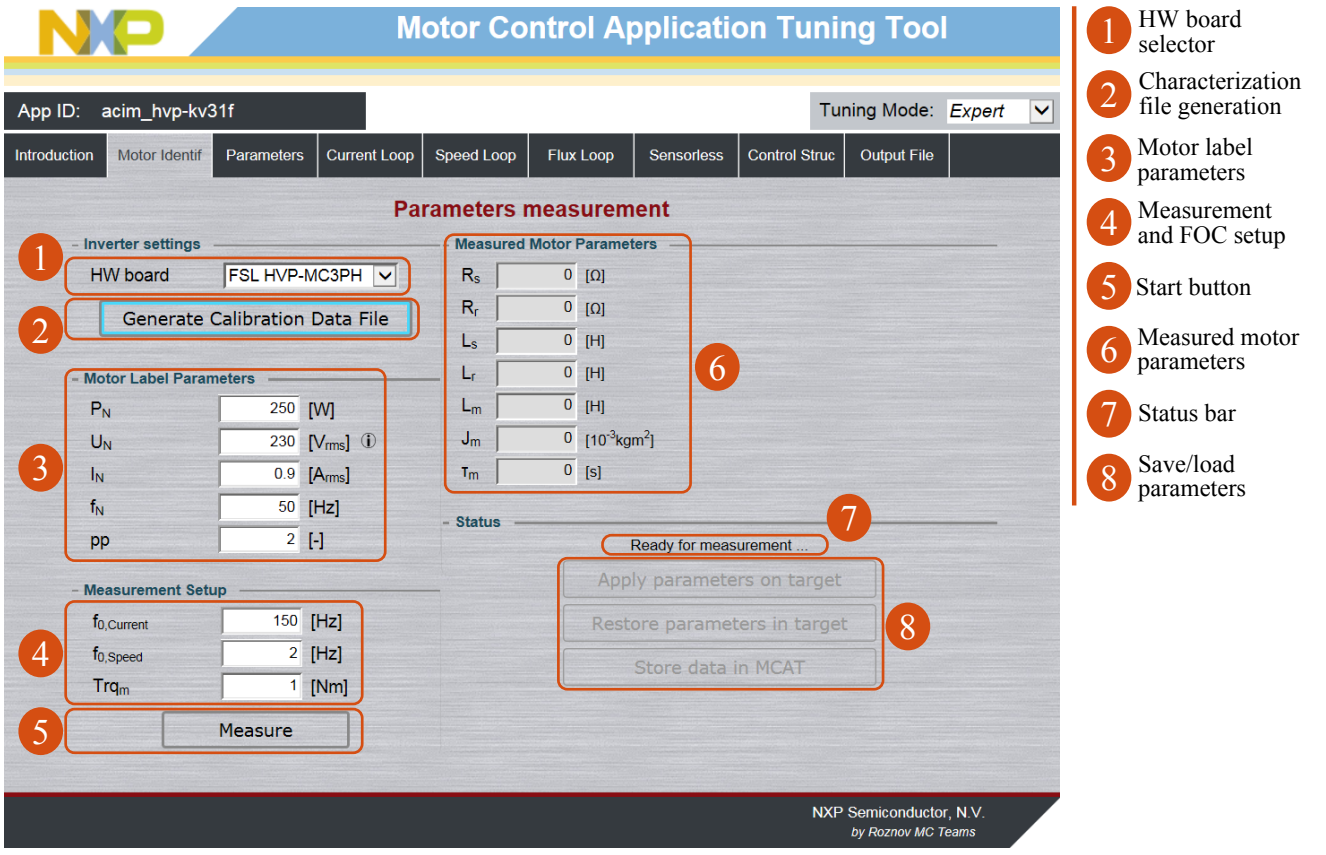

Figure 16. MCAT ACIM parameters identification page

There are several faults and warnings that can occur during the measurement or calibration processes. Do not confuse the measurement faults with the application faults, such as the DC-bus under-voltage (see *Sensorless ACIM Field-Oriented Control* (document DRM150)). The measurement faults are listed in Table 3, together with their sources and possible troubleshooting. If any of these faults occur, the identification process ends immediately:

Table 3. Measurement faults and their description

Fault number	Fault description	Fault source	Troubleshooting
1	User abort	Measurement aborted by user	—

Table continues on the next page...

Table 3. Measurement faults and their description (continued)

Fault number	Fault description	Fault source	Troubleshooting
2	Motor not connected	i_{ph}>50mAcannot be reached using the available DC-bus voltage	Check that a motor is connected
3	R_S too high for calibration	$i_{ph,cal}$ could not be reached using the available DC-bus voltage	Use a motor with a lower R_S for the power stage characterization
4	Mechanical measurement timeout	Mechanical measurement takes too long	Repeat the measurement process with a different setup

Unlike faults, warnings do not stop the identification process, but inform you of the possible problem sources. Warnings can often be ignored. The warnings that can occur during the measurement process are described in Table 4.

Table 4. Measurement warnings and their description

Warning number	Warning description	Warning source	Troubleshooting
1	R_S measurement current $I_{ph,DC}$ not reached	The defined $I_{ph,DC}$ was not reached, so the measurement was taken with a lower value	Raise the DC-bus voltage to reach the $I_{ph,DC}$, or lower the I_{phN} to avoid this warning
2	No-load test voltage U_{phN} not reached	User-defined U_{phN} was not reached, so the measurement was taken with a lower value	Raise the DC-bus voltage to reach the U_{phN}, or lower the value to avoid this warning
3	Blocked-rotor test current I_{phN} not reached	User-defined I_{phN} was not reached, so the measurement was taken with a lower value	Raise the DC-bus voltage to reach the I_{phN} or lower the value to avoid this warning
4	Low precision of the R_S measurement	The DC measurement voltage and current were low and the calculated value might not be precise	Raise the I_{phN} value to avoid this warning (beware of overloading the motor)

To access the expert settings of the measurement algorithms, navigate to the *mid_def.h* file in your ACIM application. In that file, you can change various well-commented definitions in case of measurement failures that cannot be troubleshot.

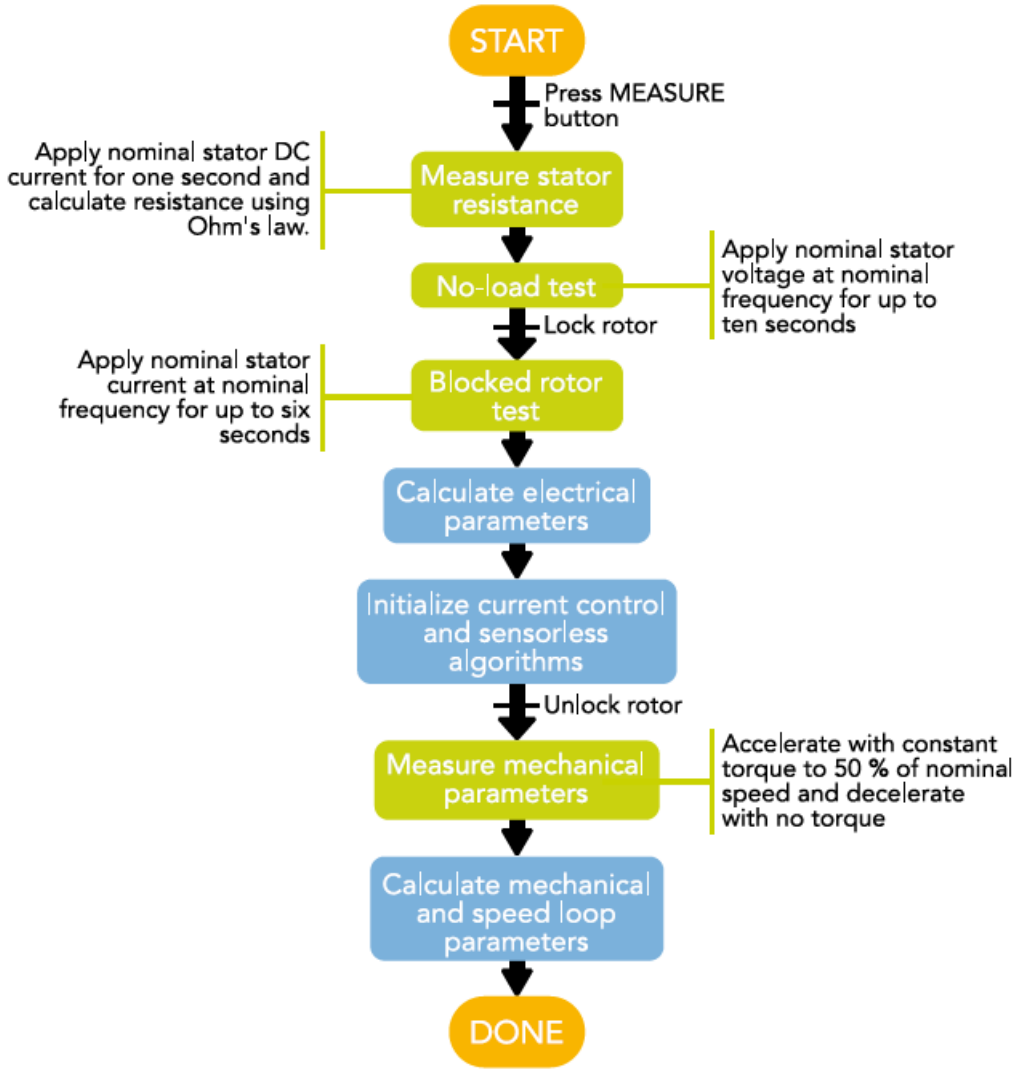

Figure 17. Parameter measurement flowchart

7.2 Application control using MCAT

Control the application using the *Control Struc* tab, which is shown in Figure 18. The application state control area on the left-hand side of the screen (points one and two) shows the current application state and enables switching the main application switch on or off (turning the running application off disables all PWM outputs). The *Cascade Control Structure* area is placed on the right-hand side of the screen (points three to six). Here you can choose between the scalar and FOC control using the appropriate buttons. Enable the selected parts of the FOC cascade structure by selecting *Voltage FOC*, *Current FOC*, or *Speed FOC*. This is useful for application tuning and debugging.

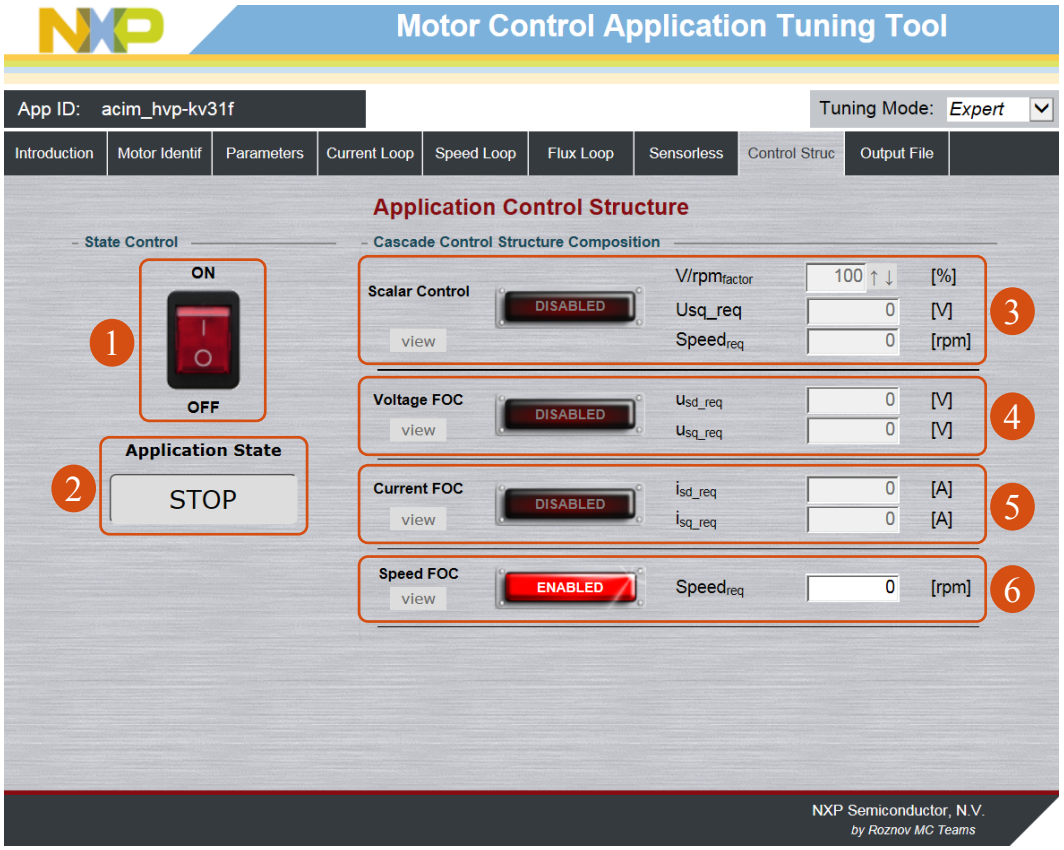

Figure 18. MCAT for ACIM control page

The scalar control diagram is shown in Figure 19. It is the simplest type of an ACIM control strategy. The ratio between the magnitude of the stator voltage and the frequency (frequency information is contained in the $Speed_{req}$ value) is kept at the nominal value, which results in a nominal flux amplitude. This control method is sometimes called Volt per Hertz (V/Hz). The position-estimation Rotor Flux Observer (RFO) algorithm is running in the background to enable the RFO tuning.

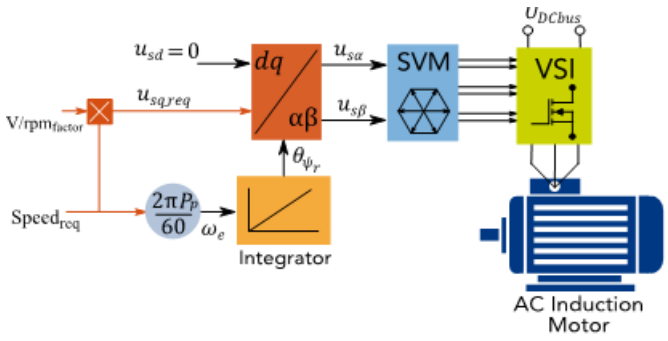

Figure 19. Scalar control mode

The block diagram of the *Voltage FOC* is shown in Figure 20. Unlike V/Hz, the position feedback is closed using the RFO algorithm and the stator voltage magnitude is not dependent on the motor speed. Specify both the *d*-axis and *q*-axis stator voltages using the u_{sd_req} and u_{sq_req} fields. This control method is useful for the RFO tuning as well.

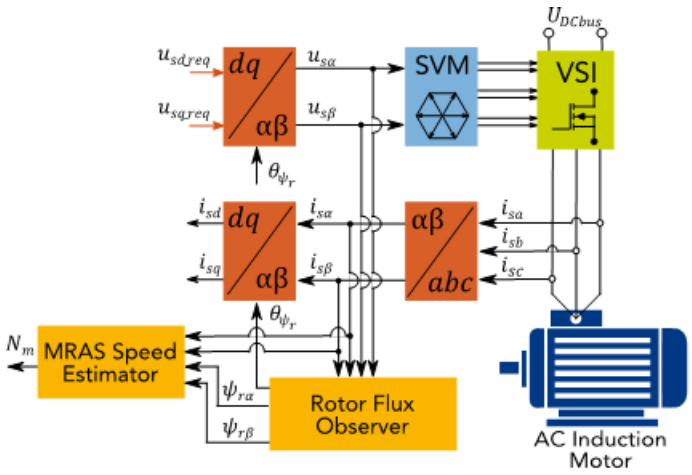

Figure 20. Voltage FOC control mode

The *Current FOC* (torque) control requires the rotor position feedback as well as the currents transformed into the rotor flux frame. Control the motor using the reference variables i_{sd_req} and i_{sq_req}, as shown in Figure 21. The *d*-axis current component i_{sd_req} generates the rotor flux, while the *q*-axis current component of the current i_{sq_req} generates the torque for the motor to run. Change the polarity of the i_{sq_req} current to change the rotation direction. The *Current FOC* control structure can be used for the current controller tuning, provided that the RFO is tuned correctly.

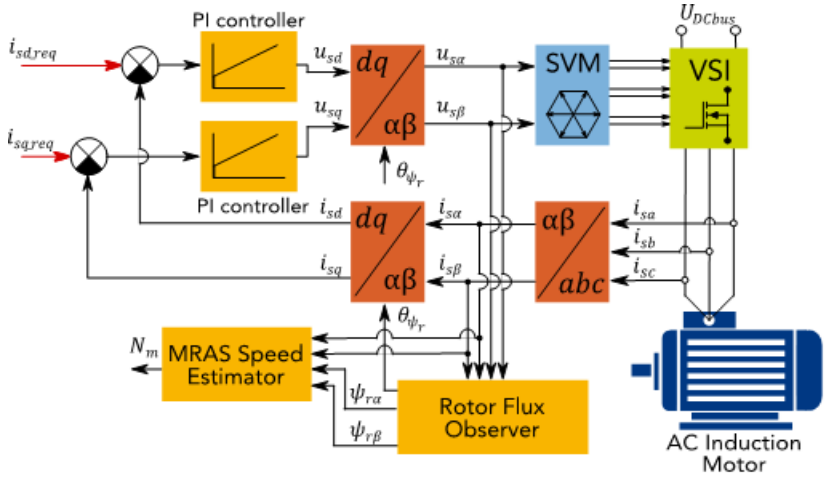

Figure 21. Current FOC (torque) control mode

The full ACIM sensorless FOC is activated by enabling the *Speed FOC* control structure. The block diagram is shown in Figure 22. Two outer control loops were added when compared to the *Current FOC*. The speed loop contains the PI controller, which controls the rotor speed and sets the *q*-axis current i_{sq_req}. The flux loop contains the Max Torque Per Ampere (MTPA) and Flux Weakening (FW) algorithms, which set the *d*-axis current i_{sd_req} to optimize the power efficiency and allow the motor to run at a speed that is higher than nominal. To run a motor at the required speed, simply enter the required value into the $Speed_{req}$ field. This control scheme is used for the speed PI controller and the flux loop design (see Speed loop tuning and Flux loop tuning), which is the final stage of the ACIM sensorless application tuning.

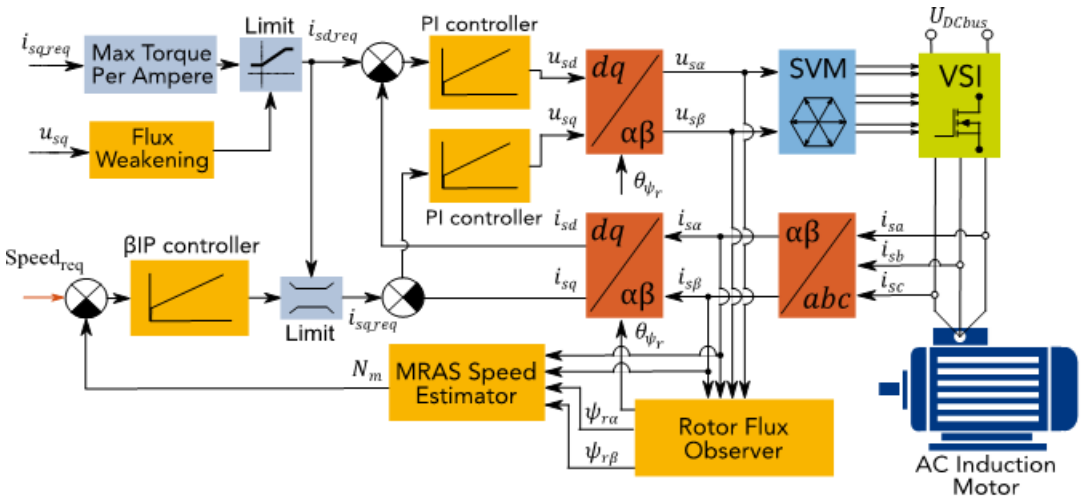

Figure 22. Speed FOC control mode

7.3 Application tuning using MCAT

The ACIM sensorless FOC algorithm tuning is described in this section. The flowchart of the complete process of connecting and running a new ACIM is shown in Figure 23. The first step of acquiring the motor parameters using the identification algorithms is described in ACIM parameter identification. The control of the ACIM sensorless FOC application using MCAT is described in Application control using MCAT. The subsequent steps, including the tuning of the sensorless Rotor Flux Observer (RFO), current loops, speed loop, and the flux loop, are described in the following sections. Only the expert MCAT tuning mode is described. When in the basic mode, omit the grayed-out input fields and leave them at their pre-defined values. Most of the input field labels in the MCAT also show a short description of the item and the maximum range of input parameters when you hover over them with the mouse cursor.

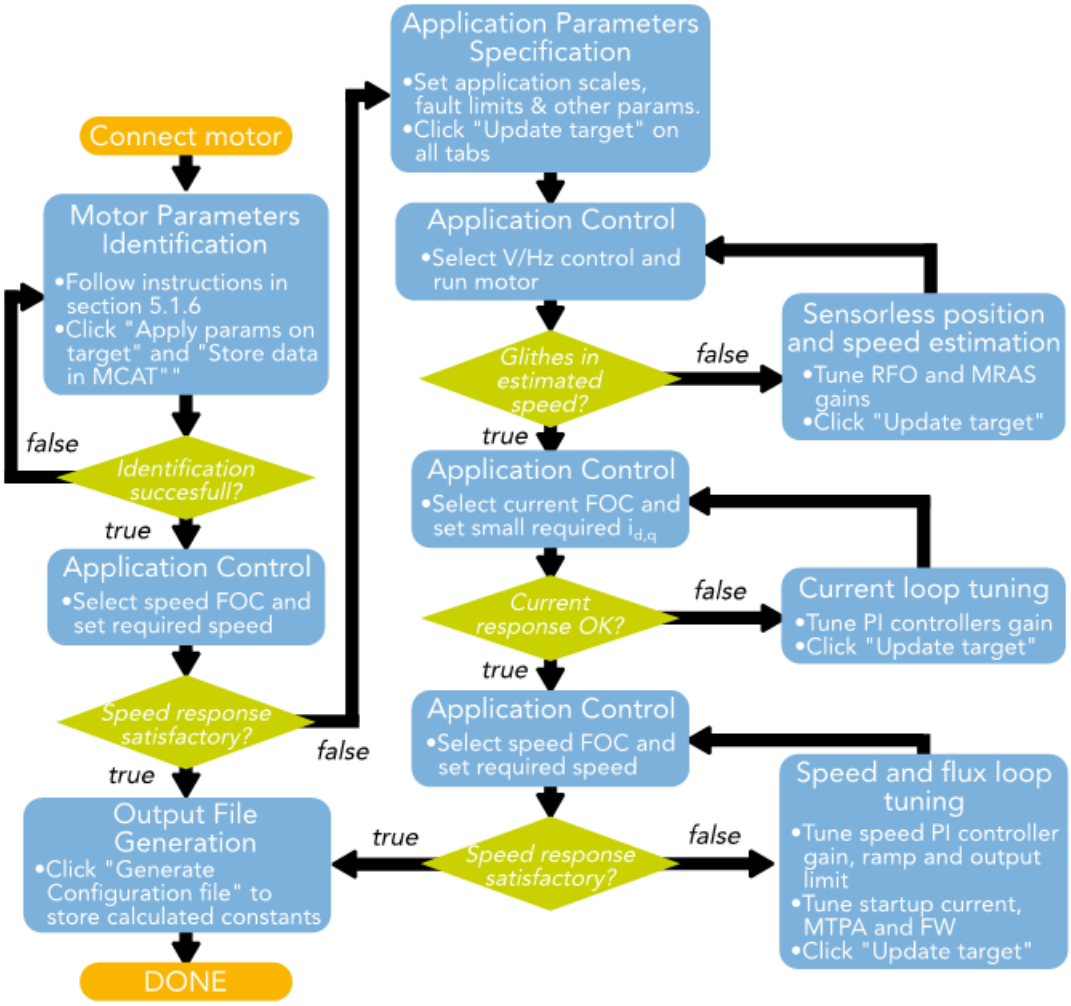

Figure 23. Running a new ACIM flowchart

7.3.1 Input Application Parameters tab

When the parameters of a connected ACIM are obtained using the identification algorithms or simply known before, navigate to the *Parameters* tab, as shown in Figure 24. On the left side, you can modify the motor parameters (point one) and the hardware board scales (point two). Do not change the latter one unless using a user-specific hardware. The right side contains the *Fault Limits* area (point three), which is accessible only in the expert mode.

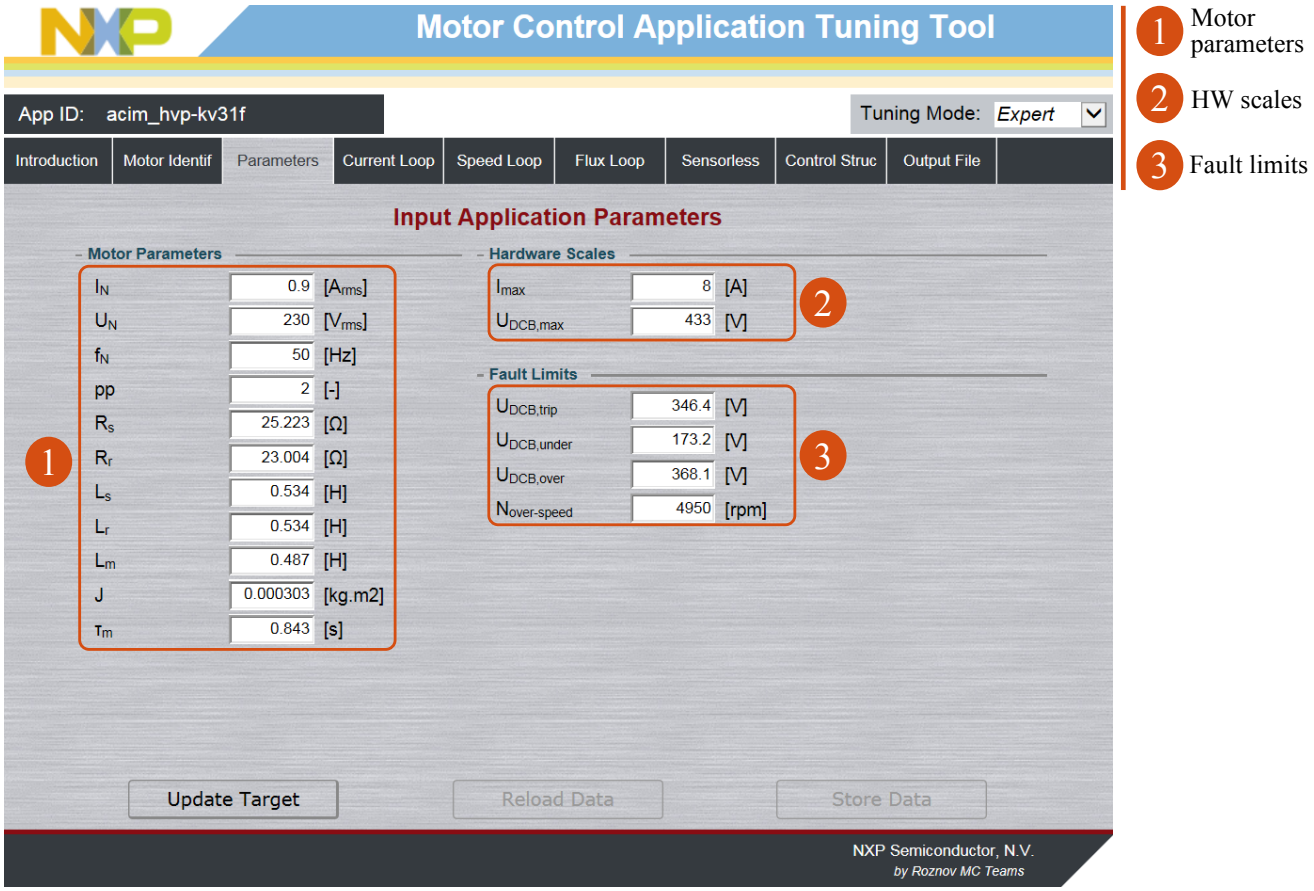

Figure 24. MCAT input application parameters tab

Table 5 shows the list of MCAT input parameters with their physical units, brief description, impacted algorithms, and accessibility status in the basic mode:

Table 5. Parameters tab inputs

Input name	Units	Description	Use in constant calculation	Basic mode accessibility
I_N	A_{rms}	Nominal stator current	Speed and flux loop	yes
U_N	V_{rms}	Nominal stator voltage	Current and flux loop	yes
f_N	Hz	Nominal frequency	Speed and flux loop	yes
pp	—	Number of motor pole pairs	Speed control, RFO, and MRAS	yes
R_S	Ω	Stator resistance	Current loop and RFO	yes
R_r	Ω	Rotor resistance	Current loop and RFO	yes
L_S	H	Stator inductance	Current loop and RFO	yes
L_r	H	Rotor inductance	Current loop and RFO	yes

Table continues on the next page...

Table 5. Parameters tab inputs (continued)

Input name	Units	Description	Use in constant calculation	Basic mode accessibility
L_m	H	Magnetizing inductance	Current loop, flux loop, and RFO	yes
J	kgm^2	Moment of inertia	Speed loop	yes
T_m	s	Mechanical time constant	Speed loop	yes
I_{max}	A	Hardware current-sensing scale	Current sensing	yes
$U_{DCB,max}$	V	Hardware DC-bus voltage sensing scale	Voltage sensing	yes
$U_{DCB,trip}$	V	Trigger value that switches an external DC-bus braking resistor on	Fault protection	no
$U_{DCB,under}$	V	Voltage value that generates the DC-bus under-voltage fault	Fault protection	no
$U_{DCB,over}$	V	Voltage value that generates the DC-bus over-voltage fault	Fault protection	no
$N_{over-speed}$	rpm	Over-speed threshold	Fault protection	no

7.3.2 Sensorless rotor flux position and speed estimation

The rotor flux position and mechanical speed feedback signals are obtained using the sensorless RFO and the Model Reference Adaptive System (MRAS) speed-estimation algorithm. For information about their principles, see *Sensorless ACIM Field-Oriented Control* (document DRM150). Tune both algorithms using the *Sensorless* sub-module MCAT tab, as shown in Figure 25. All the sensorless sub-module tab inputs are listed in Table 6.

Most of the RFO parameters are calculated automatically by MCAT, and they do not need any tuning. The only parameters left for you to tune are the proportional gain $K_{p,CMPNS}$ and the integral gain $K_{i,CMPNS}$ of the RFO compensation PI controller. These parameters are usually set manually, because the settings do not vary greatly for different motors, and you can keep them at the default settings. A similar situation applies to the MRAS speed estimator and its proportional and integral gains $K_{p,MRAS}$and $K_{i,MRAS}$ of the internal PI controller. To tune the parameters of these algorithms, run the motor in the scalar control mode, while referring to the *Speed* scope located in the *Scalar/Voltage Control* sub-block in the FreeMASTER project tree. Here you can see the estimated filtered rotor speed. The estimated and scalar rotor speeds are not going to exactly match the properly-tuned RFO and MRAS because of the speed slip.

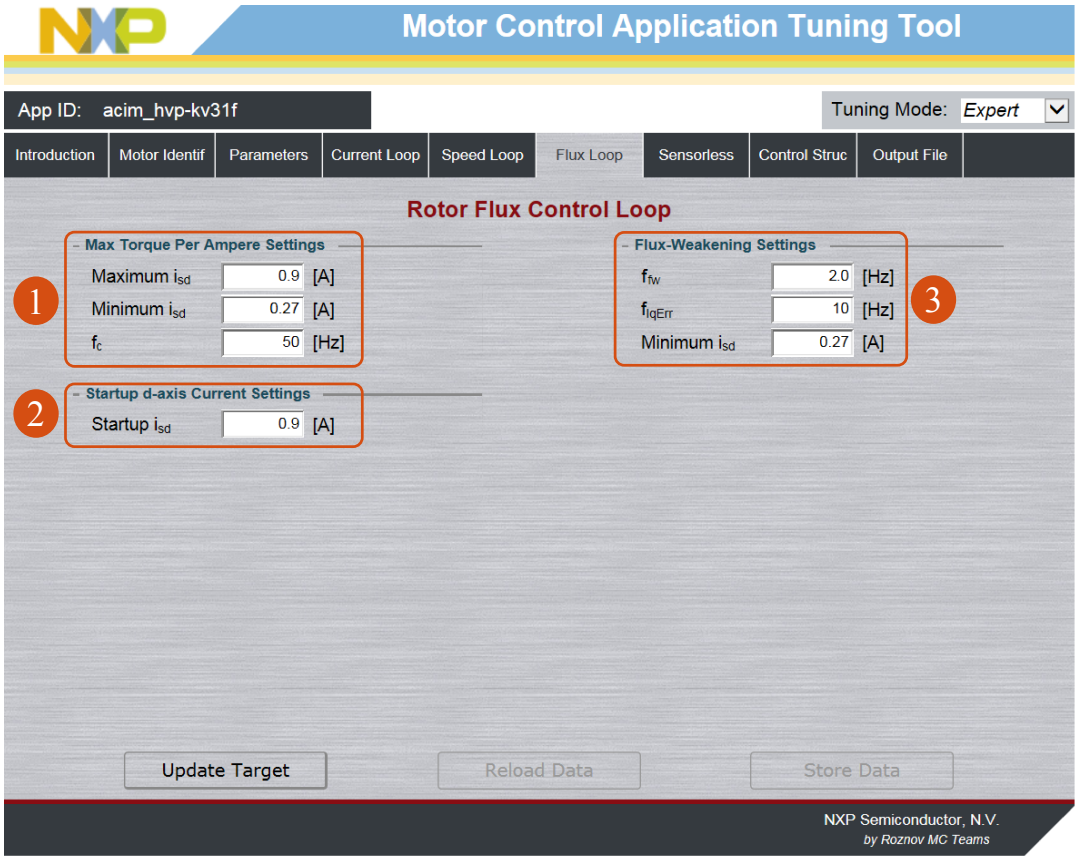

Figure 25. MCAT sensorless position and speed estimation tab

Table 6. MCAT sensorless position and speed estimation tab inputs

Parameter Name	Units	Description	Use in constant calculation	Basic mode accessibility
K_{PCMPNS}	—	Compensation PI controller proportional gain	RFO	no
K_{ICMPNS}	—	Compensation PI controller integral gain	RFO	no
$f_{PsiSInt}$	Hz	Stator flux integrator filter frequency	RFO	no
K_{PMRAS}	—	Compensation PI controller proportional gain	MRAS speed estimation	no
K_{IMRAS}	—	Compensation PI controller integral gain	MRAS speed estimation	no

A part of the RFO algorithm requires an internal calculation of the stator flux, which involves pure integration that has problems with the integrator drift. These problems are solved by approximating the pure integrator with the low-pass filter. See *Sensorless ACIM Field-Oriented Control* (document DRM150) for more details. The low-pass filter cut-off frequency is set in the $f_{PsiSInt}$ input

field and it is recommended to be set in the range from 1 Hz to 3 Hz. Higher values can lead to a high number of flux and speed estimation errors.

7.3.3 Current loop tuning

The *Current Loop* tab is designed for the current control loop tuning. The current control loop is the most inner loop in the cascade control structure of a vector-control algorithm. One of the FOC characteristics is a separate control of the rotor flux-producing (*d*-axis) and torque-producing (*q*-axis) components of the current. Therefore, the ACIM control structure has two current loops, and each of them contains a PI controller. The *Current Loop* tab is shown in Figure 26. The individual fields are described in Table 7. Set all of the inputs on the left-hand side of the tab (points one and two). The PI controller resulting gains are located on the right-hand side (point three). The sampling time field is filled in automatically when the MCU platform is successfully detected by the MCAT and cannot be changed.

Table 7. MCAT current control loop tab inputs

Parameter name	Units	Description	Use in constant calculation	Basic mode accessibility
F0	Hz	Current control loop bandwidth	Current loop	no
ζ	—	Damping ratio of the current control loop	Current loop	no
Output limit	%	Current loop output limit in percentage of the DC-bus voltage	Current loop	no

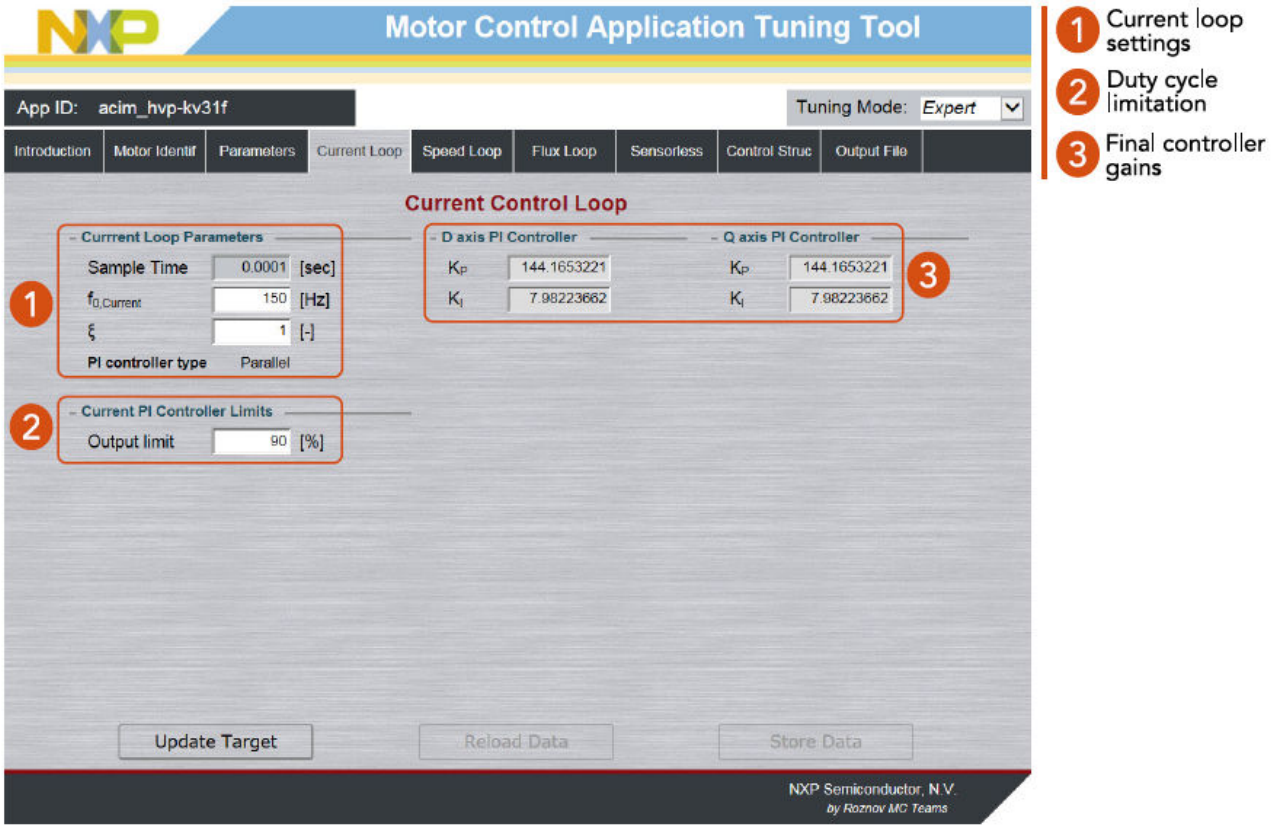

Figure 26. MCAT Current Control Loop tab

The simplified block diagrams of both the *d*-axis and *q*-axis current control loops are shown in Figure 27. The non-linear coupling parts of the stator voltage are ignored and treated as unmeasured errors entering the controlled system. The parasitic time constants (such as the inverter time constant) are ignored as well.

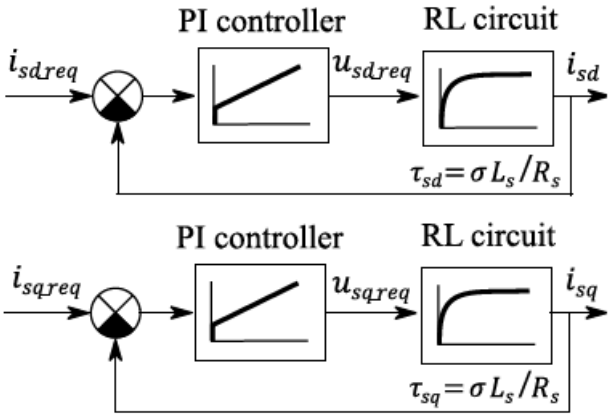

Figure 27. d-axis and q-axis current loop block diagram

Ignoring the non-linear coupling portion of the stator voltage (which is simply dealt with by the integral part of the current PI controllers), the transfer functions of the stator currents i_{sd} and i_{sq} are:

$$F_{isd}(s) = \frac{I_{sd}(s)}{U_{sd}(s)} = \frac{1/R_s}{\tau_{sd}s + 1} = \frac{1}{\sigma L_s s + R_s}$$

$$F_{isq}(s) = \frac{I_{sq}(s)}{U_{sq}(s)} = \frac{1/R_s}{\tau_{sq}s + 1} = \frac{1}{\sigma L_s s + R_s}$$

where s is the Laplace operator, τ_{Sd} and τ_{Sq} are the stator d-axis and q-axis electric time constants, and

$$\sigma = 1 - \frac{L_m^2}{L_s L_r} \quad [-]$$

is the leakage coefficient.

The transfer function of the PI controller in a parallel form is:

$$F_{PI}(s) = K_p + \frac{K_i}{s} = \frac{K_p s + K_i}{s}$$

where K_p is the proportional gain and K_i is the integral gain. The closed-loop d-axis and q-axis current transfer functions are:

$$F_{wisd}(s) = \frac{I_{sd}(s)}{I_{sd\_req}(s)} = \frac{F_{PI}(s)F_{isd}(s)}{1 + F_{PI}(s)F_{isd}(s)} = \frac{\frac{K_{pd}}{K_{id}}s + 1}{\frac{\sigma L_s}{K_{id}}s^2 + \frac{R_s + K_{pd}}{K_{id}}s + 1}$$

$$F_{wisq}(s) = \frac{I_{sq}(s)}{I_{sq\_req}(s)} = \frac{F_{PI}(s)F_{isq}(s)}{1 + F_{PI}(s)F_{isq}(s)} = \frac{\frac{K_{pq}}{K_{iq}}s + 1}{\frac{\sigma L_s}{K_{iq}}s^2 + \frac{R_s + K_{pq}}{K_{iq}}s + 1}$$

By comparing these transfer functions to the transfer function of a second-order system with the unity gain

$$F_{2nd}(s) = \frac{1}{\frac{s^2}{4\pi^2 f_0^2} + \frac{s\zeta}{\pi f_0} + 1},$$

where f_0 is the system natural frequency (or bandwidth) and ζ is the system damping ratio, you obtain:

$$K_{pd} = K_{pq} = 4\pi f_0 \zeta \sigma L_s - R_s$$

$$K_{id} = K_{iq} = 4\pi^2 f_0^2 \sigma L_s$$

The proportional and integral gains of a discrete version of the current PI controller can be obtained using the bilinear transformation method:

$$K_{pz} = K_p$$

$$K_{iz} = T_s K_i$$

NOTE

The correct value of the integral gain (according to the bilinear transformation) must be half the value stated in Eq. 30. The division by two is not shown because it is conducted internally by the PI controller algorithm in the RTCESL (see www.nxp.com/rtcesl.)

The effect of the damping ratio ζ on the step response of a second-order system $F_{2nd}(s)$ is shown in Figure 28. MCAT allows setting the damping ratio ζ in the range from 0.5 to 2.0. It is not recommended to divert from the value of 1 too much. Choose the natural frequency in the range from tens to hundreds of Hz, but at least one order higher than the speed loop bandwidth. If the bandwidth value is too high, it leads to problems with the sampling frequency, voltage limitation, and stability.

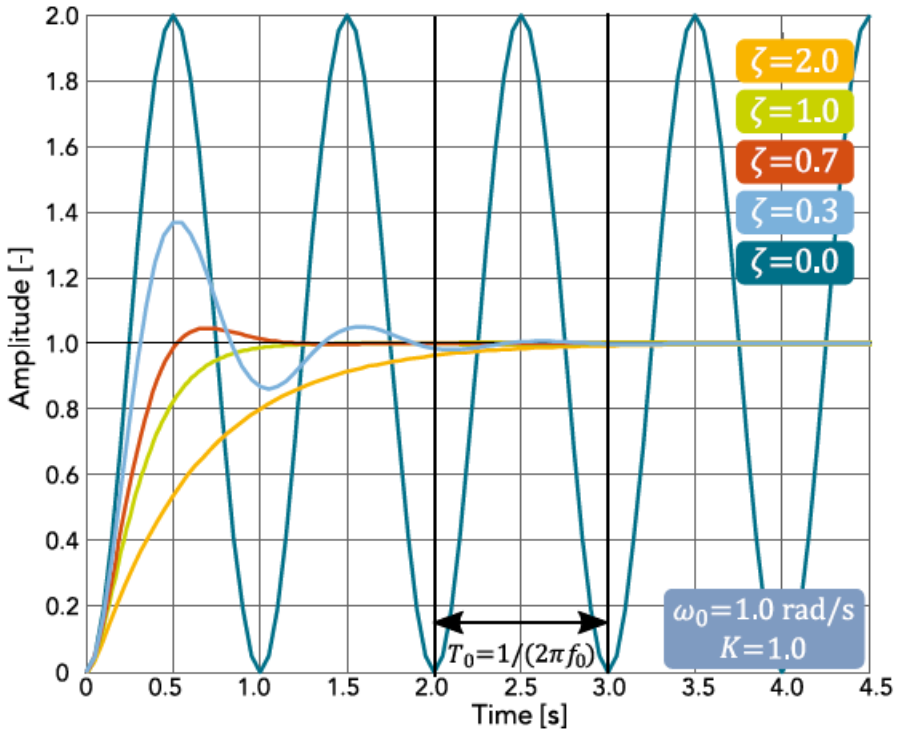

Figure 28. Second-order system steps response for various damping ratios

When comparing the transfer functions in Eq. 24, Eq. 25, and Eq. 26, the numerator (zero of the system) with the time constant K_p / K_i of the current closed-loop transfer function is ignored, because it has minor impact on the resulting system stability.

To check the current response, use the FreeMASTER recorder called *Current Control*, which is triggered during motor startup. The examples of the *d*-axis current response for different setups of the current loop bandwidth are shown in Figure 29. The ideal current response must not be too slow (as in case C), but it must neither contain a high overshoot. A very high current loop bandwidth can lead to instability. If you are not satisfied with the automatically-calculated current loop PI controller parameters, tune them manually. To do so, perform these steps:

- Go to the *Current Loop* tab and select the *Expert* tuning mode.
- Set the desired current loop bandwidth f_0 and click *Update Target*. It is recommended to start with a lower value and then keep increasing it until the desired response is achieved.
- Select the *Current Loop* recorder. The message at the bottom of the recorder must read *"Running, waiting for trigger…"*.
- Go to the *Control struc* tab, select the *Current FOC* control mode, and set small required values of both the *d*-axis and *q*-axis currents.
- Run the application and wait for the data to load.
- Check the downloaded response in the recorder and repeat the procedure from step two (if necessary).

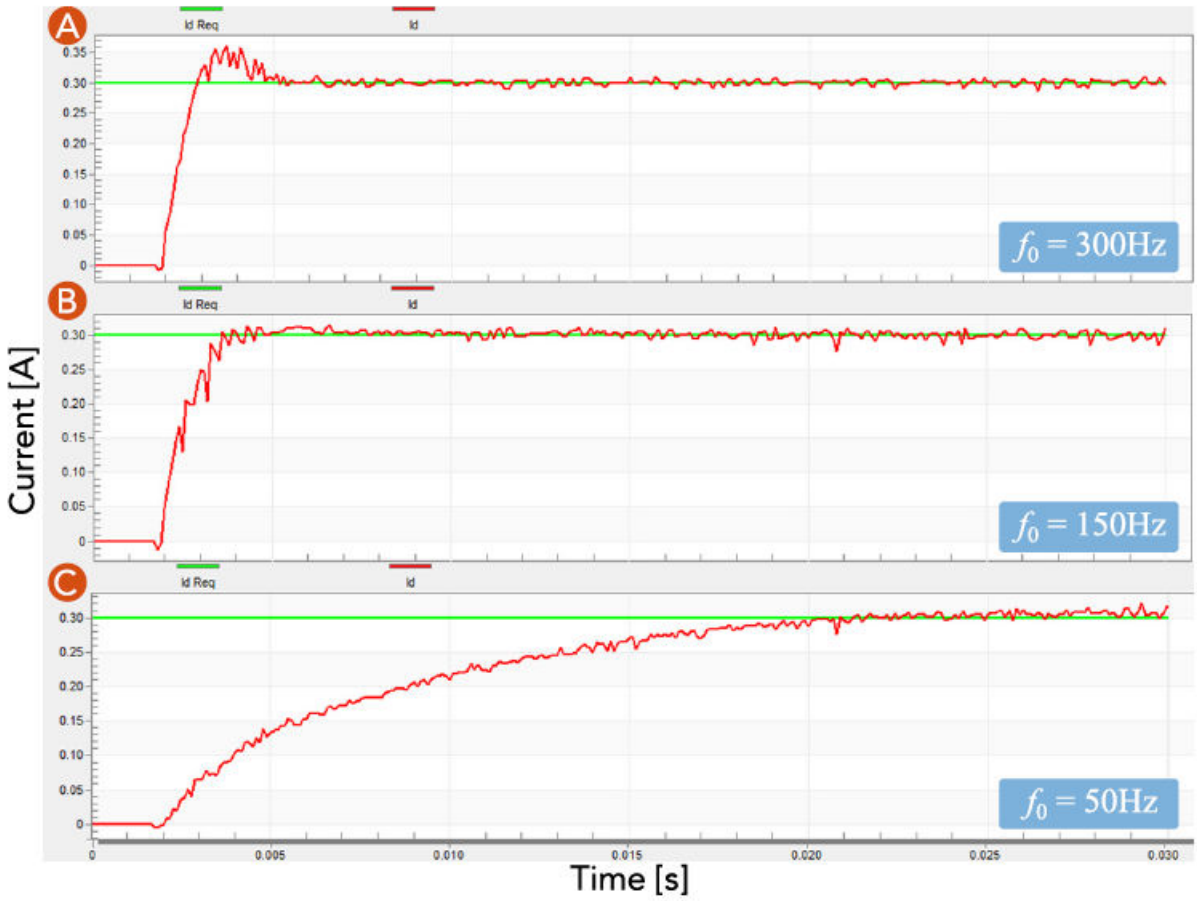

Figure 29. Response of d-axis current to different settings of current loop bandwidth

7.3.4 Speed loop tuning

The *Speed Loop* tab is designed to tune the speed-control loop. The speed-control loop is an outer loop in the cascade-control structure of a vector-controlled ACIM. The speed loop consists of the PI controller, the estimated speed filter, and the S-ramp function, which limits the maximum, minimum, acceleration, and jerk of the required speed. The screenshot of the *Speed Loop* tuning page is shown in Figure 30 and the individual fields are described in Table 8.

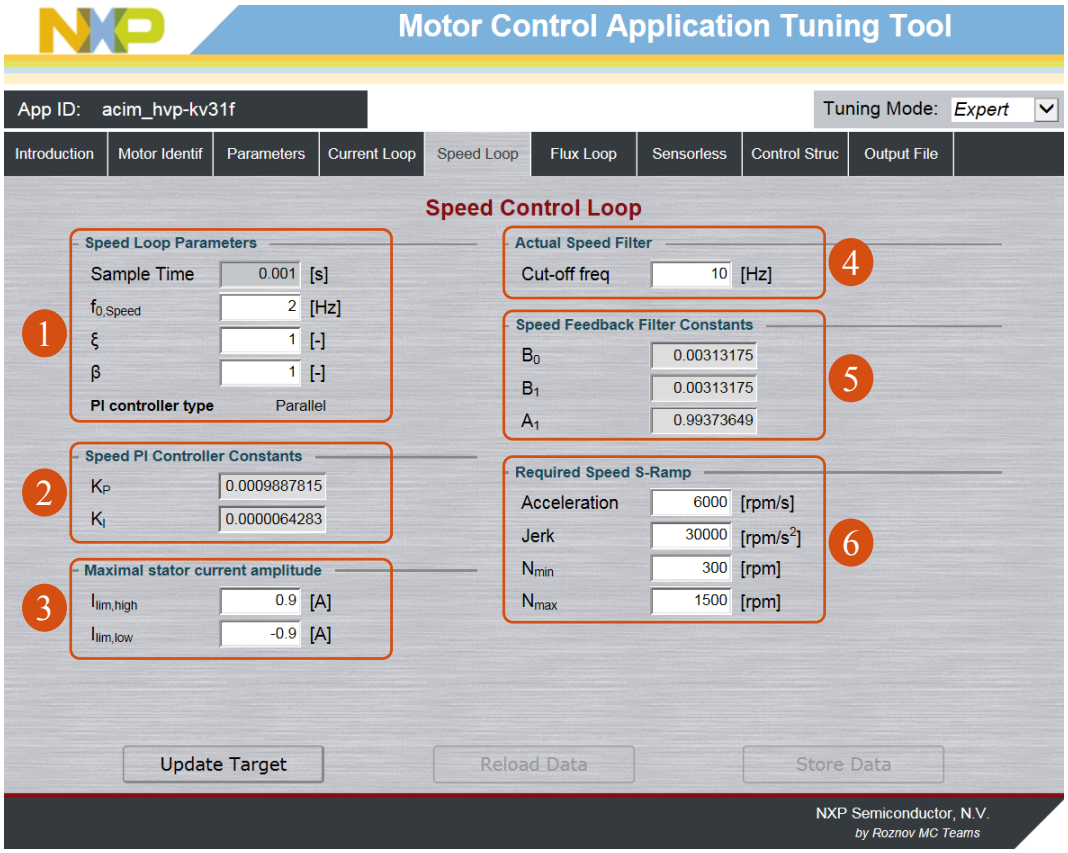

Figure 30. MCAT Speed Control Loop tab

Table 8. MCAT Speed Control Loop tab parameters

Parameter name	Units	Description	Use in constant calculation	Basic mode accessibility
Sample time	s	Speed loop sampling time period	Speed loop	no
F_0	Hz	Speed control loop bandwidth	Speed loop	yes
ζ	—	Damping ratio of speed control loop	Speed loop	no
β	—	Overshoot damping coefficient	Speed loop	no
$I_{lim,high}$	A	Speed loop output upper limit	Speed loop	no
$I_{lim,low}$	A	Speed loop output lower limit	Speed loop	no
Cut-off freq	Hz	Speed filter cut-off frequency	Speed loop	no

Table continues on the next page...

Table 8. MCAT Speed Control Loop tab parameters (continued)

Parameter name	Units	Description	Use in constant calculation	Basic mode accessibility
Acceleration	rpm/s	Acceleration of the required speed	Speed loop	yes
Jerk	rpm/s^2	Jerk of the required speed	Speed loop	no
N_{max}	—	Maximal required speed	Speed loop	no
N_{min}	%	Minimal required speed	Speed loop	no

A simplified block diagram of the speed-control loop is shown in Figure 31. The simplification lies in ignoring the current loop dynamics, because it is presumed to be much faster than the dynamics of a mechanical system.

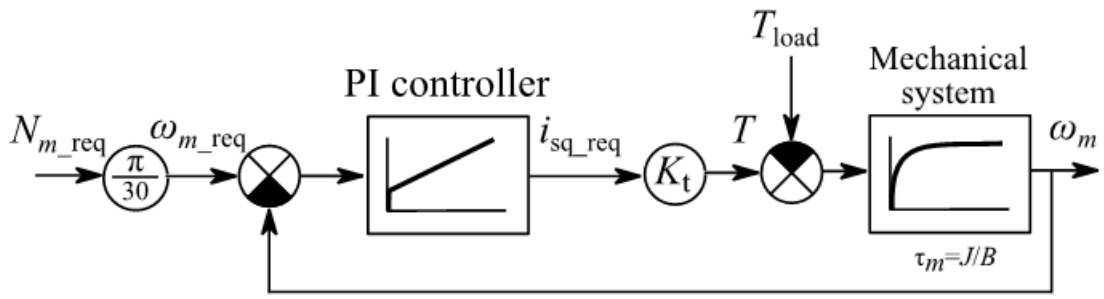

Figure 31. Speed loop block diagram

Calculate the torque constant K_t of an ACIM as follows:

$$K_t = \frac{3}{4} P_p \frac{L_m^2}{L_r}$$

Note that the previous equation ignores the *d*-axis current. This value may change during the motor operation, which affects the behavior of the transient speed response. The torque constant is therefore adapted in run-time according to the required *d*-axis current required value and the speed controller is calculated for case $i_{Sd} = 1A$.

When ignoring the load torque, the transfer function of a complete driven mechanical system is:

$$F_\omega(s) = \frac{\Omega_m(s)}{I_{sq\_req}(s)} = \frac{1/B}{\tau_m s + 1} = \frac{1}{Js + B}$$

where τ_m is the mechanical time constant, J is the moment of inertia, and B is the mechanical viscous friction. Considering the PI controller to be in a parallel form according to Eq. 23, the closed speed control loop is:

$$F_{w\omega}(s) = \frac{\Omega_m(s)}{\Omega_{m\_req}(s)} = \frac{F_{PI}(s)F_{\omega}(s)}{1 + F_{PI}(s)F_{\omega}(s)} = \frac{\frac{K_{p\omega}}{K_{i\omega}}s + 1}{\frac{J}{K_t K_{i\omega}}s^2 + \frac{B + K_t K_{p\omega}}{K_t K_{i\omega}}s + 1}$$

where $K_{p\omega}$ is the speed PI controller proportional gain and $K_{i\omega}$ is the integral gain. By comparing this transfer function with the transfer function of a second-order system in Eq. 26, you obtain:

$$K_{p\omega} = \frac{4\zeta\pi f_0 J - B}{K_t}$$

$$K_{i\omega} = \frac{4\pi^2 f_0^2 J}{K_t}$$

The selection of damping ratio ζ and speed loop bandwidth f_0 follows similar rules as in the current loop. Choose a bandwidth at least one order smaller (in case of the current loop). Calculate the proportional and integral gains of a discrete version of the speed *βIP* controller using the bilinear transformation method, as shown in Eq. 29 and Eq. 30. The parameter of the controller β can be used to suppress the required speed response overshoot, while keeping a quick response to the changing load. The β parameter can be set in a range from zero (maximal overshoot suppression and slower response) to one (no suppression, the response is equal to a classic PI controller).

To check the speed response, open the FreeMASTER scope named *Speed*, located under the *Speed Control* sub-block. If you are not satisfied with the speed response resulting from the automatically calculated parameters, tune the controller manually. To do so, perform these steps:

- Go to the *Speed Loop* tab and select the *Expert* tuning mode.
- Set the desired speed loop bandwidth f_0 and click *Update Target*. It is recommended to start with a lower value (in the range of Hz, depending on the mechanical time constant) and then increasing it until the desired response is achieved.
- Select the *Speed* scope in the *Speed Control* sub-section.
- Set the required speed and observe the response.
- Check the downloaded response in the recorder and repeat from step two (if necessary).

7.3.5 Flux loop tuning

The *Flux Loop* tab is designed to tune the Max Torque Per Ampere (MTPA) and Flux Weakening (FW) algorithms, which forms the second outer loop in the cascade-control structure of the ACIM vector control. Both algorithms are more closely described in *Sensorless ACIM Field-Oriented Control* (document DRM150). The screenshot of the *Flux Loop* tuning page is shown in Figure 32 and the individual fields are described in Table 9:

Table 9. MCAT flux control loop tab parameters

Parameter name	Units	Description	Use in constant calculation	Basic mode accessibility
Maximum i_{sd}	A	Maximal d-axis current	Speed loop	no
Minimum i_{sd}	A	Minimal d-axis current	Speed loop	no
f_c	Hz	Required d-axis current filter	Speed loop	no
Startup i_{sd}	A	Startup d-axis current	Speed loop	no
f_{fw}	Hz	FW controller bandwidth	Speed loop	no
f_{IqErr}	Hz	Speed loop output lower limit	Speed loop	no

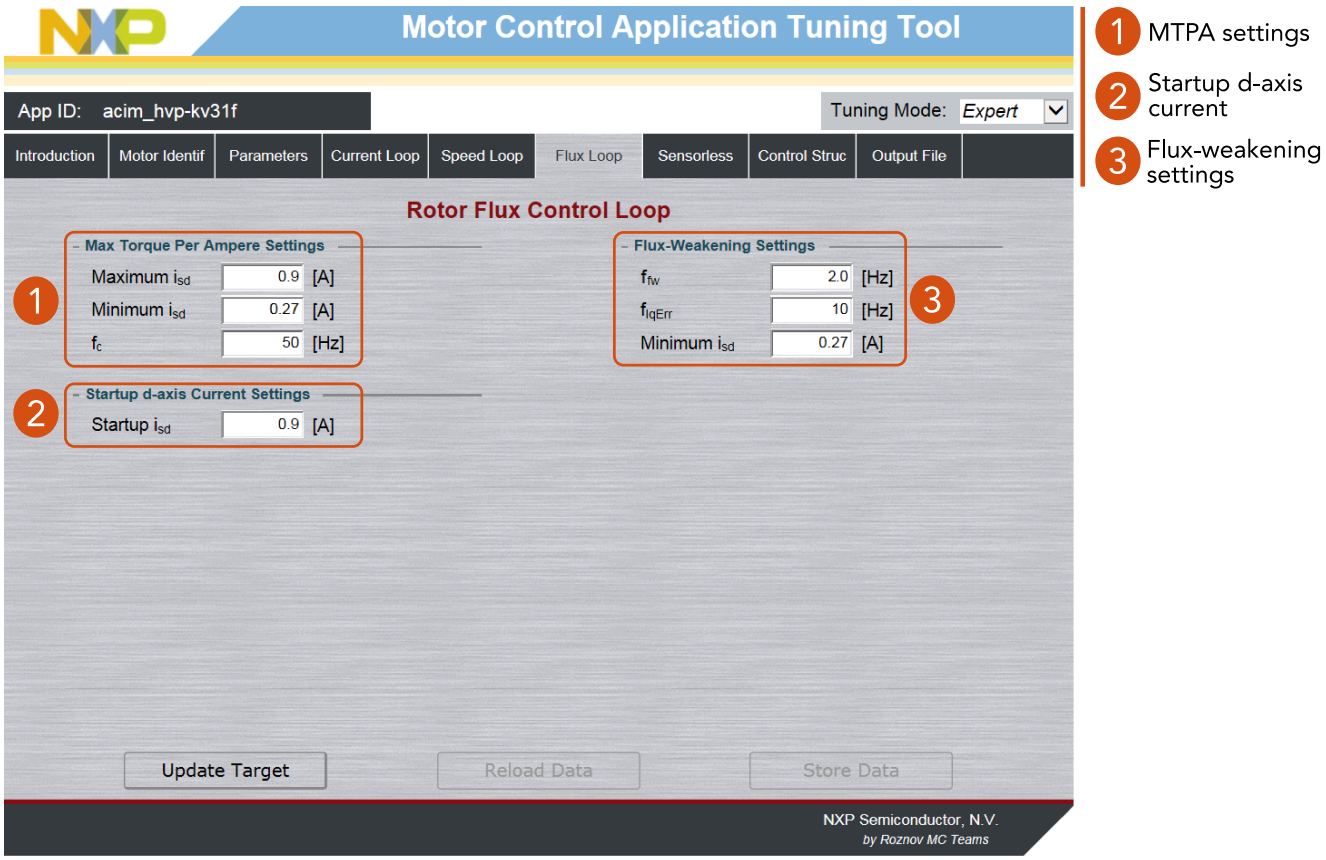

Figure 32. MCAT flux loop tuning tab

The only parameters that are required to be set for the MTPA are the *d*-axis current limits $i_{Sd\_req,max}$ and $i_{Sd\_req,min}$ and the filter bandwidth. The upper limit should be set to a value that corresponds to the nominal amplitude of the rotor flux, which means:

$$i_{sd\_req,max} = 0.707 I_{phN} \quad [\mathrm{A}]$$

Setting the lower *d*-axis current limit low allows for better power optimization (depends on the load). However, setting it too low might affect the RFO performance and lead to a control failure. It is recommended to set $i_{Sd\_req,min}$ to at least 25 % of the upper limit $i_{Sd\_req,max}$ (or more).

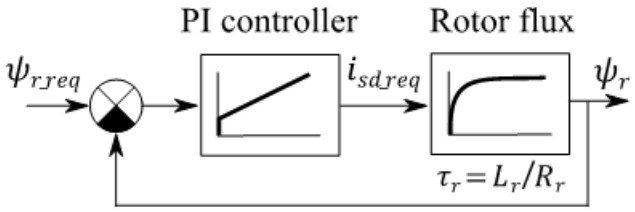

Figure 33. Rotor flux control loop block diagram

The rotor flux control loop to tune the flux-weakening PI controller is shown in Figure 33. The transfer function of the controlled rotor flux system is:

$$F_{\psi}(s) = \frac{\Psi_r(s)}{I_{sd\_req}(s)} = \frac{1/L_m}{\tau_r s + 1}$$

Considering the PI controller to be in a parallel form according to Eq. 23, the open control loop is:

$$F_{0\psi}(s) = \frac{K_{p\Psi} s + K_{i\Psi}}{s} \frac{1/L_m}{\tau_r s + 1}$$

By placing the controller zero to the systems pole, which means:

$$K_{p\psi} = K \frac{\tau_r}{L_m}$$

$$K_{i\psi} = K \frac{1}{L_m}$$

The open loop transfer reduces to $F_{0\Psi} = K / s$, where K is the general constant. Setting the $K = 2\pi f_0$ leads to the closed loop transfer function:

$$F_{w\psi}(s) = \frac{\Psi_r(s)}{\Psi_{r\_req}(s)} = \frac{1}{\frac{1}{2\pi f_0} s + 1}$$

where f_0 is the flux-weakening controller bandwidth. The final discrete controller gains are therefore calculated as follows:

$$K_{p\psi z} = 2\pi f_0 \frac{\tau_r}{L_m}$$

$$K_{i\psi z} = T_s 2\pi f_0 \frac{1}{L_m}$$

7.3.6 MCAT output file generation

When you successfully tune the application and want to store all the calculated parameters to the embedded application, navigate to the *Output File* tab. View the list of all definitions generated by MCAT there. Clicking the *Generate Configuration File* button overwrites the older version of the *m1_acim_appconfig.h* file, which contains all FOC algorithm definitions. To generate the file into a correct location, connect the target MCU via FreeMASTER. Otherwise, when in the offline mode, the file is generated next to the *.pmp* file.

Chapter 8
Conclusion

This user's guide describes the implementation of the sensorless ACIM application. The hardware-dependent part of the software, which includes peripheral initialization and application timing, is described in MCU peripheral settings. The initialization and API of the Motor-Control Peripheral Drivers, which allows for a simple and unified access to the PWM and ADC on all supported devices, is in Motor-Control Peripheral Drivers. The last part of the document describes the sensorless ACIM application tuning and control using the FreeMASTER-based MCAT tool. All the steps necessary for running the ACIM-like parameter identification, current loop, speed loop, and flux loop tuning are described as well.

Chapter 9
Acronyms and abbreviations

Table 10. Acronyms and abbreviations

Term	Meaning
AC	Alternating Current
ACIM	AC Induction Machine
ADC	Analog-to-Digital Converter
CPU	Central Processing Unit
CMP	Comparator
DC	Direct Current
DRM	Design Reference Manual
FOC	Field-Oriented Control
FW	Flux-Weakening
FTM	FlexTimer Module
RTCESL	Real-Time Embedded Software Library
GPIO	General-Purpose Input/Output
HVP	High-Voltage development Platform
I/O	Input/Output interface
MCAT	Motor Control Application Tuning tool
MCDRV	Motor Control Peripheral Drivers
MCU	Microcontroller Unit
MRAS	Model Reference Adaptive System
MTPA	Maximum Torque Per Ampere
PDB	Programmable Delay Block
PI	Proportional Integral controller
PWM	Pulse-Width Modulation
RFO	Rotor Flux Observer
UART	Universal Asynchronous Receiver/Transmitter
VSI	Voltage Source Inverter

Chapter 10
References

These references are available on www.nxp.com:

- *Sensorless ACIM Field Oriented Control* (document DRM150)
- *KV31F Sub-Family Reference Manual* (document KV31P100M120SF7RM)
- *KV4x Reference Manual* (documentKV4XP100M168RM)
- *KV5x Sub-Family Reference Manual* (document KV5XP144M240RM)
- *NXP High-Voltage Motor Control Platform User's Guide* (documentHVPMC3PHUG)
- *HVP-KV31F120M User's Guide* (document HVPKV31F120MUG)
- *HVP-KV46F150M User's Guide* (document HVPKV46F150MUG)
- *HVP-KV58F220M User's Guide* (document HVPKV58F220MUG)
- *Using FlexTimer in ACIM/PMSM Motor Control Applications* (document AN3729)
- *Tips and Tricks Using PDB in Motor Control Applications on Kinetis* (document AN4822)
- *Motor Control Application Tuning (MCAT) Tool for Three-Phase PMSM* (document AN4642)
- *Filter-Based Algorithm for Metering Applications* (document AN4265)

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: 05/2020

Document identifier: ACIMKV46DEMOUG

arm