NXP Semiconductors
Application Note

Document Number: AN12374
Rev. 0, 03/2019

PMSM Field-Oriented Control on FRDM-KV31F with Hall and Encoder Sensors

1. Introduction

This document describes the implementation of the sensor (hall and encoder sensors) and sensorless speed motor-control software for 3-phase Permanent Magnet Synchronous Motors (PMSM), including the motor parameters identification algorithm, on the FRDM-KV31F development platform.

The FRDM-MC-LVPMSM NXP Freedom board is used as a hardware platform for the PMSM control reference solution. The hardware-dependent part of the motor-control software is addressed as well, including a detailed peripheral setup and driver description. The motor parameters identification theory and the algorithms are also described in this document.

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 runtime debugging tool. These tools represent a simple and user-friendly way of the motor parameters identification, algorithm tuning, software control, debugging, and diagnostics.

Contents

2. Hardware setup

The PMSM Field-Oriented Control (FOC) application runs on the FRDM-MC-LVPMSM development platform with the FRDM-KV31F development tool, in combination with the Teknic M-2310P-LN or Linix 45ZWN24-40 permanent magnet synchronous motors.

2.1. FRDM-KV31F MCU board

The FRDM-KV31F board is a low-cost development tool for Kinetis KV3x family of MCUs built around the Arm® Cortex®-M4 core. The FRDM-KV31F board is form-factor compatible with the Arduino™ R3 pin layout, providing a broad range of expansion board options, including FRDM-MC-LVPMSM and FRDM-MC-LVBLDC for permanent-magnet and brushless-DC motor control.

The FRDM-KV31F board features OpenSDA (NXP open-source hardware embedded serial and debug adapter) running an open-source bootloader. This circuit offers several options for serial communication, flash programming, and run-control debugging.

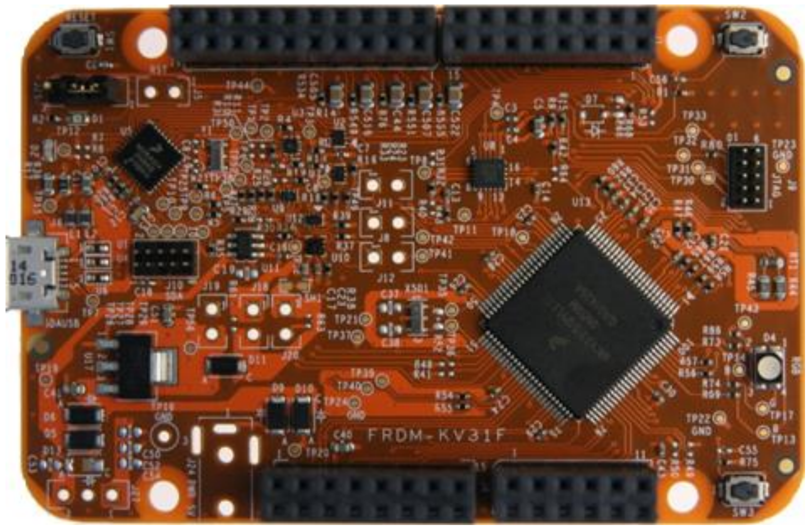


Figure 1. FRDM-KV31F development board

2.2. FRDM-MC-LVPMSM power stage

The FRDM-MC-LVPMSM low-voltage, 3-phase Permanent Magnet Synchronous Motor (PMSM) Freedom development platform board has the power supply input voltage of 24-48 VDC with a reverse polarity protection circuitry. The auxiliary power supply of 5.5 VDC is created to supply the FRDM MCU boards. The output current is up to 5 A RMS. The inverter itself is realized by a 3-phase bridge inverter (six MOSFETs) and a 3-phase MOSFET gate driver. The analog quantities (such as the 3-phase motor currents, DC-bus voltage, and DC-bus current) are sensed on this board. There is also an interface for speed and position sensors (encoder, hall). The block diagram of a complete NXP Freedom motor-control development kit is shown in Figure 2.

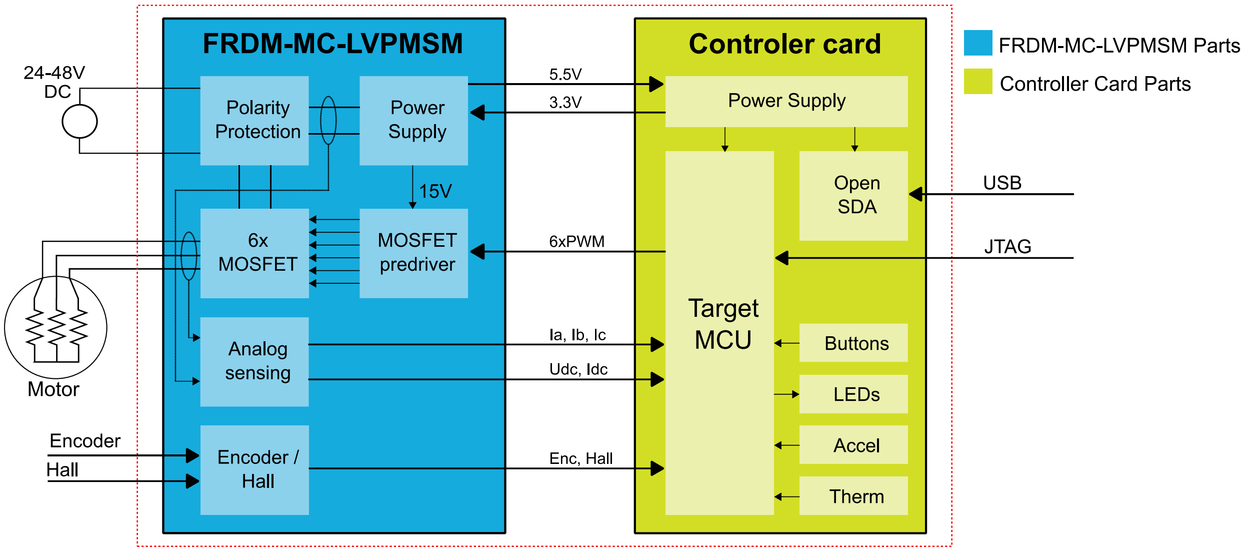


Figure 2. Motor-control development platform block diagram

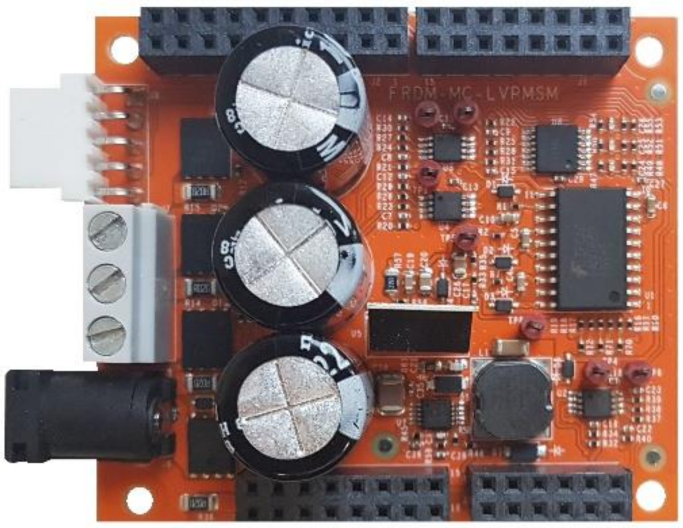


Figure 3. FRDM-MC-LVPMSM

The FRDM-MC-LVPMSM board does not require a complicated setup and there is only one way to connect this shield board to the Freedom MCU board. For more information about the NXP Freedom development platform, see www.nxp.com/freedom.

2.3. Linix 45ZWN24-40 motor

Linix 45ZWN24-40 is the default low-voltage 3-phase motor (described in Table 1) used in PMSM applications.

Table 1. Linix 45ZWN24-40 motor parameters

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

Figure 4. Linix motor

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

2.4. Teknic M-2310P motor

The Teknic M-2310P-LN-04K motor is another low-voltage 3-phase permanent-magnet motor used in PMSM applications. When you are using this motor, rename the *M1_params_pmsm_frdm-kv31f-hall_teknic.txt* file located in the *freemaster\mcat\param_files\* folder to *M1_params_pmsm_frdm-kv31f-hall.txt*. Rename also the *m1_pmsm_appconfig_teknic.h* header file located in *\src\projects\frdmkv31f\ to m1_pmsm_appconfig.h*. Then rebuild and download the project to the target device, as shown in Section 6, "Building and debugging applications". The motor has two feedback sensors (hall and encoder sensors). For the feedback sensors' wiring, see the datasheet on the manufacturer web pages. The motor parameters are summarized in Table 2.

Table 2. Teknic M-2310P motor parameters

Characteristic	Symbol	Value	Units
Rated voltage	Vt	40	V
Rated speed @ Vt	—	6000	RPM
Rated torque	T	0.274	Nm
Rated power	P	170	W
Continuous current	Ics	7.1	A
Number of pole pairs	pp	4	—

Figure 5. Teknic M-2310P permanent-magnet synchronous motor

For the sensorless control mode, only the power input wires are needed. When using hall or encoder sensors, connect also the sensor wires to the NXP Freedom power stage.

2.5. NXP Freedom system assembling

1. Connect the FRDM-MC-LVPMSM shield on top of the FRDM-KV31F board
   (there is only one possible option).
1. Connect the Teknic or Linix motor 3-phase wires into the screw terminals on the board.
2. If you want to use a feedback sensor, connect the encoder (or hall) sensor to the connector on the board.
3. Plug the USB cable from the USB host to the OpenSDA micro-USB connector.
4. Plug the 24 VDC power supply to the DC power connector.

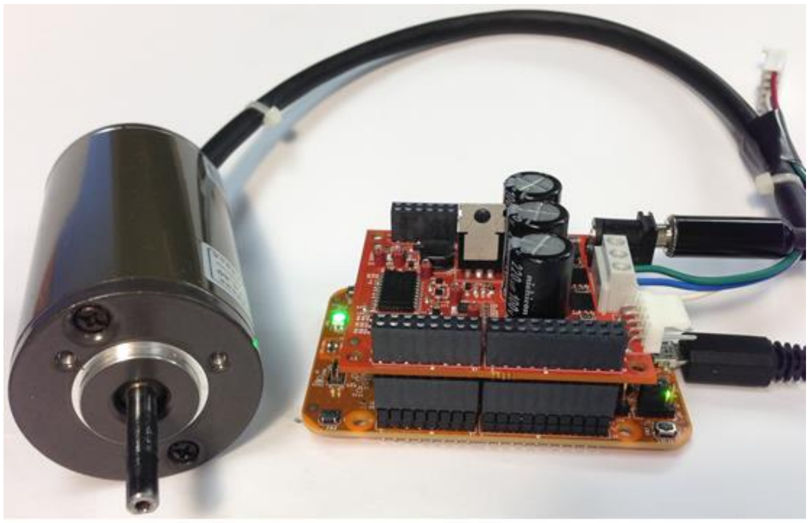


Figure 6. Assembled NXP Freedom system

3. KV31F MCU features and peripheral settings

The peripherals used for motor control are dependent on the Kinetis V series MCU used. The following sections describe the peripheral settings and application timing for the Kinetis KV31F MCU.

3.1. KV31F MCU

The KV31F MCU is a highly scalable member of the Kinetis V series and provides a high-performance and cost-competitive motor-control solution. Built upon the Arm Cortex-M4 core running at 120 MHz with up to 512 KB of flash and up to 96 KB of RAM and combined with the floating-point unit, it delivers a platform enabling customers to build a scalable solution portfolio. The additional features include dual 16-bit ADCs sampling at up to 1.2 MS/s in a 12-bit mode, 20 channels of flexible motor-control timers (PWMs) across four independent time bases, and a large RAM block, enabling local execution of fast control loops at a full clock speed. For more information, see the *KV31F Sub-Family Reference Manual* (document KV31P100M120SF7RM).

3.1.1. Hardware timing and synchronization

Correct and precise timing is crucial in motor-control applications. The motor-control-dedicated peripherals handle the timing and synchronization on the hardware layer. In addition, you can set the PWM frequency as a multiple of the ADC interrupt (FOC calculation) frequency; in this case, $FOC_{freq} = PWM_{freq}/2$. The timing diagram is shown in Figure 7.

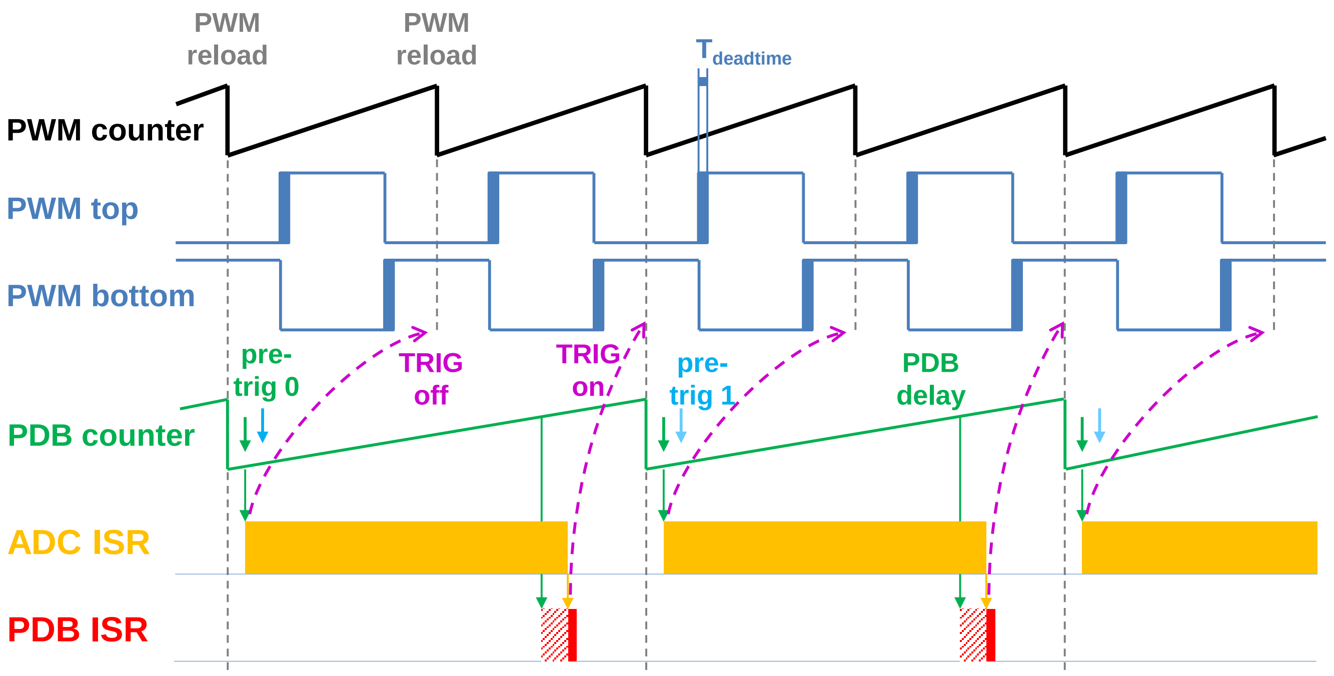


Figure 7. Hardware timing and synchronization on KV31F

- The top signal (**PWM counter**) shows the FTM counter reloads. The dead time is emphasized on the **PWM top** and **PWM bottom** signals. The FTM_TRIG is generated at the **PWM reload**, which triggers the PDB (resets the **PDB counter**).
- The PDB generates a first pre-trigger for the first ADC (phase current) sample with a delay of approximately $\mathbf{T_{deadtime}}$ / 2. This delay ensures correct current sampling at duty cycles close to 100 %.
- When the conversion of the first ADC sample (phase current) is completed, the **ADC ISR** is entered. At first, the next FTM_TRIG is disabled (**TRIG off**). This ensures that the **PDB**

counter does not reset at the next **PWM reload**. The FOC is then calculated.

- In the middle of the next PWM period (**PDB delay**), the **PDB ISR** is called. This interrupt enables the FTM_TRIG (**TRIG on**) at the next **PWM reload**. The **PDB ISR** has lower priority than the **ADC ISR**. The **PDB delay** length determines the ratio between the PWM and FOC frequencies.
- The PDB uses a back-to-back mode to automatically generate the **pre-trig 1** (to measure the DC-bus voltage) immediately after the first conversion is completed.

3.1.2. Peripheral settings

This section describes only the peripherals used for motor control. KV31F uses a 6-channel FlexTimer (FTM) to generate a 6-channel PWM and two 16-bit SAR ADCs to measure the phase currents and the DC-bus voltage. The FTM and ADC are synchronized via the Programmable Delay Block (PDB). One channel from another independent FTM is used for slow-loop interrupt generation.

Table 3. FRDM-KV31F peripheral settings

Peripheral	Feature	Freedome
FTM0	PWM polarity	High sides active high Low side active high
	Fault source	FAULT1, CMP1 out
	Fault polarity	Active high
	Dead time	0.5 µs
PDB	Pre-trigger 0 delay	0.25 µs

3.1.2.1. PWM generation—FTM0

- The FTM is clocked from the 60-MHz bus clock.
- Only six channels are used, the other two are masked in the OUTMASK register.
- Channels 0+1, 2+3, and 4+5 are combined in pairs and running in a complementary mode.
- The fault mode is enabled for each combined pair with automatic fault clearing (PWM outputs are re-enabled at the first PWM reload after the fault input returns to zero).
- The PWM period (frequency) is determined as the time for the FTM to count from CNTIN to MOD. By default, CNTIN = -MODULO / 2 = -3000 and MOD = MODULO / 2 - 1 = 2999. The FTM is clocked from the 60 MHz system clock, so it takes 0.0001 s (10 kHz).
- Dead time insertion is enabled for each combined pair. The dead time length is calculated as the system clock 60 MHz × $T_{deadtime}$.
- The FTM generates a trigger for the PDB on counter initialization.
- The FTM fault input is enabled, but its polarity and source vary among platforms.

3.1.2.2. Analog sensing—ADC0 and ADC1

- The ADCs operate as 12-bit, single-ended converters.
- The clock source for both ADCs is the 48-MHz IRC48 clock divided by 2 = 24 MHz.
- For ADC calibration purposes, the ADC clock is set to 6 MHz. The continuous conversion and

averaging with 32 samples are enabled in the SC3 register. After the calibration is done, the SC register is filled with its default values and the clock is set back to 24 MHz.
- Both ADCs are triggered by the PDB pre-triggers.
- There is an interrupt that serves the FOC fast-loop algorithm and is generated after the first conversion is completed.

3.1.2.3. PWM and ADC synchronization—PDB0

- Like the FTM, the PDB is clocked from the 60-MHz bus clock.
- The PDB is triggered by the FTM0_TRIG.
- The pre-trigger 0 at each channel is generated $0.5 \times T_{deadtime}$ after the FTM0_TRIG.
- The pre-trigger 1 at each channel is generated immediately after the first conversion is completed using the back-to-back mode.
- The PDB sequence error interrupt is enabled. This interrupt is generated when a certain result register is not read and the same pre-trigger occurs at the ADC.
- The PDB delay interrupt is enabled. This interrupt is generated when the PDB_IDLY is reached. This interrupt enables the FTM_TRIG (Figure 4).
- The PDB sequence error and PDB delay interrupts both share a common interrupt vector. Which event generated the interrupt is determined at the beginning of the interrupt according to the ERR flag.

3.1.2.4. Over-current detection—CMP1

- The plus input for the CMP is taken from the analog pin.
- The minus input for the CMP is taken from the 6-bit DAC0 reference. The DAC reference is set to 3.197 V (62 / 64 × VDD), which corresponds to 7.73 A (in the 8.25 A scale).
- The CMP filter is enabled and four consecutive samples must match.

3.1.2.5. Slow-loop interrupt generation—FTM1

- The slow loop is usually ten times slower than the fast loop. Therefore, the FTM1 is clocked from the system clock / 16 to keep its modulo value reasonably low.
- The FTM counts from CNTIN = 0 to MOD = SPEED_MODULO.
- The interrupt that serves the slow loop is enabled and generated at the reload.

3.1.2.6. Encoder signal processing—FTM2

- The FTM2 in quadrature mode is used for feedback encoder sensor signal processing.
- The counter of the FTM2 runs in the BDM mode.
- The FTM2 modulo counts from CNTIN = 0 to MOD = (number of encoder pulses × 4 – 1).

3.1.2.7. Hall sensors signals processing—FTM2

- The FTM2 is clocked from the 60-MHz bus clock / 128 = 46.875 MHz.
- The FTM2 counter runs in the BDM mode.
- Via the SIM module, the input is set to the XOR of the FTM2_CH0, FTM2_CH1, and FTM1_CH1 pins applied to FTM2_CH1.
- The FTM counter is reset after the FTM2_CH1 input event is detected.
- The FTM counter is captured on the rising and falling edges.
- The FTM2_CH1 interrupt is enabled.

3.1.3. CPU load and memory usage

The following information apply to the demonstration application built using the IAR Embedded Workbench® IDE. Table 4 shows the memory usage and CPU load. The memory usage is calculated from the linker **.map* file, including the 2-KB FreeMASTER recorder buffer (allocated in RAM). The CPU load is measured using the SysTick timer. The CPU load depends on the fast-loop (FOC calculation) and slow-loop (speed-loop) frequencies. In this case, it applies to the fast-loop frequency of 10 kHz and the slow-loop frequency of 1 kHz. The total CPU load is calculated using these equations:

$$CPU_{fast} = cycles_{fast} \cdot \frac{f_{fast}}{f_{CPU}} \cdot 100 \quad [\%] \qquad \text{Eq. 1.}$$

$$CPU_{slow} = cycles_{slow} \cdot \frac{f_{slow}}{f_{CPU}} \cdot 100 \quad [\%] \qquad \text{Eq. 2.}$$

$$CPU_{total} = CPU_{fast} + CPU_{slow} \ [\%] \qquad \text{Eq. 3.}$$

Where:

CPU_{fast}	—the CPU load taken by the fast loop.
$cycles_{fast}$	—the number of cycles consumed by the fast loop.
f_{fast}	—the frequency of the fast-loop calculation (10 kHz).
f_{CPU}	—CPU frequency.
CPU_{slow}	—the CPU load taken by the slow loop.
$cycles_{slow}$	—the number of cycles consumed by the slow loop.
f_{slow}	—the frequency of the slow-loop calculation (1 kHz).
CPU_{total}	—the total CPU load consumed by the motor control.

Table 4. KV31 CPU load and memory usage

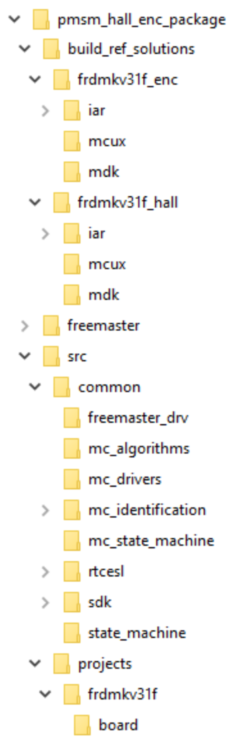
—	MKV31F
CPU load [%]	29,4
Read-only code memory [B]	34386
Read-only data memory [B]	10218
Read-write data memory [B]	11558

4. Project file structure

All the necessary files are included in one package, which simplifies the distribution and decreases the size of the final package. The directory structure of this package is simple, easy to use, and organized in a logical manner. The folder structure used in the IDE is different from the structure of the PMSM package installation, but it uses the same files. The different organization is chosen due to a better manipulation with folders and files in workplaces and due to the possibility to add or remove files and directories.

4.1. PMSM project structure

The folder tree of the PMSM package installation is shown in Figure 8.



```
pmsm_hall_enc_package
  build_ref_solutions
    frdmkv31f_enc
      iar
      mcux
      mdk
    frdmkv31f_hall
      iar
      mcux
      mdk
  freemaster
  src
    common
      freemaster_drv
      mc_algorithms
      mc_drivers
      mc_identification
      mc_state_machine
      rtcesl
      sdk
      state_machine
    projects
      frdmkv31f
        board
```

Figure 8. Package structure

The PMSM package is composed of these folders:

- *build_ref_solutions*.
- *freemaster*.
- *src*.

The main project folder */build_ref_solutions/frdmkv31_enc/* contains these folders:

- *iar*—for the IAR Embedded Workbench IDE.
- *mcux*—for the MCUXpresso IDE.
- *mdk*—for the µVision® Keil® IDE.

Each folder contains IDE-related configuration files for the PMSM project (project files, output executable, linker files, and so on). See the step-by-step tutorial on how to open, run, and debug the PMSM project in Section 6, “Building and debugging applications”.

The */build_ref_solutions/frdmkv31_enc/* folder contains also the *main.c*, *mcdrv_frdmkv31f.c*, and *pinmux.c* files.

- *main.c*—contains the basic application initialization (enabling interrupts), subroutines for accessing the MCU peripherals, and interrupt service routines. The FreeMASTER communication is performed in the background infinite loop.
- *mcdrv_frdmkv31f.c*—contains the peripherals’ settings and initializations described in Section 3.1.2, “Peripheral settings”.
- *mcdrv_frdmkv31f.h*—header file for the *mcdrv_frdmkv31f.c* file. This file contains the macros for changing the PWM period and the ADC channels assigned to the phase currents and the board voltage.
- *pinmux.c*—contains the settings of signal multiplexing and pin assignments. It is recommended to generate these files in the pin tool.

The */freemaster/* folder contains the auxiliary files for the MCAT tool. This folder contains also the FreeMASTER project file *pmsm_frdm_kv31.pmp*. Open this file in the FreeMASTER tool and use it to control the application. This PMSM reference project contains also the motor-identification algorithms and the scalar-control algorithm which can be used for motor-control tuning. See Section 9, “Remote control using FreeMASTER” for more information about the Motor Control Application Tuning (MCAT) tool.

The */src/* folder contains these subfolders with the source and header files for each project:

- *common*—contains common source and header files used in other motor-control projects.
- *projects*—contains the MCU-dependent source code.

The */src/common/* folder contains these subfolders common to the other motor-control projects:

- *freemaster_drv*—contains the FreeMASTER header and source files.
- *rtcesl*—contains the mathematical functions used in the project. This folder includes the required header files, source files, and library files used in the project. It contains three subfolders (each for a different IDE) that contain precompiled library files for the Arm Cortex-CM4 cores. The *rtcesl* folder is taken from the RTCESL release 4.5 and fully compatible with the official release. See www.nxp.com/rtcesl for more information about RTCESL.
- *mc_algorithms*—contains the main control algorithms used to control the FOC and speed control loop.
- *mc_drivers*—contains the source and header files used to initialize and run motor-control applications.
- *mc_identification*—contains the source code for the automated parameter-identification routines of the motor.
- *mc_state_machine*—contains the software routines that are executed when the application is in a particular state or state transition.
- *sdk*—contains the functions for the startup routines, header files, core-specific functions, linker files used by the IDEs available in this package, and the routines for the core clock settings.

- *state_machine*—contains the state machine functions for the FAULT, INITIALIZATION, STOP, and RUN states.
- *freemaster_cfg.h*—the FreeMASTER configuration file containing the FreeMASTER communication and features setup.

The */src/projects/* folder contains the reference solution software files. They specify the peripheral initialization routines, FreeMASTER initialization, motor definitions, and state machines. The source code contains a lot of comments. The functions of the files are explained in this list:

- *frdmkv31f/m1_pmsm_appconfig.h*—contains the definitions of constants for the application control processes, parameters of the motor and regulators, and the constants for other vector control-related algorithms. When you tailor the application for a different motor using the Motor Control Application Tuning (MCAT) tool, the tool generates this file at the end of the tuning process. The "*m1_*" (meaning motor 1) process is used in single-motor applications. This number designates the motor number in multi-motor applications. There are three of these files in the *frdmkv31f* folder. The *m1_pmsm_appconfig.h* file is valid for the Linix motor (*m1_pmsm_appconfig.h* is the same as *m1_pmsm_appconfig_linix.h*). If you are using the Teknic motor, rename the *m1_pmsm_appconfig_teknic.h* file to *m1_pmsm_appconfig.h.* Also, rename the *M1_params_pmsm_frdm-kv31f-hall_teknic.txt* file located in the *freemaster\mcat\param_files\* folder to *M1_params_pmsm_frdm-kv31f-hall*. Then rebuild and download the project to the target device, as shown in Section 6, "Building and debugging applications".
- *board/board.c*—contains the functions for the UART, GPIO, and SysTick initialization.
- *board/board.h*—contains the definitions of the board LEDs, buttons, UART instance used for FreeMASTER, and so on.
- *board/clock_config*—contains the CPU clock setup functions. These files are going to be generated by the clock tool in the future.
- *board/char_pwrstg*—contains the power-stage characterization.

5. Tools

Install the FreeMASTER Run-Time Debugging Tool 2.0 and one of the following IDEs on your PC to run and control the PMSM application properly:

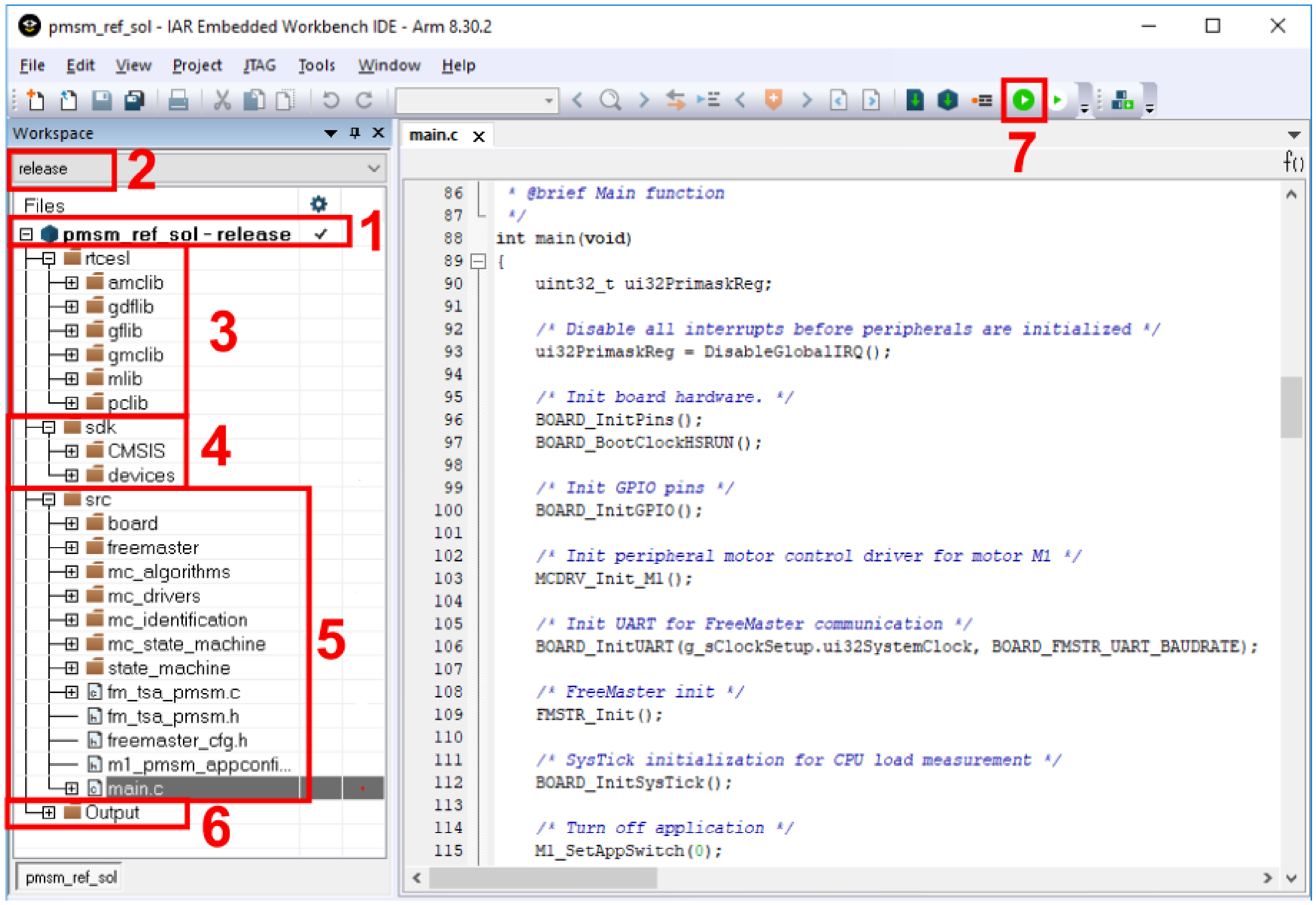
- IAR Embedded Workbench IDE v8.30.2 or higher.
- MCUXpresso v10.2.1.
- ARM-MDK - Keil µVision version 5.25.2.
- FreeMASTER Run-Time Debugging Tool 2.0.

6. Building and debugging applications

The package contains the PMSM reference project for the IAR Embedded Workbench, MCUXpresso, and µVision Keil IDEs. The release configuration is the default one and there are no special requirements to run and debug the demonstration applications.

6.1. IAR Embedded Workbench IDE

The first step is to open the project file. The project for the solution with the encoder sensor is in the *build_ref_solutions/frdmkv31_enc/iar* folder. To open the project, double-click the *pmsm_ref_sol.eww* file. The project opened in the IAR Embedded Workbench IDE is fully configured and includes all the necessary source and header files required by the application, such as startup code, clock configuration, and peripherals configuration. Point 1 in Figure 9 shows the IAR IDE workspace with the opened project.



```
 * @brief Main function
 */
int main(void)
{
    uint32_t ui32PrimaskReg;

    /* Disable all interrupts before peripherals are initialized */
    ui32PrimaskReg = DisableGlobalIRQ();

    /* Init board hardware. */
    BOARD_InitPins();
    BOARD_BootClockHSRUN();

    /* Init GPIO pins */
    BOARD_InitGPIO();

    /* Init peripheral motor control driver for motor M1 */
    MCDRV_Init_M1();

    /* Init UART for FreeMaster communication */
    BOARD_InitUART(g_sClockSetup.ui32SystemClock, BOARD_FMSTR_UART_BAUDRATE);

    /* FreeMaster init */
    FMSTR_Init();

    /* SysTick initialization for CPU load measurement */
    BOARD_InitSysTick();

    /* Turn off application */
    M1_SetAppSwitch(0);
```

Figure 9. IAR Embedded Workbench IDE

The project opened in the IAR Embedded Workbench IDE is fully configured and includes all necessary source and header files required by the application, such as startup code, clock configuration, and peripherals configuration. You can choose between two compiling conditions (Debug or Release), as shown in Figure 9, point 2. Each of the two conditions has its own setting:

- Debug—used for debugging, optimization has the “None – turned off” flag.
- Release—used for releasing, optimization has the “High – Highest optimization for speed” flag.

The source code (shown in Figure 9) includes these source files and folders:

- Point 3—the *rtcesl* library source folder contains header files for the mathematical and control functions used in this project. The theory about using and applying these functions is described in the user’s guides specific for each library. The user’s guides are at www.nxp.com/rtcesl.

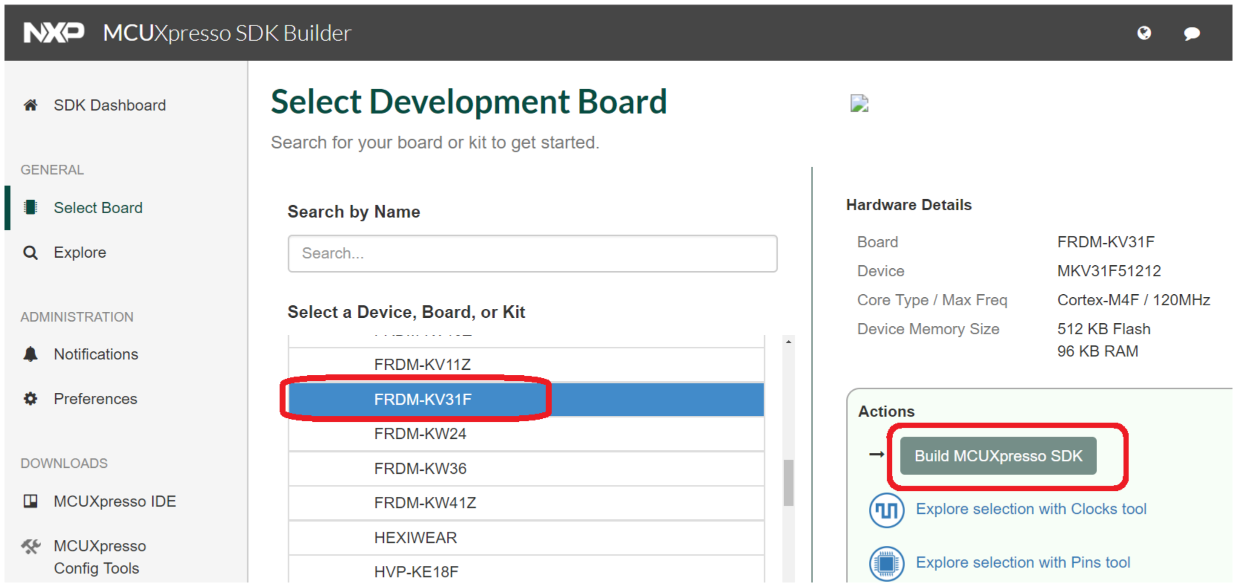
- Point 4—the */sdk/* folder contains the startup routines, system initialization and clock definition, linker file, and header file for the MCU. It also contains the basic CMSIS routines for interrupt handling.
- Point 5—the */src/* folder contains the application source code. The structure of this folder is different from that described in Section 4, "Project file structure". However, it is composed of the same files and organized to fit into the IDE workspace.

- Point 6—shows the output file generated by the compiler and ready to be used with the default debugger (P&E Micro—OpenSDA). This debugger is set as the default one and can be changed in the project options by right-clicking Point 1, selecting "Options", and clicking "Debugger".
- Point 7—start the project debugging by clicking Point 7.

6.2. MCUXpresso IDE

Firstly, the MCUXpresso SDK must be imported into the MCUXpresso IDE:

1. Go to mcuxpresso.nxp.com/en/welcome and click "Select Development Board".
2. Select the "FRDM-KV31F" board. On the right-hand side, click "Build MCUXpresso SDK".



Select Development Board

Search for your board or kit to get started.

Search by Name

Search...

Select a Device, Board, or Kit

FRDM-KV11Z
FRDM-KV31F
FRDM-KW24
FRDM-KW36
FRDM-KW41Z
HEXIWEAR
HVP-KE18F

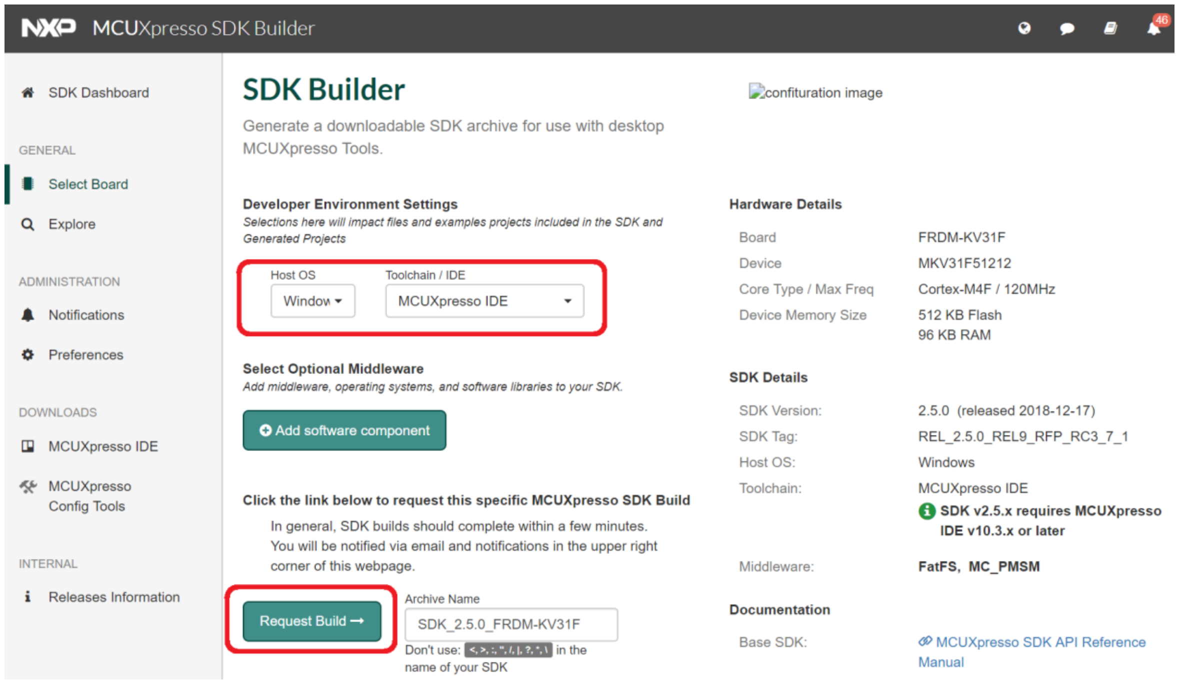
Hardware Details

Board	FRDM-KV31F
Device	MKV31F51212
Core Type / Max Freq	Cortex-M4F / 120MHz
Device Memory Size	512 KB Flash 96 KB RAM

Actions

Figure 10. MCUXpresso SDK builder

3. Select the host OS and IDE and click "Request Build", as shown in Figure 11. It is also possible to add the MC_PMSM software component that contains the SDK motor-control examples for several Kinetis and i.MX boards.



SDK Builder

Generate a downloadable SDK archive for use with desktop MCUXpresso Tools.

Developer Environment Settings

Selections here will impact files and examples projects included in the SDK and Generated Projects

Host OS: Window
Toolchain / IDE: MCUXpresso IDE

Select Optional Middleware

Add middleware, operating systems, and software libraries to your SDK.

Click the link below to request this specific MCUXpresso SDK Build

In general, SDK builds should complete within a few minutes. You will be notified via email and notifications in the upper right corner of this webpage.

Archive Name: SDK_2.5.0_FRDM-KV31F

Don't use: <, >, :, ", /, \, |, ?, *, \ in the name of your SDK

Hardware Details

Board	FRDM-KV31F
Device	MKV31F51212
Core Type / Max Freq	Cortex-M4F / 120MHz
Device Memory Size	512 KB Flash 96 KB RAM

SDK Details

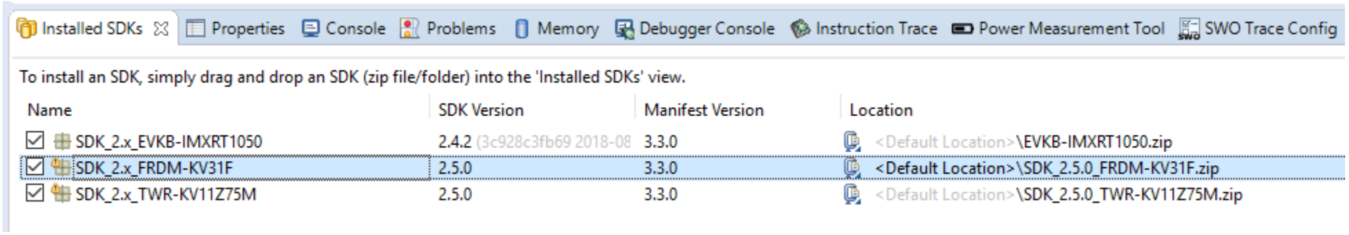
SDK Version:	2.5.0 (released 2018-12-17)
SDK Tag:	REL_2.5.0_REL9_RFP_RC3_7_1
Host OS:	Windows
Toolchain:	MCUXpresso IDE **SDK v2.5.x requires MCUXpresso IDE v10.3.x or later**
Middleware:	**FatFS, MC_PMSM**

Documentation

Base SDK:	MCUXpresso SDK API Reference Manual

Figure 11. MCUXpresso SDK—"Request Build” button

4. After building, download the SDK archive and open the MCUXpresso IDE. At the bottom of the workspace, there is a panel with several tools. Click the “Installed SDKs” tab (Figure 12) and drag and drop the downloaded file into the “SDKs” field.



To install an SDK, simply drag and drop an SDK (zip file/folder) into the 'Installed SDKs' view.

Name	SDK Version	Manifest Version	Location
✓ SDK_2.x_EVKB-IMXRT1050	2.4.2 (3c928c3fb69 2018-08	3.3.0	<Default Location>\EVKB-IMXRT1050.zip
✓ SDK_2.x_FRDM-KV31F	2.5.0	3.3.0	<Default Location>\SDK_2.5.0_FRDM-KV31F.zip
✓ SDK_2.x_TWR-KV11Z75M	2.5.0	3.3.0	<Default Location>\SDK_2.5.0_TWR-KV11Z75M.zip

Figure 12. MCUXpresso installed SDKs

5. After the import, the SDK successfully installs into the MCUXpresso IDE.

The second step is to import the project to the IDE. At the top of the MCUXpresso IDE, click “File->Import”. When the new dialog box appears, select “Existing Projects into Workspace” and click the “Next” button (Figure 13).

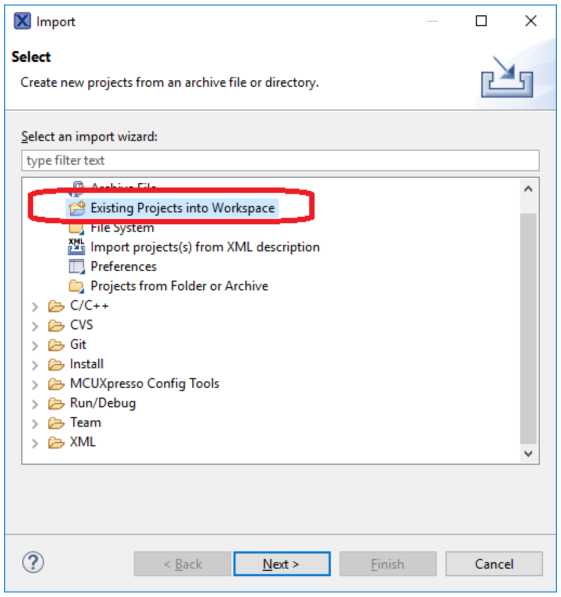
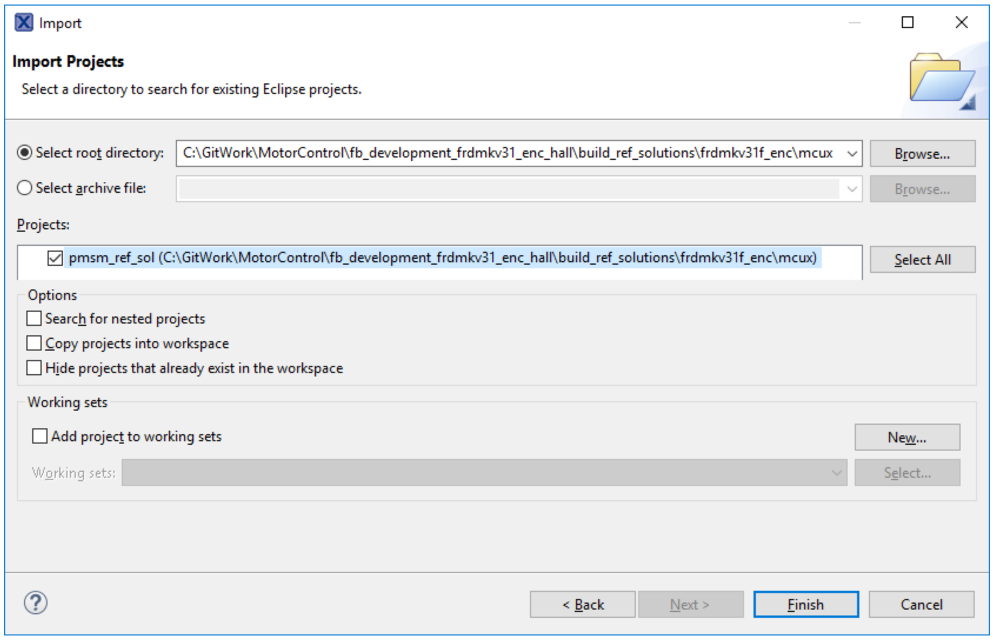


Figure 13. MCUXpresso import

The next step is selecting the folder that contains the project files. Click the “Browse” button and find the *mcux* folder on your hard disk. For the PMSM project, it is in the *build_ref_solutions/frdmkv31f_hall/mcux* folder (Figure 14). Confirm the import by clicking the “Finish” button in the bottom right-hand corner.



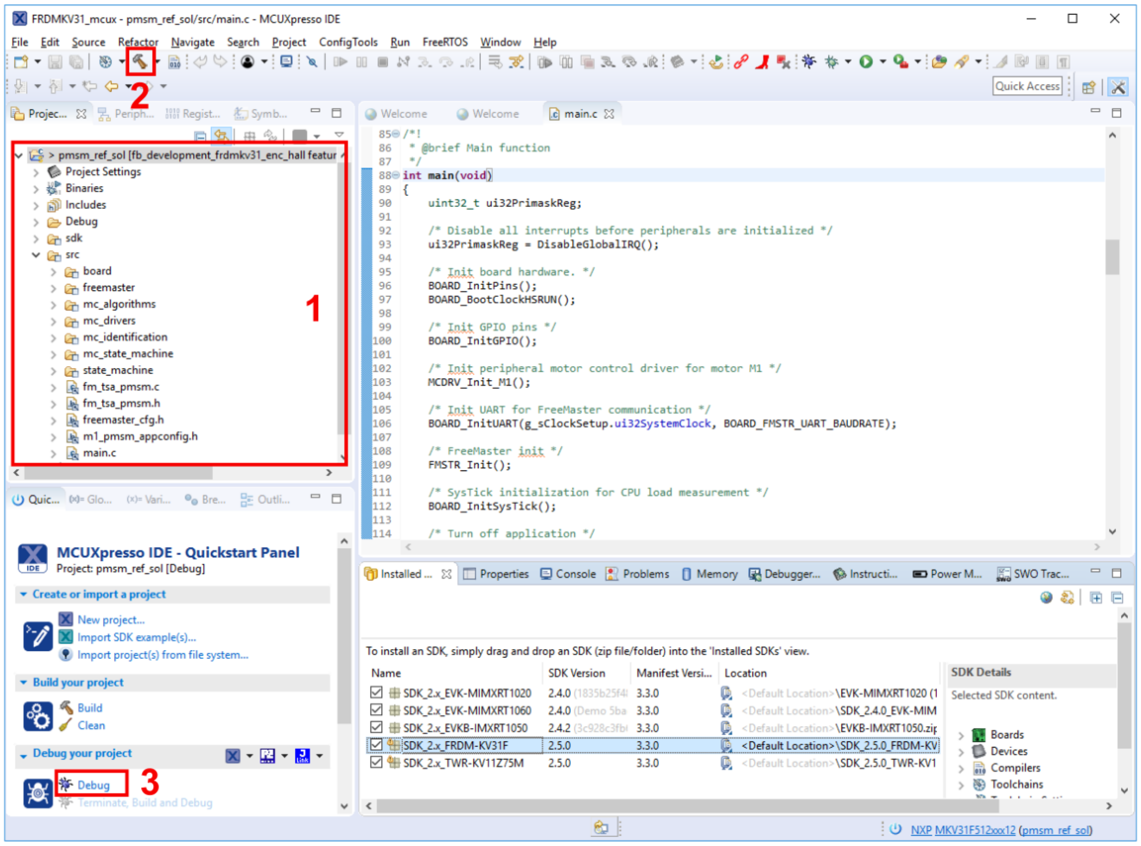
Select root directory: C:\GitWork\MotorControl\fb_development_frdmkv31_enc_hall\build_ref_solutions\frdmkv31f_enc\mcux
Select archive file:
Projects:
☑ pmsm_ref_sol (C:\GitWork\MotorControl\fb_development_frdmkv31_enc_hall\build_ref_solutions\frdmkv31f_enc\mcux)
Options
☐ Search for nested projects
☐ Copy projects into workspace
☐ Hide projects that already exist in the workspace
Working sets
☐ Add project to working sets
Working sets:

Figure 14. MCUXpresso import

Now you see the MCUXpresso workspace with the PMSM project imported (Figure 15). The project in the MCUXpresso IDE is fully configured and includes all necessary source and header files required by the application, such as the startup code, clock configuration, and peripherals’ configuration.

- Point 1—the imported project file structure.

- Point 2—the “Build” button. You may also choose the build configuration using the little arrow on this button. There are two build configurations available (Debug and Release). Each of the two conditions has its own settings (described below).
- Point 3—the “Debug” button. This button downloads the compiled project into the RAM/FLASH MCU memory. It also creates the debug configuration (if it does not exist).



```
/*!
 * @brief Main function
 */
int main(void)
{
    uint32_t ui32PrimaskReg;

    /* Disable all interrupts before peripherals are initialized */
    ui32PrimaskReg = DisableGlobalIRQ();

    /* Init board hardware. */
    BOARD_InitPins();
    BOARD_BootClockHSRUN();

    /* Init GPIO pins */
    BOARD_InitGPIO();

    /* Init peripheral motor control driver for motor M1 */
    MCDRV_Init_M1();

    /* Init UART for FreeMaster communication */
    BOARD_InitUART(g_sClockSetup.ui32SystemClock, BOARD_FMSTR_UART_BAUDRATE);

    /* FreeMaster init */
    FMSTR_Init();

    /* SysTick initialization for CPU load measurement */
    BOARD_InitSysTick();

    /* Turn off application */
```

To install an SDK, simply drag and drop an SDK (zip file/folder) into the 'Installed SDKs' view.

Name	SDK Version	Manifest Versi...	Location
☑ SDK_2.x_EVK-MIMXRT1020	2.4.0 (1835b25f4	3.3.0	<Default Location>\EVK-MIMXRT1020 (1
☑ SDK_2.x_EVK-MIMXRT1060	2.4.0 (Demo 5ba	3.3.0	<Default Location>\SDK_2.4.0_EVK-MIM
☑ SDK_2.x_EVKB-IMXRT1050	2.4.2 (3c928c3fb	3.3.0	<Default Location>\EVKB-IMXRT1050.zip
☑ SDK_2.x_FRDM-KV31F	2.5.0	3.3.0	<Default Location>\SDK_2.5.0_FRDM-KV
☑ SDK_2.x_TWR-KV11Z75M	2.5.0	3.3.0	<Default Location>\SDK_2.5.0_TWR-KV1

Figure 15. MCUXpresso workspace

There are two project configurations:

- Debug—used for debugging. The optimization has the “None—turned off” flag.
- Release—used for releasing. The optimization has the “High—highest optimization for speed” flag.

To run an application, choose your configuration, build the project, and click the “Debug” button to enter the debug session. In debug section, use “Resume”, “Suspend”, and “Terminate” buttons to control the application (Figure 16).

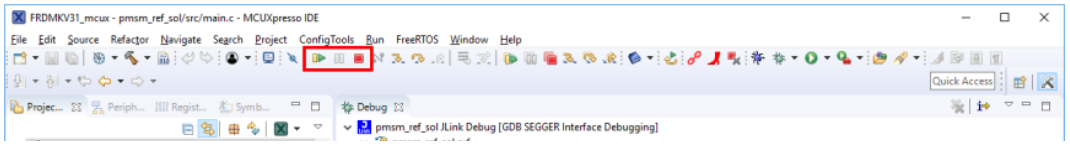
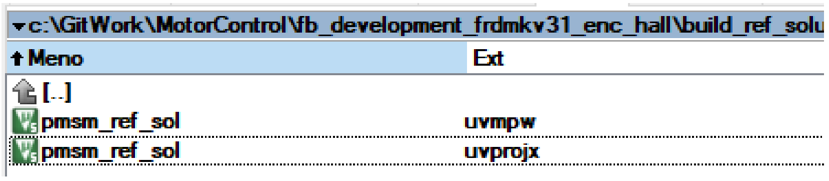

FRDMKV31_mcux - pmsm_ref_sol/src/main.c - MCUXpresso IDE
File Edit Source Refactor Navigate Search Project ConfigTools Run FreeRTOS Window Help
Quick Access
Projec...
Periph...
Regist...
Symb...
Debug
pmsm_ref_sol JLink Debug [GDB SEGGER Interface Debugging]

Figure 16. MCUXpresso debug workspace

6.3. Arm-MDK Keil µVision

The Arm-MDK Keil µVision (Keil) is an IDE that you can use to develop and test software for NXP MCUs. It supports a wide range of Kinetis devices, such as the powerful K series, low-power KL series, and KV series targeted at motor control. Keil includes tools for compiling, linking, and debugging of source code. Keil supports a wide range of debuggers, such as P&E Micro or J-Link (and others). You can download the latest release of Keil from the official Keil website www2.keil.com/mdk5/uvision. For installation and configuration, see the *Kinetis Design Studio V3.0.0 User's Guide* (document KDSUG).

To open a project, double-click the *pmsm_ref_sol.uvprojx* project file located in the *build_ref_solutions/frdmkv31_enc/mdk* folder:

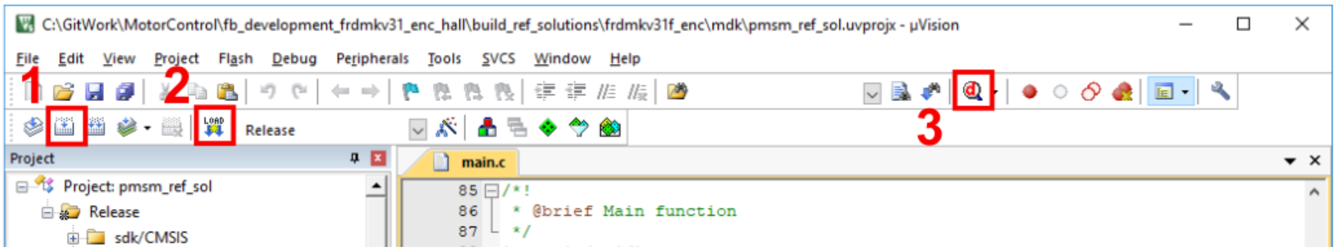


c:\GitWork\MotorControl\fb_development_frdmkv31_enc_hall\build_ref_solu

Meno	Ext
[..]	
pmsm_ref_sol	uvmpw
pmsm_ref_sol	uvprojx

Figure 17. Keil project location

Now the project is open. Click the "Build" button (Point 1) to compile the project. Click the "Download" button (Point 2) to download the code to the target. Then click the "Debug" button to enter the debug session (Point 3):


C:\GitWork\MotorControl\fb_development_frdmkv31_enc_hall\build_ref_solutions\frdmkv31f_enc\mdk\pmsm_ref_sol.uvprojx - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
1
2
3
Release
Project
Project: pmsm_ref_sol
Release
sdk/CMSIS
main.c

```
/*!
 * @brief Main function
 */
```

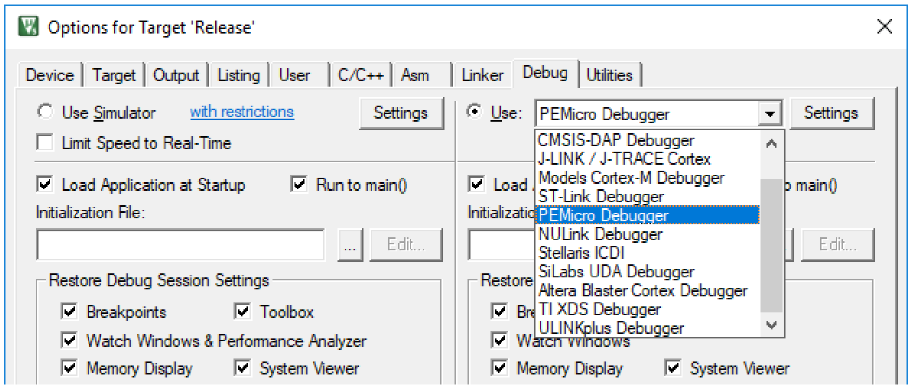
Figure 18. Keil PMSM project

Before downloading the code to the target, select the proper target device. If you are using the P&E OpenSDA debugger, select the KV31M512M12 device.

There are two project configurations:

- Debug—used for debugging. The optimization has the "None – turned off" flag.
- Release—used for releasing. The optimization has the "High – highest optimization for speed" flag.

Use the predefined debugger (such as P&E Micro—OpenSDA) or choose a different debugger from the menu. In the top list menu, select "Project-> Options for Target release/debug -> Debug" to define a different type of debugger. You can open this menu also using the "Alt+F7" key shortcut. To run an application in the debug session, press the "F5" key.



Use Simulator with restrictions

Limit Speed to Real-Time

☑ Load Application at Startup ☑ Run to main()

Initialization File:

Restore Debug Session Settings

☑ Breakpoints ☑ Toolbox

☑ Watch Windows & Performance Analyzer

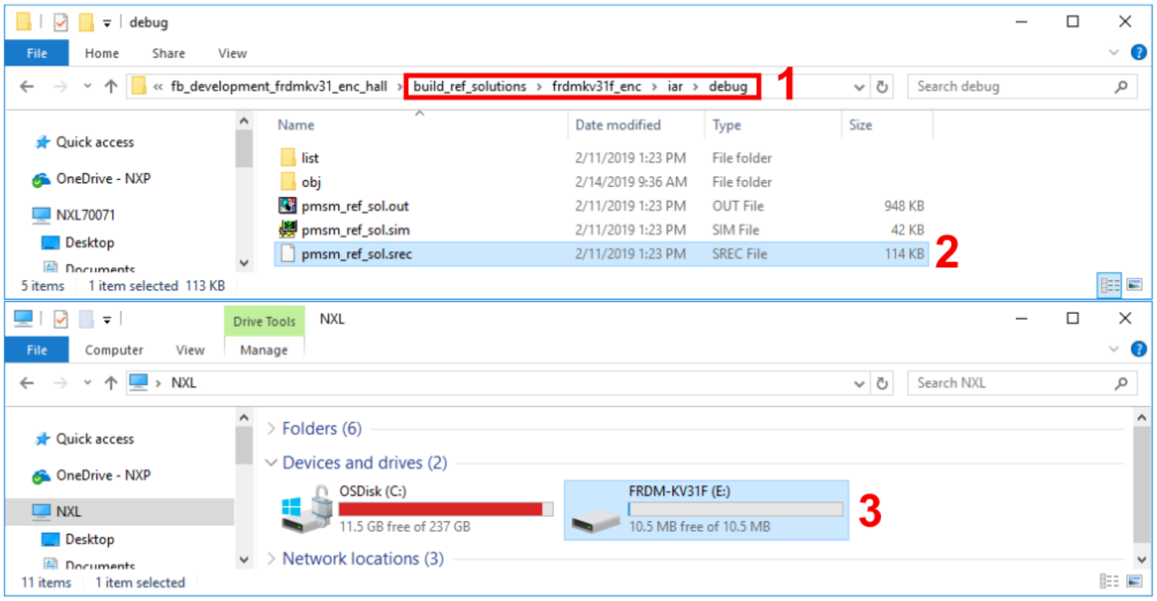
☑ Memory Display ☑ System Viewer

[✓ Use: PEMicro Debugger]

Figure 19. Keil—"Options for Target”

6.4. OpenSDA debugger

NXP development boards include the OpenSDA serial and debugger adapter, which is used as a bridge for serial and debug communication between the target processor and the host computer. This debugger provides a virtual serial port on the host computer, which can be accessed as a “COM” port.
The embedded target is connected to the UART peripheral. In the same time, you can control the debug interface that controls the JTAG or SWD debug interfaces in the target development platform.
The OpenSDA debug interface provides the Mass Storage Device Flash Programmer (MSD Flash Programmer), which is an easy way to program applications into the flash memory of the target processor. It appears as a removable drive in the host operating system, with a volume label that matches the board name. The raw binary Motorola S-Record files that are copied to the drive are programmed directly into the target device memory. The MSD Flash Programmer is designed to program a specific target configuration. It does not support verification or configuration, and it is not recommended as a production programmer.



« fb_development_frdmkv31_enc_hall › build_ref_solutions › frdmkv31f_enc › iar › debug

Name	Date modified	Type	Size
list	2/11/2019 1:23 PM	File folder	
obj	2/14/2019 9:36 AM	File folder	
pmsm_ref_sol.out	2/11/2019 1:23 PM	OUT File	948 KB
pmsm_ref_sol.sim	2/11/2019 1:23 PM	SIM File	42 KB
pmsm_ref_sol.srec	2/11/2019 1:23 PM	SREC File	114 KB

5 items 1 item selected 113 KB

OSDisk (C:) 11.5 GB free of 237 GB

FRDM-KV31F (E:) 10.5 MB free of 10.5 MB

11 items 1 item selected

Figure 20. OpenSDA MSD

To load the generated application directly to the target MCU, perform these steps (applicable for Windows® OS):

1. Open Windows Explorer.
2. Locate the generated S-record file (**.srec*, **.s19*, or **.bin* file extension), Point 1 (Figure 20).
3. Drag/drop or copy/paste the selected S-record file (Figure 20Point 2) to the MSD removable drive with the volume labelled as the target hardware (Point 3).
4. After successful programming, the embedded application executes automatically.
5. Reconnect the target device.

6.5. Compiler warnings

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

In these projects (in IDEs), some warnings are suppressed. These warnings are mentioned below and the other warnings are set to a standard configuration.

The IAR Embedded Workbench IDE suppresses these warnings:

- Pa082—undefined behavior; the order of volatile accesses is not defined in this statement.

The MCUXpresso IDE suppresses these warnings:

- Unused function—function is defined, but not used.
- Uninitialized—variable is used, but not initialized.
- Main—due to optimization, main is void (not integer). It does not return a value.

The Arm-MDK Keil uVision IDE suppresses these warnings:

- 66—enumeration value is out of "int" range.
- 1035—single-precision operand implicitly converted to double-precision.

There are no other warnings shown during the compiling process by default.

7. Motor-control peripheral initialization

The motor-control peripherals are initialized by calling the *MCDRV_Init_M1()* function during MCU startup and before the peripherals are used. All initialization functions are in the *mcdrv_frdmkv31f.c* source file and the *mcdrv_ frdmkv31f.h* header file. The definitions specified by the user are also in these files. The features provided by the functions are the 3-phase PWM generation and 3-phase current measurement, as well as the DC-bus voltage and auxiliary quantity measurement. The principles of both the 3-phase current measurement and the PWM generation using the Space Vector Modulation (SVM) technique are described in *Sensorless PMSM Field-Oriented Control* (document DRM148).

The *mcdrv_ frdmkv31f.h* header file provides several macros, which can be defined by the user:

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

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

- The available APIs for the ADC are:
  - *mcdrv_adc_t*—MCDRV ADC structure data type.
  - *bool_t M1_MCDRV_ADC_PERIPH_INIT()*—this function is by default called during the ADC peripheral initialization procedure invoked by the *MCDRV_Init_M1()* function and should not be called again after the peripheral initialization is done.
  - *bool_t M1_MCDRV_CURR_3PH_CHAN_ASSIGN(mcdrv_adc_t*)*—calling this function assigns proper ADC channels for the next 3-phase current measurement based on the SVM sector. 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 standstill motor and filters them using moving average filters. The goal is to obtain the value of the measurement offset. The length of the window for moving the average filters is set to eight samples by default. This function always returns true.
  - *bool_t M1_MCDRV_CURR_3PH_CALIB_SET(mcdrv_adc_t*)*—this function asserts the phase-current measurement offset values to the internal registers. Call this function after a sufficient number of *M1_MCDRV_CURR_3PH_CALIB()* calls. This function always returns true.
  - *bool_t M1_MCDRV_ADC_GET(mcdrv_adc_t*)*—this function reads and calculates the actual values of the 3-phase currents, DC-bus voltage, and auxiliary quantity. This function always returns true.

- The available APIs for the PWM are:
  - *mcdrv_pwma_pwm3ph_t*—MCDRV PWM structure data type.
  - *bool_t M1_MCDRV_PWM_PERIPH_INIT*—this function is by default called during the PWM periphery initialization procedure invoked by the *MCDRV_Init_M1()* function.
  - *bool_t M1_MCDRV_PWM3PH_SET(mcdrv_pwma_pwm3ph_t*)*—this function updates the PWM phase duty cycles based on the required values. This function always returns true.
  - *bool_t M1_MCDRV_PWM3PH_EN(mcdrv_pwma_pwm3ph_t*)*—calling this function enables all PWM channels. This function always returns true.
  - *bool_t M1_MCDRV_PWM3PH_DIS (mcdrv_pwma_pwm3ph_t*)*—calling this function disables all PWM channels. This function always returns true.
  - *bool_t M1_MCDRV_PWM3PH_FLT_GET(mcdrv_pwma_pwm3ph_t*)*—this function returns the state of the over-current fault flags and automatically clears the flags (if set). This function returns true when an over-current event occurs. Otherwise, it returns false.
- The available APIs for the FlexTimer encoder are:
  - *mcdrv_ftm_enc_t*—MCDRV FTM encoder structure data type.
  - *bool_t M1_MCDRV_TMR_SENSOR_INIT()*—this function is by default called during the FTM periphery initialization procedure invoked by the *MCDRV_Init_M1()* function.
  - *bool_t M1_MCDRV_QD_GET(mcdrv_qd_enc_t*)*—this function returns the actual position and speed. This function always returns true.
  - *bool_t M1_MCDRV_QD_CLEAR(mcdrv_qd_enc_t*)*—this function clears the internal variables and decoder counter. This function always returns true.
- The available APIs for the FlexTimer hall sensors are:
  - *mcdrv_ftm_enc_t*—MCDRV FTM encoder structure data type.
  - *bool_t M1_MCDRV_TMR_SENSOR_INIT()*—this function is by default called during the FTM periphery initialization procedure invoked by the *MCDRV_Init_M1()* function.
  - *bool_t M1_MCDRV_FTM_HALL_GET_SPEED_POS(mcdrv_ftm_hs_t*)*—this function returns the actual position and speed. This function always returns true.
  - *bool_t M1_MCDRV_FTM_HALL_SET_INIT_POS(mcdrv_ftm_hs_t*)*—this function calculates the electrical position based on the current hall state. This function always returns true.
  - *bool_t M1_MCDRV_HALL_CLEAR(mcdrv_ftm_hs_t*)*—this function clears the internal variables. This function always returns true.

8. User interface

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

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

The application can be controlled the using these two interfaces:

- The button on the FRDM-KV31F development board—SW2 (controlling the demo mode).
- Remote control using FreeMASTER:
  - Using the Motor Control Application Tuning (MCAT) interface in the “Control Structure” tab or the “Application control” tab (controlling the demo mode).
  - By setting a variable in the FreeMASTER Variable Watch.

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

9. Remote control using FreeMASTER

This section provides information about the tools and recommended procedures to control the sensor/sensorless PMSM Field-Oriented Control (FOC) application using FreeMASTER. The application contains the embedded-side driver of the FreeMASTER real-time debug monitor and data visualization tool for communication with the PC. It supports non-intrusive monitoring, as well as the modification of target variables in real time, which is very useful for the algorithm tuning. Besides the target-side driver, the FreeMASTER tool requires the installation of the PC application as well. You can download FreeMASTER 2.0 at www.nxp.com/freemaster. To run the FreeMASTER application including the MCAT tool, double-click the *pmsm_frdm_kv31.pmp* file located in the *freemaster/* folder. The FreeMASTER application starts and the environment is created automatically, as defined in the **.pmp* file.

9.1. Establishing FreeMASTER communication

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

1. Download the project from your chosen IDE to the MCU and run it, as shown in Section 6, “Building and debugging applications”.
2. Open the FreeMASTER file *pmsm_frdm_kv31.pmp* located in the *freemaster/* folder. The PMSM project uses the TSA by default, so it is not necessary to select a symbol file for FreeMASTER.
3. Click the communication button (the red “STOP” button in the top left-hand corner) to establish the communication.

2.

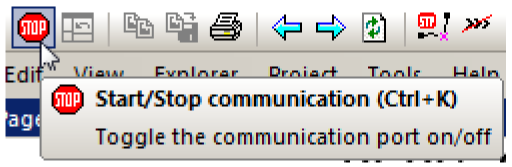

Start/Stop communication (Ctrl+K)
Toggle the communication port on/off
Edit
View
Explorer
Project
Tools
Help

Figure 21. Red “STOP” button placed in top left-hand corner

4. If the communication is established successfully, the FreeMASTER communication status in the bottom right-hand corner changes from “Not connected” to “RS232 UART Communication; COMxx; speed=115200”. Otherwise, the FreeMASTER warning popup window appears.

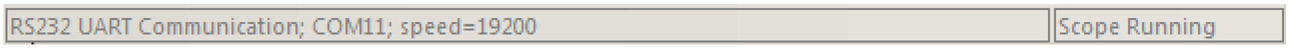
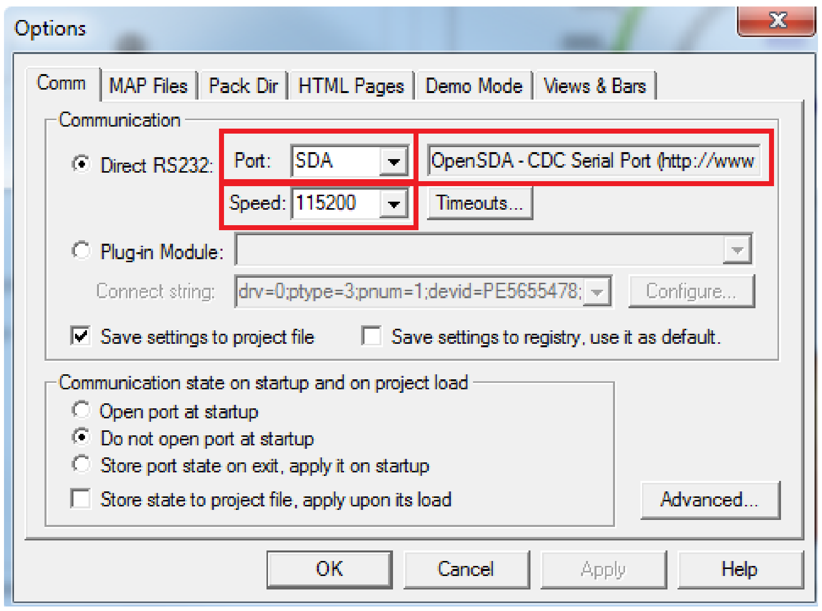

RS232 UART Communication; COM11; speed=19200 | Scope Running

Figure 22. FreeMASTER—communication is established successfully

5. Control the PMSM motor using the MCAT “Control structure” tab, the MCAT “Application demo control” tab, or by directly writing to a variable in a variable watch. (Section 9.2.1, “Control structure—“Control Struc” tab” and Section 9.2.2, “Application demo control—“App control” tab”)
6. If you rebuild and download the new code to the target, turn the FreeMASTER application off and on.

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

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



Communication

(●) Direct RS232: Port: SDA — OpenSDA - CDC Serial Port (http://www
Speed: 115200

( ) Plug-in Module:
Connect string: drv=0;ptype=3;pnum=1;devid=PE5655478;

[✓] Save settings to project file
[ ] Save settings to registry, use it as default.

Communication state on startup and on project load

( ) Open port at startup
(●) Do not open port at startup
( ) Store port state on exit, apply it on startup
[ ] Store state to project file, apply upon its load

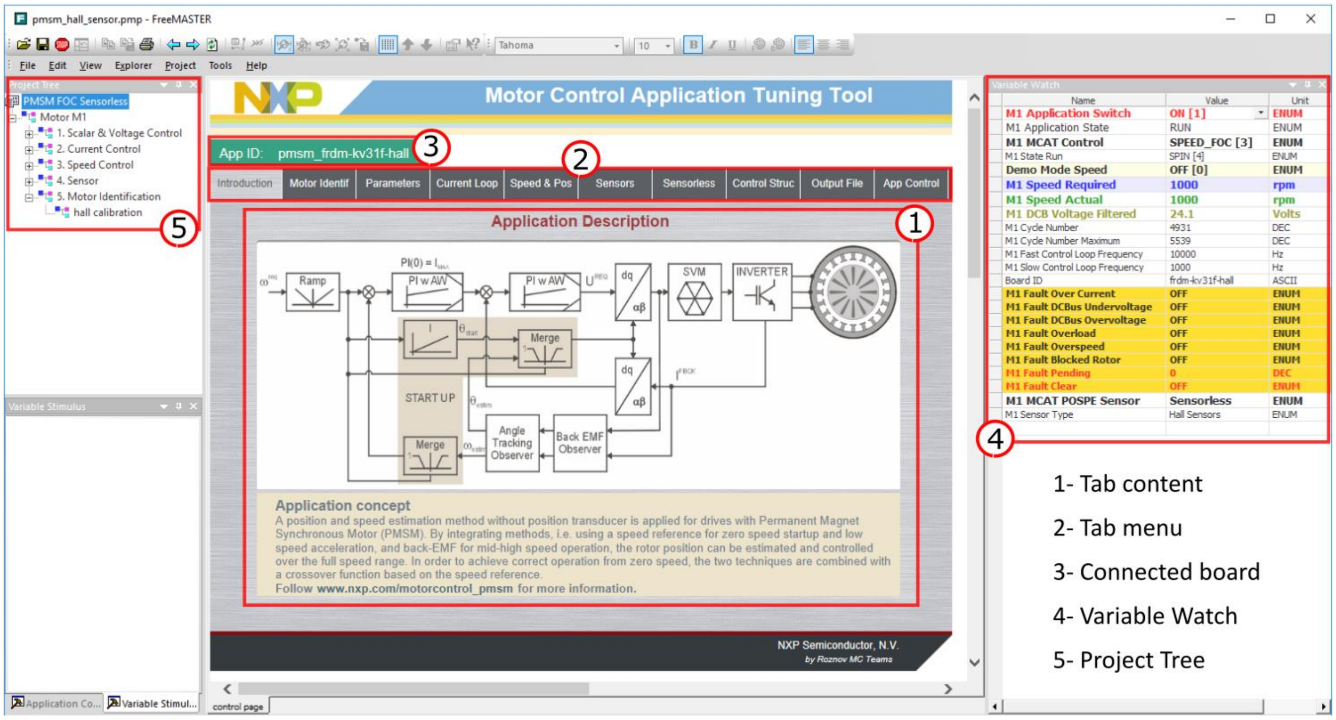
Figure 23. FreeMASTER communication setup window

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.

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.

9.2. MCAT FreeMASTER interface (Motor Control Application Tuning)

The PMSM sensor/sensorless FOC application can be easily controlled and tuned using the Motor Control Application Tuning (MCAT) plug-in for PMSM. The MCAT for PMSM is a user-friendly modular page, which runs within FreeMASTER. The tool consists of the tab menu, tuning mode selector, and workspace shown in Figure 24. Each tab from the tab menu represents one sub-module which enables you to tune or control different aspects of the application. Besides the MCAT page for PMSM, several scopes, recorders, and variables in the project tree are predefined in the FreeMASTER project file to further simplify the motor parameter tuning and debugging. When the FreeMASTER is not connected to the target, the "App ID" line shows "offline". When the communication with the target MCU is established using a correct software, the "App ID" line displays the board name "pmsm_frdm-kv31f-hall" and all stored parameters for the given MCU are loaded.



Application concept

A position and speed estimation method without position transducer is applied for drives with Permanent Magnet Synchronous Motor (PMSM). By integrating methods, i.e. using a speed reference for zero speed startup and low speed acceleration, and back-EMF for mid-high speed operation, the rotor position can be estimated and controlled over the full speed range. In order to achieve correct operation from zero speed, the two techniques are combined with a crossover function based on the speed reference.
Follow www.nxp.com/motorcontrol_pmsm for more information.

Name	Value	Unit
M1 Application Switch	ON [1]	ENUM
M1 Application State	RUN	ENUM
M1 MCAT Control	SPEED_FOC [3]	ENUM
M1 State Run	SPIN [4]	ENUM
Demo Mode Speed	OFF [0]	ENUM
M1 Speed Required	1000	rpm
M1 Speed Actual	1000	rpm
M1 DCB Voltage Filtered	24.1	Volts
M1 Cycle Number	4931	DEC
M1 Cycle Number Maximum	5539	DEC
M1 Fast Control Loop Frequency	10000	Hz
M1 Slow Control Loop Frequency	1000	Hz
Board ID	frdm-kv31f-hall	ASCII
M1 Fault Over Current	OFF	ENUM
M1 Fault DCBus Undervoltage	OFF	ENUM
M1 Fault DCBus Overvoltage	OFF	ENUM
M1 Fault Overload	OFF	ENUM
M1 Fault Overspeed	OFF	ENUM
M1 Fault Blocked Rotor	OFF	ENUM
M1 Fault Pending	0	DEC
M1 Fault Clear	OFF	ENUM
M1 MCAT POSPE Sensor	Sensorless	ENUM
M1 Sensor Type	Hall Sensors	ENUM

1- Tab content
2- Tab menu
3- Connected board
4- Variable Watch
5- Project Tree

Figure 24. MCAT layout

In the default configuration, these tabs are available:

- "Introduction"—welcome page with the PMSM sensor/sensorless FOC diagram and a short description of the application.
- "Motor Identif"—PMSM semi-automated parameter measurement control page. The PMSM parameter identification is more closely described further on in this document.
- "Parameters"—this page enables you to modify the motor parameters, specification of hardware and application scales, alignment, and fault limits.
- "Current Loop"—current loop PI controller gains and output limits.
- "Speed & Pos"—this tab contains fields for the specification of the speed controller proportional and integral gains, as well as the output limits and parameters of the speed ramp. The position proportional controller constant is also set here.

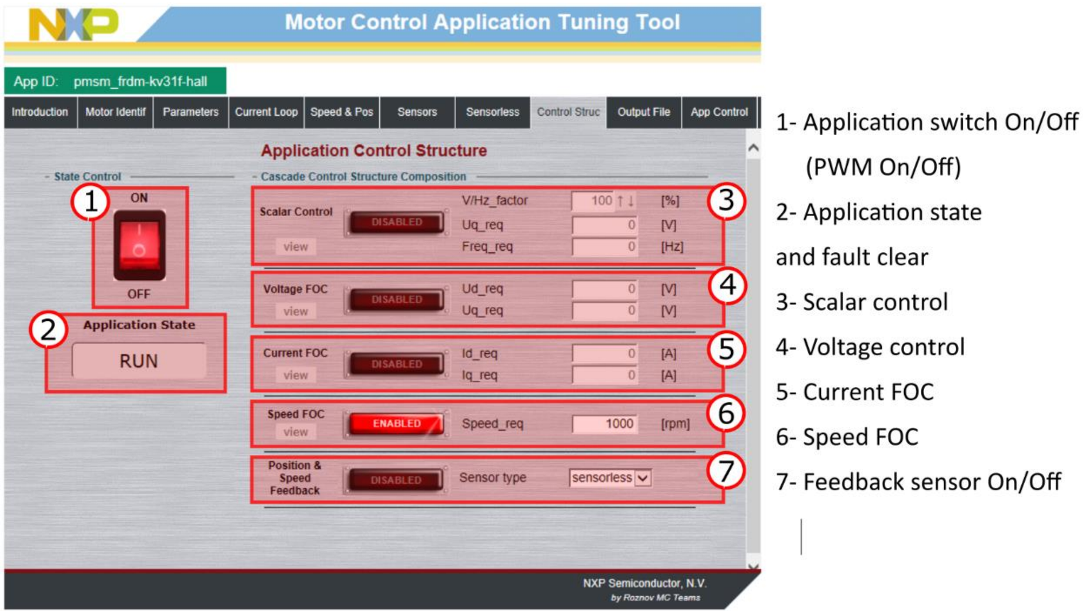
- “Sensors”—this page contains the encoder parameters and position observer parameters.
- “Sensorless”—this page enables you to tune the parameters of the BEMF observer, tracking observer, and open-loop startup.
- “Control Struc”—this application control page enables you to select and control the PMSM using different techniques (scalar—Volt/Hertz control, voltage FOC, current FOC, speed FOC, and position FOC). The application state is also shown in this tab.
- “Output file”—this tab shows all the calculated constants that are required by the PMSM sensor/sensorless FOC application. It is also possible to generate the *m1_acim_appconfig.h* file, which is then used to preset all application parameters permanently at the project rebuild.
- “App page”—this tab contains the graphical elements like the speed gauge, DC-bus voltage measurement bar, and variety of switches which enable a simple, quick, and user-friendly application control. The fault clearing and the demo mode (which sets various predefined required speeds and positions over time) can be also controlled from here.

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

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

9.2.1. Control structure—“Control Struc” tab

The application can be controlled through the “Control Struc” tab, which is shown in Figure 25. The state control area on the left side of the screen shows the current application state and enables you to turn the main application switch on or off (turning a running application off disables all PWM outputs). The “Cascade Control Structure” area is placed in the right-hand side of the screen. Here you can choose between the scalar control and the FOC control using the appropriate buttons. The selected parts of the FOC cascade structure can be enabled by selecting “Voltage FOC”, “Current FOC”, and “Speed FOC” (sensor/sensorless). This is useful for application tuning and debugging.



App ID: pmsm_frdm-kv31f-hall

Application Control Structure

State Control: ON / OFF

Application State: RUN

Cascade Control Structure Composition

Scalar Control	DISABLED	V/Hz_factor	100	[%]
		Uq_req	0	[V]
		Freq_req	0	[Hz]
Voltage FOC	DISABLED	Ud_req	0	[V]
		Uq_req	0	[V]
Current FOC	DISABLED	Id_req	0	[A]
		Iq_req	0	[A]
Speed FOC	ENABLED	Speed_req	1000	[rpm]
Position & Speed Feedback	DISABLED	Sensor type	sensorless	

1- Application switch On/Off (PWM On/Off)
2- Application state and fault clear
3- Scalar control
4- Voltage control
5- Current FOC
6- Speed FOC
7- Feedback sensor On/Off

Figure 25. MCAT for PMSM control page

The scalar control diagram is shown in Figure 26. It is the simplest type of motor-control techniques. The ratio between the magnitude of the stator voltage and the frequency must be kept at the nominal value. Hence, the control method is sometimes called Volt per Hertz (or V/Hz). Pay attention when entering the required voltage and frequency in the “Expert” tuning mode. The ratio is kept constant in the “Basic” mode and the only required inputs are voltage and frequency. The position estimation BEMF observer and tracking observer algorithms (see *Sensorless PMSM Field-Oriented Control* (document DRM148) for more information) run in the background, even if the estimated position information is not directly used. This is useful for the BEMF observer tuning.

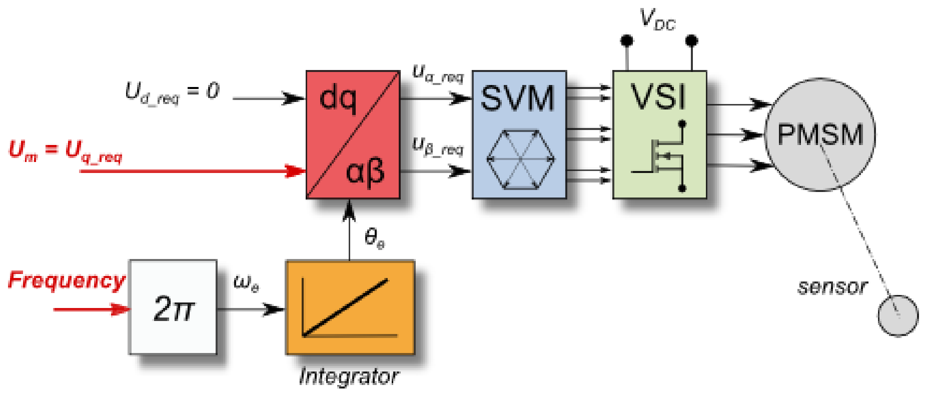


Figure 26. Scalar control modes

The block diagram of the voltage FOC is in Figure 27. Unlike the scalar control, the position feedback is closed using the BEMF observer and the stator voltage magnitude is not dependent on the motor speed. Both the *d*-axis and *q*-axis stator voltages can be specified in the “Ud_req” and “Uq_req” fields. This control method is useful for the BEMF observer functionality check.

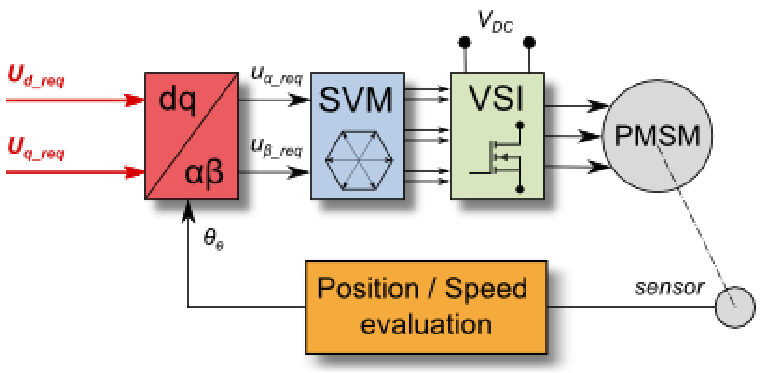


Figure 27. Voltage FOC control mode

The current FOC (or torque) control requires the rotor position feedback and the currents transformed into a d-q reference frame. There are two reference variables (“Id_req” and “Iq_req”) available for the motor control, as shown in the block diagram in Figure 28. The *d*-axis current component i_{sd_req} is responsible for the rotor flux control. The *q*-axis current component of the current i_{sq_req} generates torque and, by its application, the motor starts running. By changing the polarity of the current i_{sq_req}, the motor changes the direction of rotation. Supposing that the BEMF observer is tuned correctly, the current PI controllers can be tuned using the current FOC control structure.

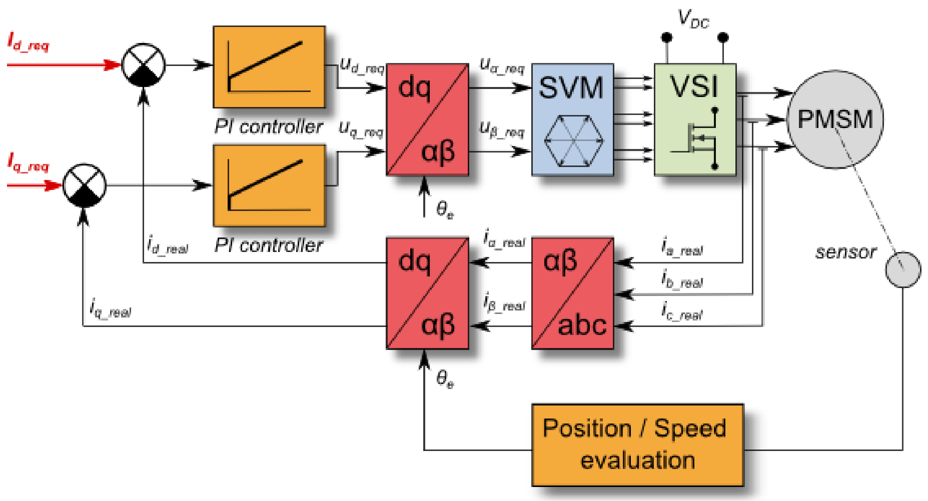


Figure 28. Current (torque) control mode

The speed PMSM sensor/sensorless FOC (its diagram is shown in Figure 29) is activated by enabling the speed FOC control structure. Enter the required speed into the “Speed_req” field. The *d*-axis current reference is held at 0 during the entire FOC operation.

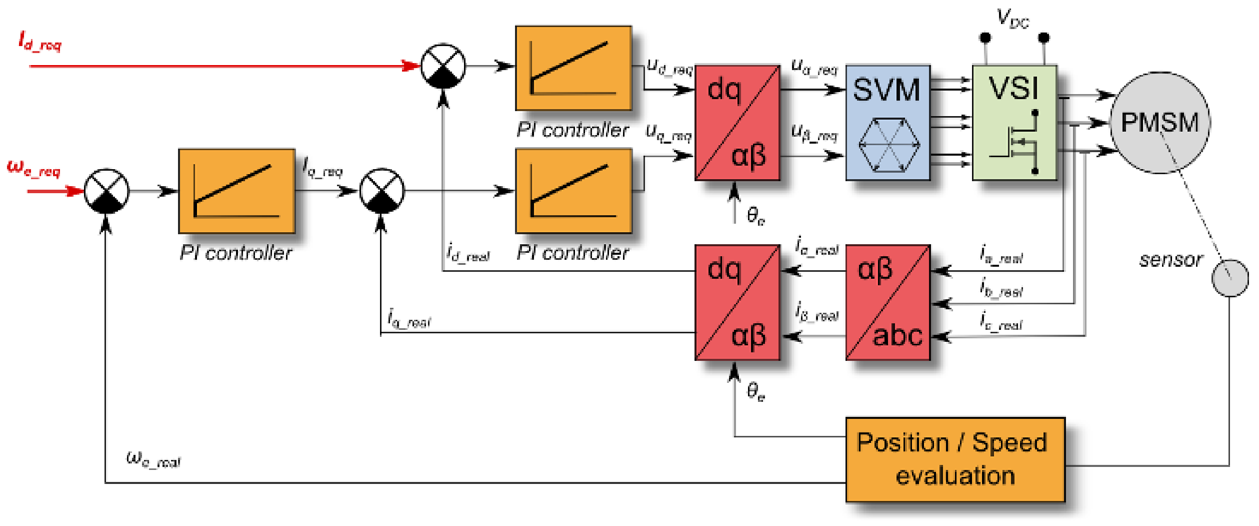


Figure 29. Speed FOC control mode

9.2.2. Application demo control—“App control” tab

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

- Speed gauge—shows the actual and required speeds.
- Required speed slider—sets up the required speed.
- DC-bus voltage—shows the actual DC-bus voltage.
- Current i_q—shows the actual torque-producing current.
- Current limitation—sets up the torque-producing current limit.
- Demo mode on/off button—turns the demonstration mode on/off.
- RUN/STOP PWM button—runs/stops the whole application (sets the PWM on and off).
- Notification—shows the notification about the actual application state (or faults).

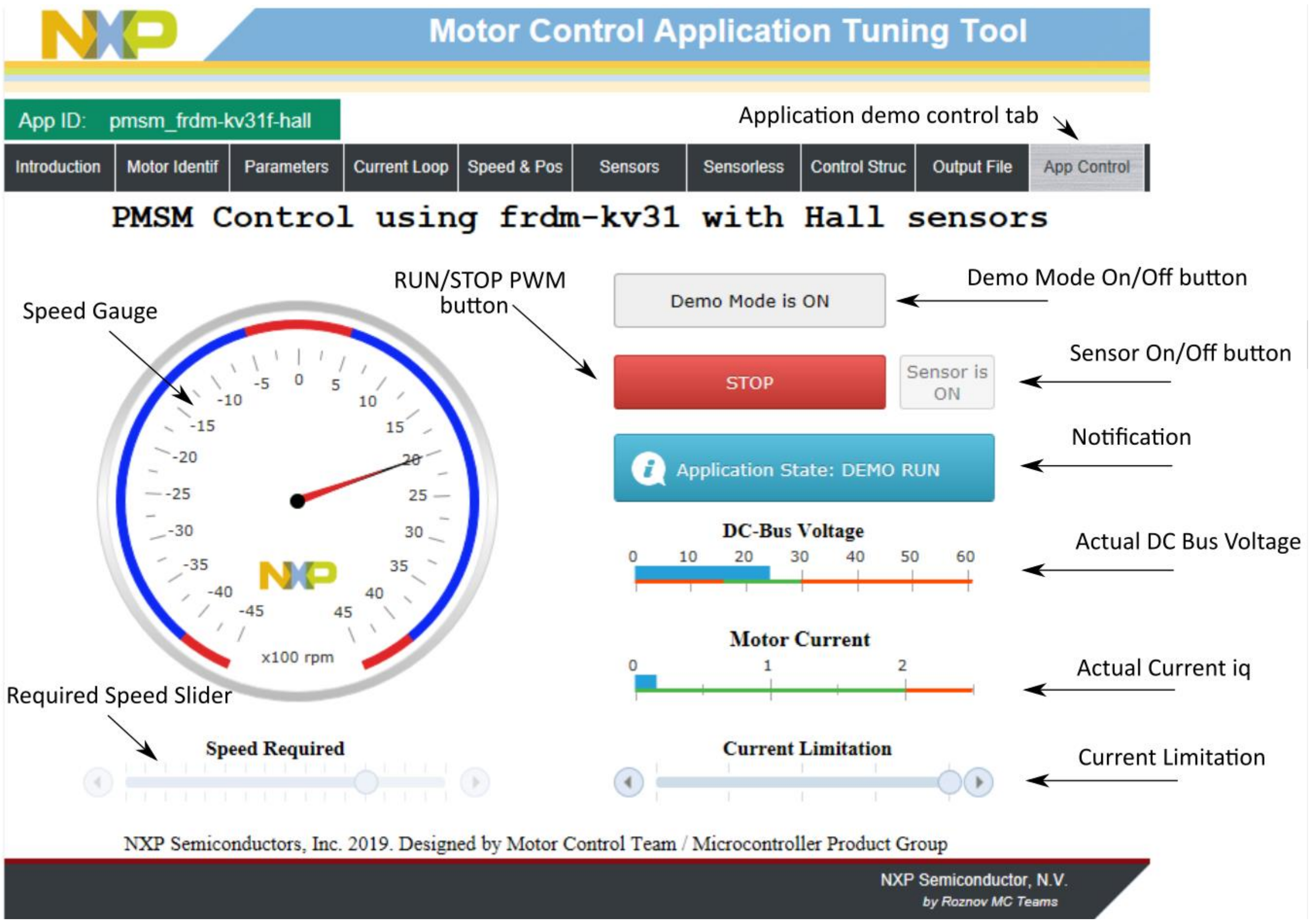


Figure 30. FreeMASTER control page

Here are basic instructions for motor control:

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

9.3. Identifying parameters of your motor using MCAT

This section provides a guide on how to run your own motor or tune the default motor in several steps. It is highly recommended to go through all the steps carefully to eliminate any possible issues during the tuning process. The state diagram in Figure 31 shows a typical PMSM sensor/sensorless control tuning process. Each tuning phase is described in more detail in the following sections.

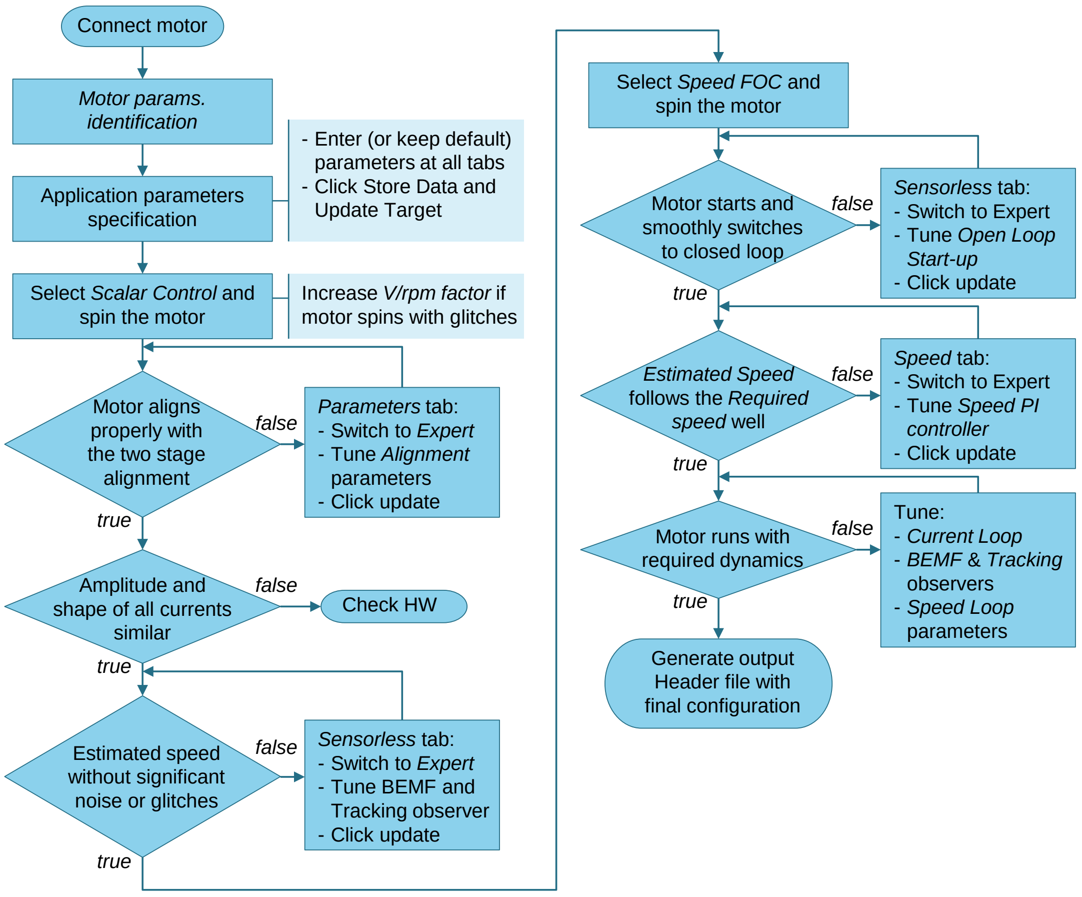

Connect motor
Motor params. identification
Application parameters specification
- Enter (or keep default) parameters at all tabs
- Click Store Data and Update Target
Select Scalar Control and spin the motor
Increase V/rpm factor if motor spins with glitches
Motor aligns properly with the two stage alignment
false
true
Parameters tab:
- Switch to Expert
- Tune Alignment parameters
- Click update
Amplitude and shape of all currents similar
Check HW
Estimated speed without significant noise or glitches
Sensorless tab:
- Switch to Expert
- Tune BEMF and Tracking observer
- Click update
Select Speed FOC and spin the motor
Motor starts and smoothly switches to closed loop
Sensorless tab:
- Switch to Expert
- Tune Open Loop Start-up
- Click update
Estimated Speed follows the Required speed well
Speed tab:
- Switch to Expert
- Tune Speed PI controller
- Click update
Motor runs with required dynamics
Tune:
- Current Loop
- BEMF & Tracking observers
- Speed Loop parameters
Generate output Header file with final configuration

Figure 31. Running a new PMSM

9.3.1. PMSM parameters 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 the stator resistance R_s, direct inductance L_d, quadrature inductance L_q, and BEMF constant K_e. If your connected PMSM motor is not the default Teknic or Linix motor described in Section 2, “Hardware setup”, you have to identify the parameters of your motor first.

9.3.1.1. Power stage characterization

Each inverter introduces the total error voltage U_{error}, which is caused by the dead time, current clamping effect, and transistor voltage drop. The total error voltage U_{error} depends on the phase current i_s and this dependency is measured during the power stage characterization process. An example of the inverter error characteristic is shown in Figure 32. The power stage characterization is a part of the MCAT and can be controlled from the “Motor Identif” tab. To perform the characterization, connect the motor with a known stator resistance R_s and enter this value into the “Calib Rs” field. Then specify the “Calibration Range”, which is the range of the stator current i_s, in which the measurement of U_{error} is performed. Start the characterization by pressing the “Calibrate” button. The characterization gradually performs 65 i_{sd} current steps (from $i_s = -I_{s,calib}$ to $i_s = I_{s,calib}$) with each taking 300 ms, so be aware that the process takes about 20 seconds and the motor must withstand this load. The acquired characterization data is saved to a file and used later for the phase voltage correction during the R_s measurement process. The following R_s measurement can be done with the $I_{s,calib}$ maximum current. It is recommended to use a motor with a low R_s for characterization purposes.

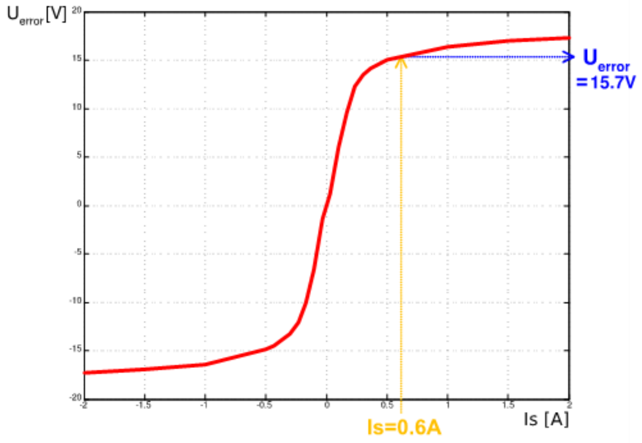


Figure 32. Example power stage characteristic

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

9.3.1.2. Stator resistance measurement

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

$$R_s = \frac{U_{DC} - U_{error}}{I_{phN}} \; [\Omega] \qquad \text{Eq. 4.}$$

9.3.1.3. 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 the DCbus/2 and *F start*, the frequency of the measuring signal is gradually decreased by *F dec* down to *F min* again, until *Id ampl* is reached. If *Id ampl* is still not reached, the measurement continues with DCbus/2 and *F min*. The tuning process is shown in Figure 33.

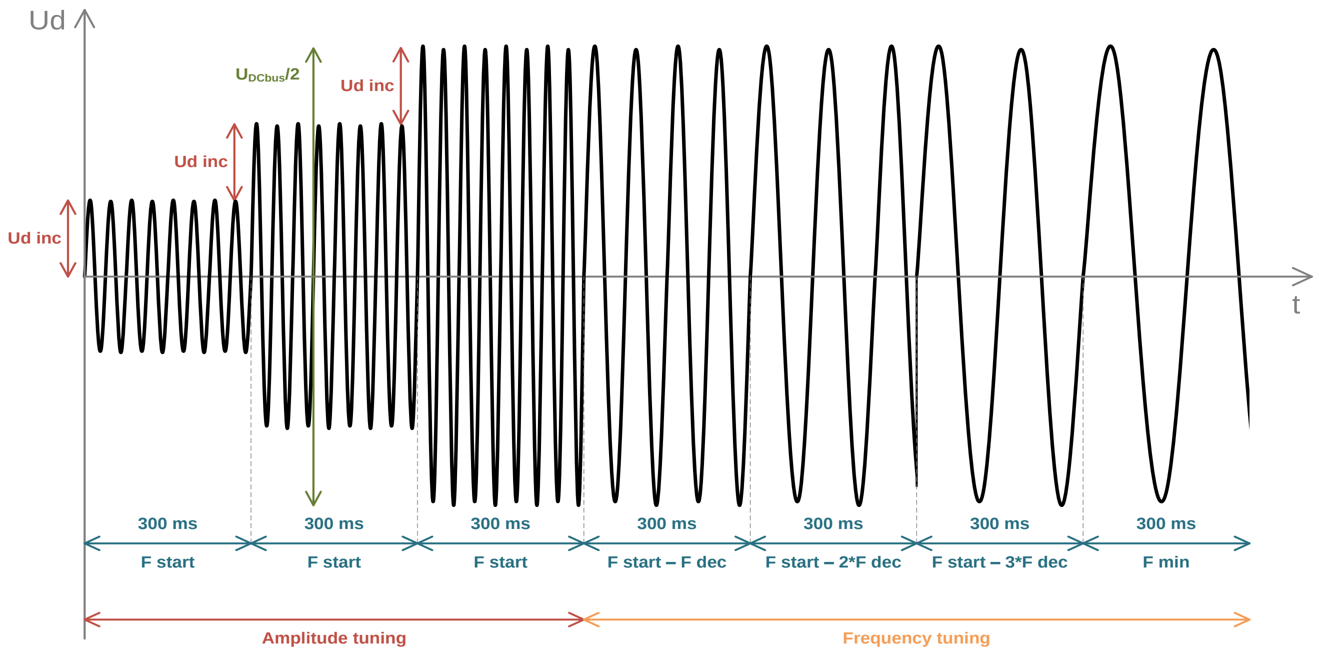


Figure 33. Tuning Ls measuring signal

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

$$Z_{RL} = \frac{U_d}{I_{d\,ampl}} \ [\Omega] \qquad \textit{Eq. 5.}$$

$$X_{Ls} = \sqrt{Z_{RL}^2 - R_S^2} \ [\Omega] \qquad \textit{Eq. 6.}$$

$$L_s = \frac{X_{Ls}}{2\pi f} \ [\Omega] \qquad \textit{Eq. 7.}$$

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

9.3.1.4. 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 spin. I_d is controlled through $I_{d\,meas}$ and the electrical open-loop position is generated by integrating the required speed, which is derived from N_{nom}. When the motor reaches the required speed, the BEMF voltages obtained by the BEMF observer are filtered and K_e is calculated:

$$\boldsymbol{K_{\mathrm{e}}} = \frac{\boldsymbol{U_{\mathrm{BEMF}}}}{\boldsymbol{\omega_{\mathrm{el}}}}\ [\boldsymbol{\Omega}] \qquad \textit{Eq. 8.}$$

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

- Ensure that the number of *pp* is correct. The required speed for the K_e measurement is also calculated from *pp*. Therefore, inaccuracy in *pp* causes inaccuracy in the resulting K_e.
- Increase $I_{d\ meas}$ to produce higher torque when spinning during the open loop.
- Decrease N_{nom} to decrease the required speed for the K_e measurement.

9.3.1.5. 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 the *pp* assistant performs one electrical revolution, stops for a few seconds, and then repeats it. Because the *pp* value is the ratio between the electrical and mechanical speeds, it can be determined as the number of stops per one mechanical revolution. It is recommended not to count the stops during the first mechanical revolution because the alignment occurs during the first revolution and affects the number of stops. During the *pp* measurement, the current loop is enabled and the I_d current is controlled to $I_{d\ meas}$. The electrical position is generated by integrating the open-loop speed. If the rotor does not move after the start of the number of *pp* assistant, stop the assistant, increase $I_{d\ meas}$, and restart the assistant.

9.3.1.6. Mechanical parameters measurement

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

$$\frac{\mathbf{d}\boldsymbol{\omega_{\mathrm{m}}}}{\mathbf{dt}} = \frac{\mathbf{1}}{\boldsymbol{J}}(\boldsymbol{T} - \boldsymbol{T_{load}} - \boldsymbol{B}\boldsymbol{\omega_{\mathrm{m}}})\ [\mathbf{rad}/\boldsymbol{s}^{\mathbf{2}}] \qquad \textit{Eq. 9.}$$

The ω_m character in the equation is the mechanical speed. The mechanical parameter identification software uses the torque profile, as shown in Figure 34. The loading torque is (for simplicity reasons) said to be 0 during the whole measurement. Only the friction and the motor-generated torque are considered. During the first phase of measurement, the constant torque T_{meas} is applied and the motor accelerates to 50 % of its nominal speed in time t_1. These integrals are calculated during the period from t_0 (the speed estimation is accurate enough) to t_1:

$$\boldsymbol{T_{int}} = \int_{t_0}^{t_1} \boldsymbol{T}\,\boldsymbol{dt}\ [\boldsymbol{Nms}] \qquad \textit{Eq. 10.}$$

$$\boldsymbol{\omega_{\mathrm{int}}} = \int_{t_0}^{t_1} \boldsymbol{\omega_{\mathrm{m}}}\,\boldsymbol{dt}\ [\boldsymbol{rad/s}] \qquad \textit{Eq. 11.}$$

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

The final mechanical parameter estimation can be calculated by integrating:

$$\boldsymbol{\omega}_{\mathbf{m}}(\boldsymbol{t}_1) = \frac{\mathbf{1}}{\boldsymbol{J}} \boldsymbol{T}_{int} - \boldsymbol{B}\boldsymbol{\omega}_{\mathbf{int}} + \boldsymbol{\omega}_{\mathbf{m}}(\boldsymbol{t}_0)\ [\boldsymbol{rad/s}] \qquad \textit{Eq. 12.}$$

The moment of inertia is:

$$\boldsymbol{J} = \frac{\boldsymbol{\tau}_{\boldsymbol{m}} \boldsymbol{T}_{int}}{\boldsymbol{\tau}_{\boldsymbol{m}}[\boldsymbol{\omega}_{\mathbf{m}}(\boldsymbol{t}_1) - \boldsymbol{\omega}_{\mathbf{m}}(\boldsymbol{t}_0)] + \boldsymbol{\omega}_{\mathbf{int}}}\ [\boldsymbol{kgm}^{\mathbf{2}}] \qquad \textit{Eq. 13.}$$

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

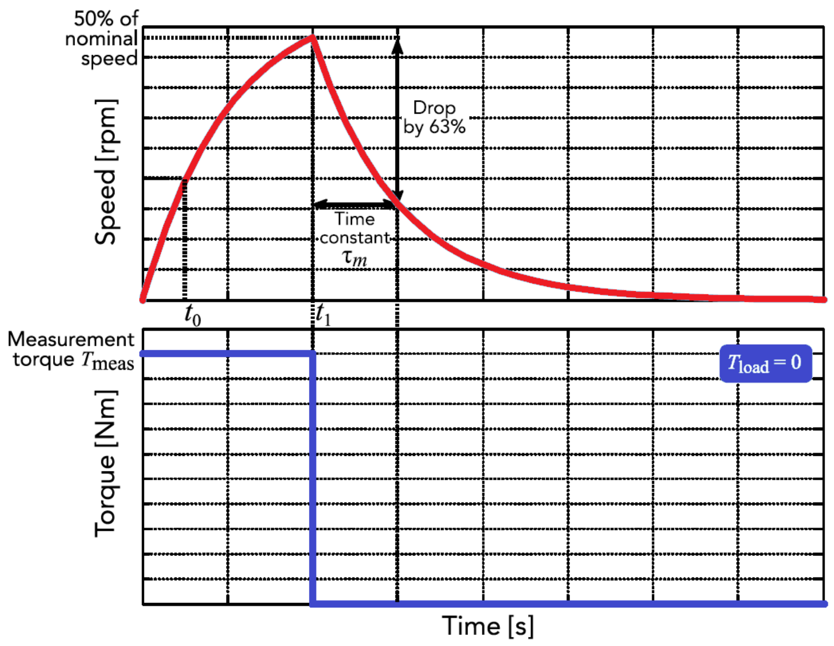


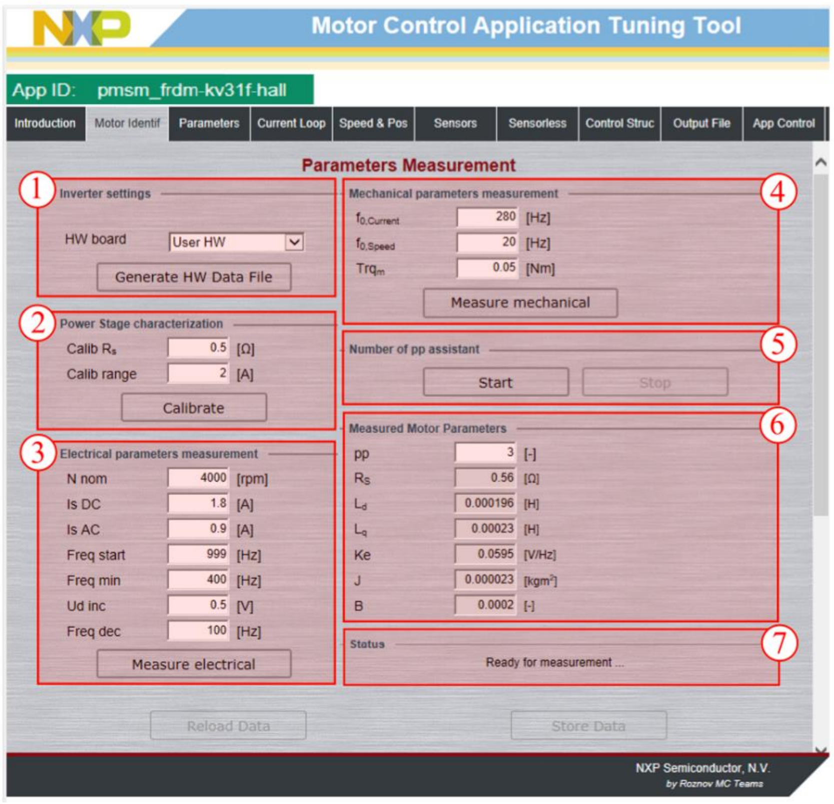
Figure 34. PMSM identification tab

9.3.2. PMSM electrical and mechanical parameters measurement process

The motor identification process can be controlled and set up in the MCAT “Motor Identif” tab, which is shown in Figure 35. To measure your own motor, follow these steps:

- Select your hardware board. Choose between the standard NXP hardware or use your own. If you use your own hardware, specify its scales (“I max” and “U DCB max” in the “Parameters” menu tab).
- If you don’t know the number of motor’s pole-pairs, use the number of pole-pair assistant described in Section 9.3.1.5, “Number of pole-pair assistant”.

- If you use your own hardware for the first time, perform the power stage characterization described in Section 9.3.1.1, “Power stage characterization”.
- Enter the motor measurement parameters and start the measurement by pressing the “Measure electrical” or “Measure mechanical” buttons. You can observe which parameter is being measured in the “Status” bar.



App ID: pmsm_frdm-kv31f-hall

Parameters Measurement

1 Inverter settings

HW board	User HW	

2 Power Stage characterization

Calib R_s	0.5	[Ω]
Calib range	2	[A]

3 Electrical parameters measurement

N nom	4000	[rpm]
Is DC	1.8	[A]
Is AC	0.9	[A]
Freq start	999	[Hz]
Freq min	400	[Hz]
Ud inc	0.5	[V]
Freq dec	100	[Hz]

4 Mechanical parameters measurement

$f_{0,Current}$	280	[Hz]
$f_{0,Speed}$	20	[Hz]
Trq_m	0.05	[Nm]

5 Number of pp assistant

6 Measured Motor Parameters

pp	3	[-]
R_s	0.56	[Ω]
L_d	0.000196	[H]
L_q	0.00023	[H]
Ke	0.0595	[V/Hz]
J	0.000023	[kgm^2]
B	0.0002	[-]

7 Status

Ready for measurement ...

1 - HW selection
2 - Power stage characterization
3 - Electrical measurement conditions
4 - Mechanical measurement conditions
5 - Start/stop pp assistant
6 - Measured parameters
7 - Measurement status

Figure 35. PMSM identification tab

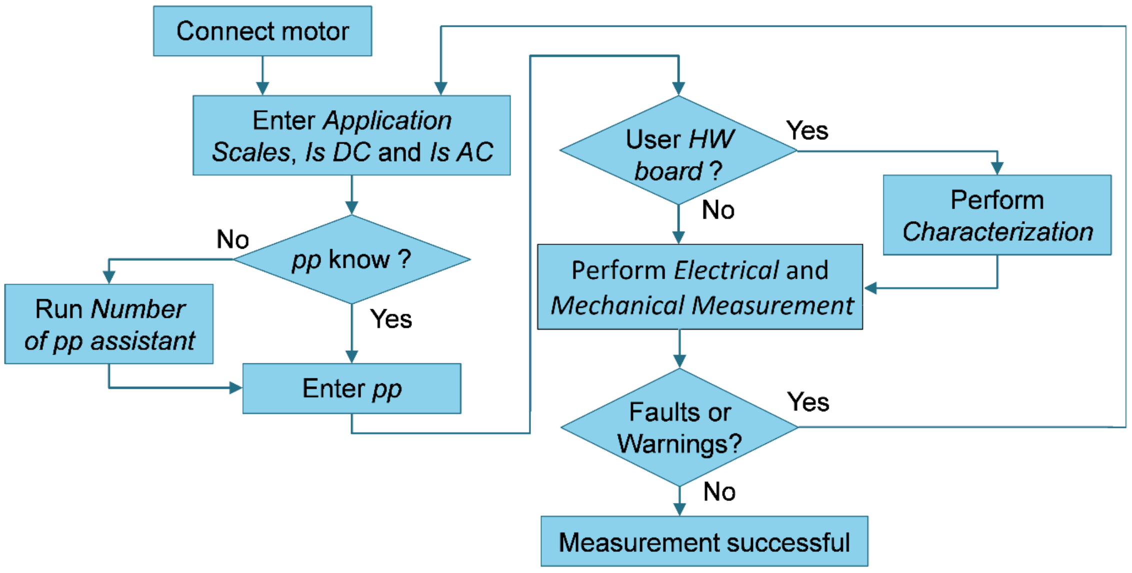


Figure 36. Measurement process diagram

During the measurement, measurement faults and warnings may occur. Do not confuse these faults for the application faults, such as overcurrent, undervoltage, and so on. The list of these faults with their description and possible troubleshooting is shown in Table 5.

Table 5. Measurement faults and warnings

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

9.3.3. Initial configuration setting and update

1. Open the PMSM control application FreeMASTER project containing the dedicated MCAT plug-in module.
2. Select the “Parameters” tab.
3. Leave the measured motor parameters or specify the parameters manually. The motor parameters can be obtained from the motor data sheet or using the PMSM parameters measurement procedure described in *PMSM Electrical Parameters Measurement* (document AN4680). All parameters provided in Table 6 are accessible. The motor inertia J expresses the overall system inertia and can be obtained using a mechanical measurement. The J parameter is used to calculate the speed controller constant. However, the manual controller tuning can also be used to calculate this constant.

Table 6. MCAT motor parameters

Parameter	Units	Description	Typical range
pp	[-]	Motor pole pairs	1 – 10
Rs	[Ω]	1-phase stator resistance	0.3 – 50
Ld	[H]	1-phase direct inductance	0.00001 – 0.1
Lq	[H]	1-phase quadrature inductance	0.00001 – 0.1
Ke	[V.sec/rad]	BEMF constant	0.001 – 1
J	[kg.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

4. Set the hardware scales—the modification of these two fields is not required when a reference to the standard power stage board is used. These scales express the maximum measurable current and voltage analog quantities.
5. Check the fault limits—these fields are not accessible in the “Basic” mode and are calculated

using the motor parameters and hardware scales (see Table 7).

Table 7. Fault limits

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

6. Check the application scales—these fields are not accessible in the “Basic” mode and are calculated using the motor parameters and hardware scales.

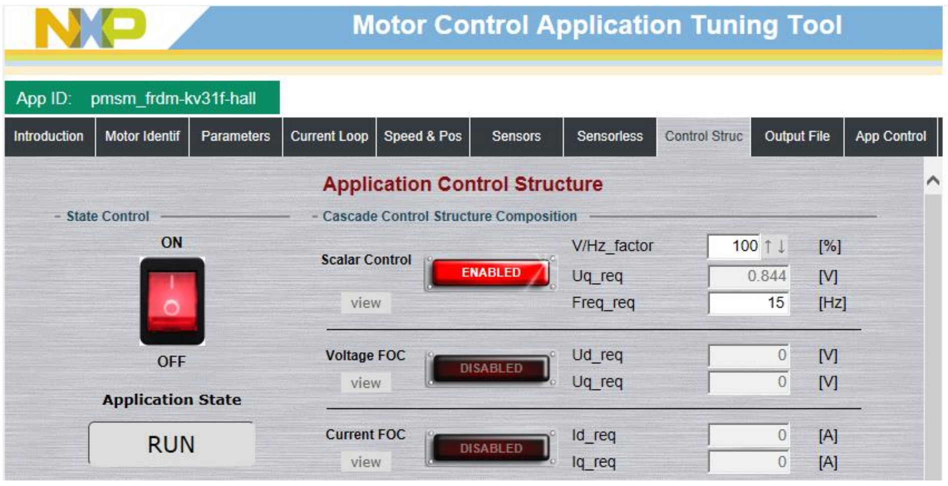
Table 8. Application scales

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

7. Check the alignment parameters—these fields are not accessible in the “Basic” mode and are calculated using the motor parameters and hardware scales. The parameters express the required voltage value applied to the motor during the rotor alignment and its duration.
8. Click the “Store Data” button to save the modified parameters into the inner file.

9.3.4. Control structure modes

1. Select the scalar control by clicking the “DISABLED” button in the “Scalar Control” section. The button color changes to red and the text changes to “ENABLED”.
2. Turn the application switch on. The application state changes to “RUN”.
3. Set the required frequency value in the “Freq_req” field; for example, 15 Hz in the “Scalar Control” section. The motor starts running (Figure 37).



V/Hz_factor	100	[%]
Uq_req	0.844	[V]
Freq_req	15	[Hz]
Ud_req	0	[V]
Uq_req	0	[V]
Id_req	0	[A]
Iq_req	0	[A]

Figure 37. MCAT scalar control

4. Select the “Phase Currents” recorder from the “Scalar & Voltage Control” FreeMASTER project tree.

5. The optimal ratio for the V/Hz profile can be found by changing the V/Hz factor directly or using the “UP/DOWN” buttons. The shape of the motor currents should be close to a sinusoidal shape (Figure 38).

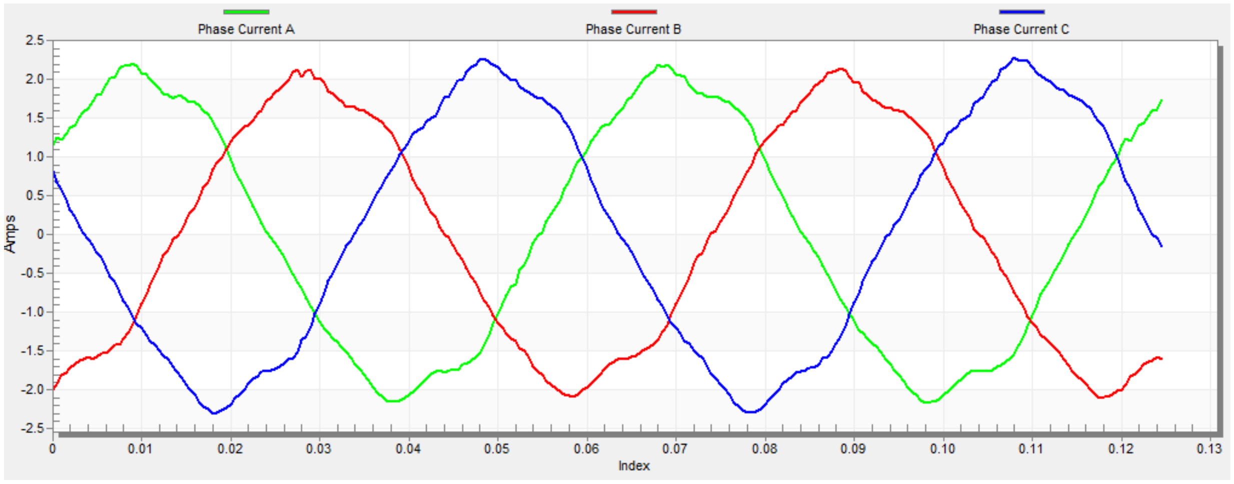


Figure 38. Phase currents

6. Select the “Position” recorder to check the observer functionality. The difference between the “Position Electrical Scalar” and the “Position Estimated” should be minimal (see Figure 39) 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.

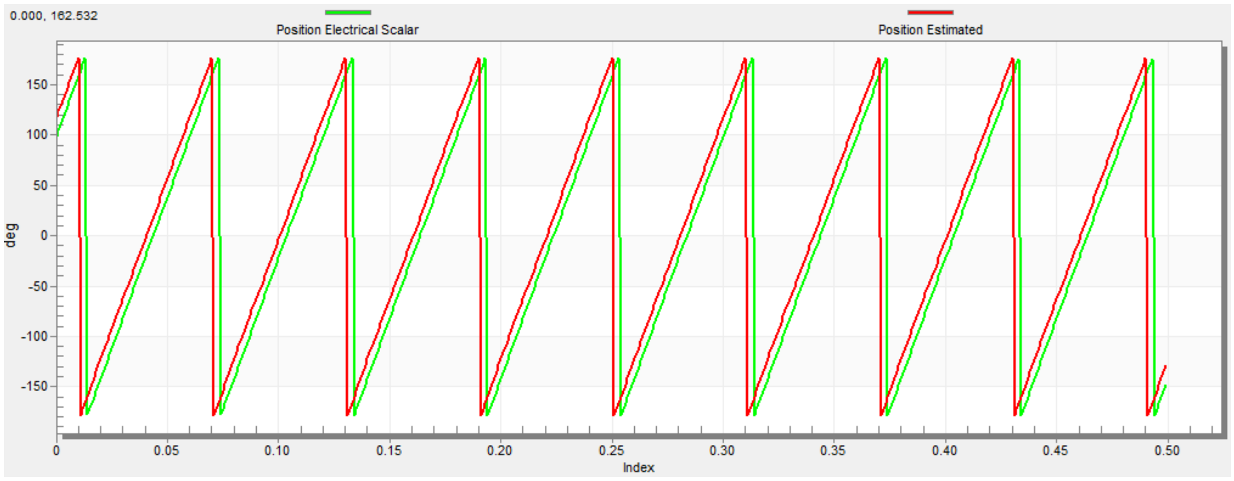


Figure 39. Generated and estimated positions

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

9.3.5. Encoder sensor setting

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

1. Select the scalar control by clicking the DISABLED button in the “Scalar Control” section of the “Control Struct” tab. The button color changes to red, and the text to “ENABLED”.
2. Turn the application switch on. The application state changes to “RUN”.
3. Set the required frequency value in the “Freq_req” field; for example, 15 Hz in the “Scalar control” section. The motor starts running.
4. Check the encoder direction. Select the “Encoder Direction Scope” from the “Scalar & Voltage Control” project tree. If the encoder direction is right, the estimated speed is equal to the measured mechanical speed. If the measured mechanical speed is opposite to the estimated speed, the direction must be changed by inverting the encoder wires—phase A and phase B (or the other way round).

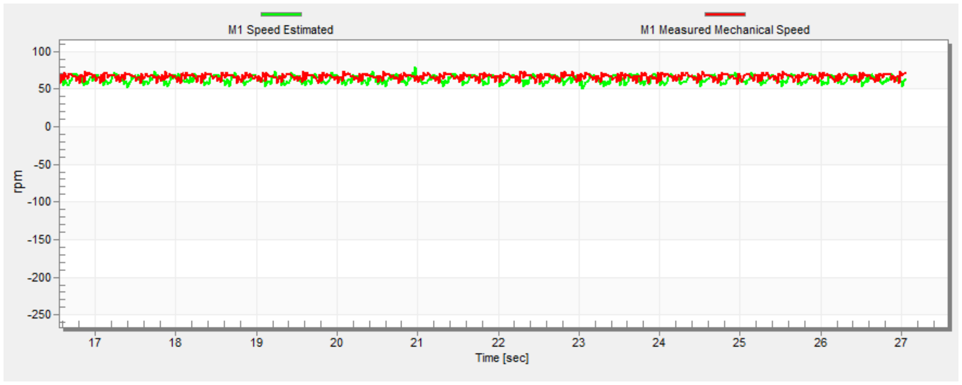


Figure 40. Encoder direction—right direction

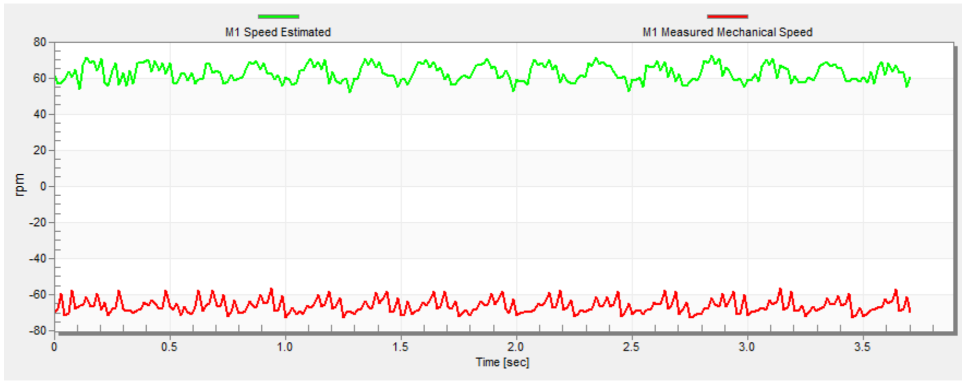


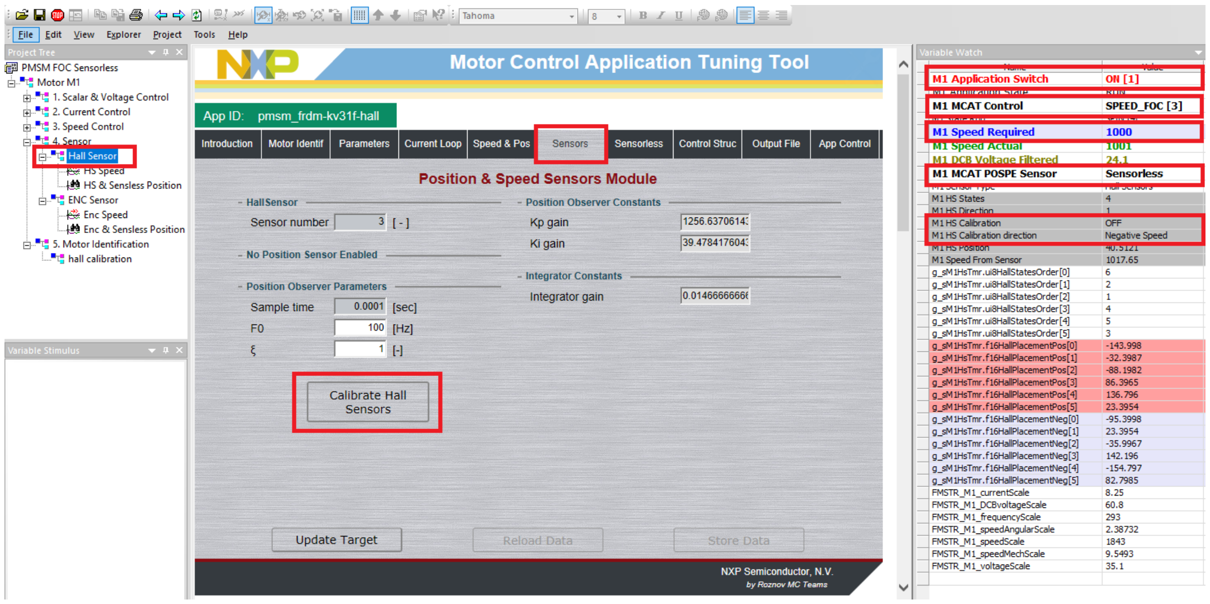
Figure 41. Encoder direction—wrong direction

9.3.6. Hall sensors setting

The hall sensors settings are in the “Sensors” tab. The hall sensors enable you to compute the speed and position for the sensor speed control. For proper sensing, it is necessary to calibrate the position of the hall sensors. There are two options of hall sensor calibration.

Automatic hall sensors calibration:

- For automatic calibration, click the “Calibrate Hall Sensors” button in the MCAT “Sensors” tab. The calibration is performed in one minute. In scalar control, the position of hall sensors calibrates the positive and negative speeds.



App ID: pmsm_frdm-kv31f-hall

Position & Speed Sensors Module

HallSensor

Sensor number 3 [-]

No Position Sensor Enabled

Position Observer Parameters

Sample time 0.0001 [sec]

F0 100 [Hz]

ξ 1 [-]

Position Observer Constants

Kp gain 1256.6370614

Ki gain 39.478417604

Integrator Constants

Integrator gain 0.014666666

Variable Watch

Name	Value
M1 Application Switch	ON [1]
M1 Application State	RUN
M1 MCAT Control	SPEED_FOC [3]
[illegible]	[illegible]
M1 Speed Required	1000
M1 Speed Actual	1001
M1 DCB Voltage Filtered	24.1
M1 MCAT POSPE Sensor	Sensorless
M1 Sensor Type	Hall Sensors
M1 HS States	4
M1 HS Direction	1
M1 HS Calibration	OFF
M1 HS Calibration direction	Negative Speed
M1 HS Position	40.5121
M1 Speed From Sensor	1017.65
g_sM1HsTmr.ui8HallStatesOrder[0]	6
g_sM1HsTmr.ui8HallStatesOrder[1]	2
g_sM1HsTmr.ui8HallStatesOrder[2]	1
g_sM1HsTmr.ui8HallStatesOrder[3]	4
g_sM1HsTmr.ui8HallStatesOrder[4]	5
g_sM1HsTmr.ui8HallStatesOrder[5]	3
g_sM1HsTmr.f16HallPlacementPos[0]	-143.998
g_sM1HsTmr.f16HallPlacementPos[1]	-32.3987
g_sM1HsTmr.f16HallPlacementPos[2]	-88.1982
g_sM1HsTmr.f16HallPlacementPos[3]	86.3965
g_sM1HsTmr.f16HallPlacementPos[4]	136.796
g_sM1HsTmr.f16HallPlacementPos[5]	23.3954
g_sM1HsTmr.f16HallPlacementNeg[0]	-95.3998
g_sM1HsTmr.f16HallPlacementNeg[1]	23.3954
g_sM1HsTmr.f16HallPlacementNeg[2]	-35.9967
g_sM1HsTmr.f16HallPlacementNeg[3]	142.196
g_sM1HsTmr.f16HallPlacementNeg[4]	-154.797
g_sM1HsTmr.f16HallPlacementNeg[5]	82.7985
FMSTR_M1_currentScale	8.25
FMSTR_M1_DCBvoltageScale	60.8
FMSTR_M1_frequencyScale	293
FMSTR_M1_speedAngularScale	2.38732
FMSTR_M1_speedScale	1843
FMSTR_M1_speedMechScale	9.5493
FMSTR_M1_voltageScale	35.1

Figure 42. Hall sensors calibration

Manual hall sensors calibration:

- Open the *Sensor/Hall Sensor/* folder in the FreeMASTER project tree.
- Run the motor in the sensorless speed control. In the FreeMASTER Variable Watch, write number 3 to the “M1 MCAT Control” variable (SPEED FOC) and write 0 (Sensorless) to the “M1 MCAT POSPE Sensor” variable. Then write a positive speed to “M1 Speed Required” (for example, 1000 rpm). Turn the application on by writing 1 to “M1 Application Switch”. The motor’s rotor starts spinning.
- Set the positive speed direction of calibration—write 1 to “M1 HS Calibration direction”.
- Turn the hall sensor calibration on by writing 1 to “M1 HS Calibration” and wait at least 30 seconds.
- Turn the calibration off by writing 0 to “M1 HS Calibration” and run the motor with negative speed by writing -1000 to “M1 Speed Required” (1000 rpm).
- Set the negative speed direction of calibration—write 0 to “M1 HS Calibration direction”.
- Turn the hall sensor calibration on by writing 1 to “M1 HS Calibration” and wait at least 30 seconds.
- Turn the calibration off. The hall sensors calibration is completed.

NOTE

The application support now only three hall sensors.

9.3.7. Alignment tuning

For the alignment parameters, navigate to the “Tab” menu and select “Parameters”. The alignment procedure sets the rotor to an accurate initial position and enables you to apply full start-up torque to the motor. The rotor-alignment parameters are available for editing in the “Expert” mode. A correct initial position is needed mainly for high start-up loads (compressors, washers, and so on). The aim of the alignment is to have the rotor in a stable position, without any oscillations before the startup.

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

9.3.8. 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.

1. Switch the tuning mode to “Expert”.
2. Lock the motor shaft.
3. 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.
4. Select the “Current Controller Id” recorder.
5. Select the “Control Structure” tab, switch to “Current FOC”, set the “Iq_req” field to a very low value (for example 0.01), and set the required step in “Id_req”. The control loop response is shown in the recorder (see Figure 32).
6. Tune the loop bandwidth and attenuation until you achieve the required response. The example waveforms show the correct and incorrect settings of the current loop parameters:
   - The loop bandwidth is low (110 Hz) and the settling time of the I_d current is long (Figure 43).

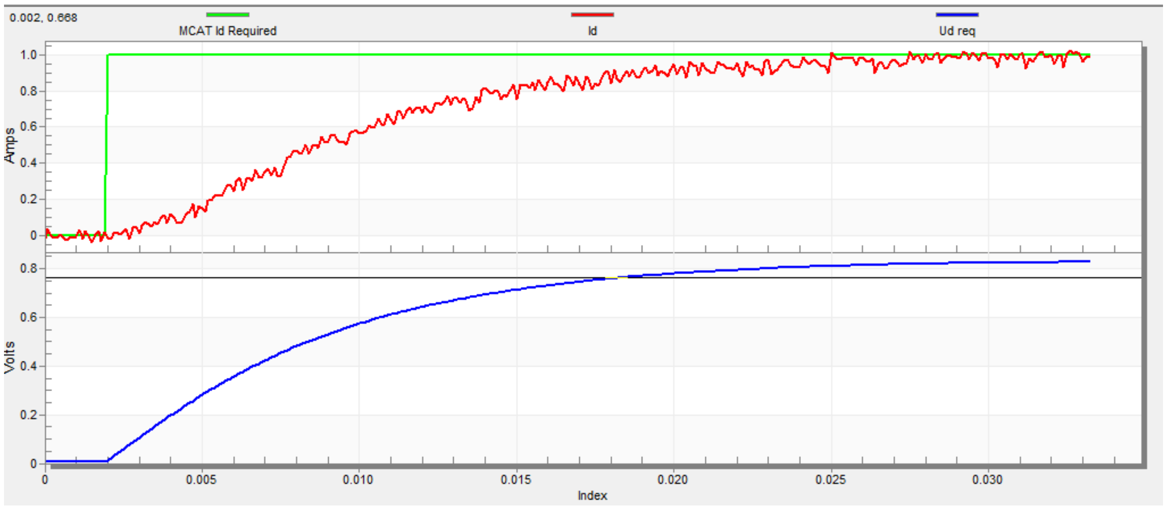


Figure 43. Slow step response of the Id current controller

– The loop bandwidth (400 Hz) is optimal and the response time of the I_d current is sufficient (see Figure 44).

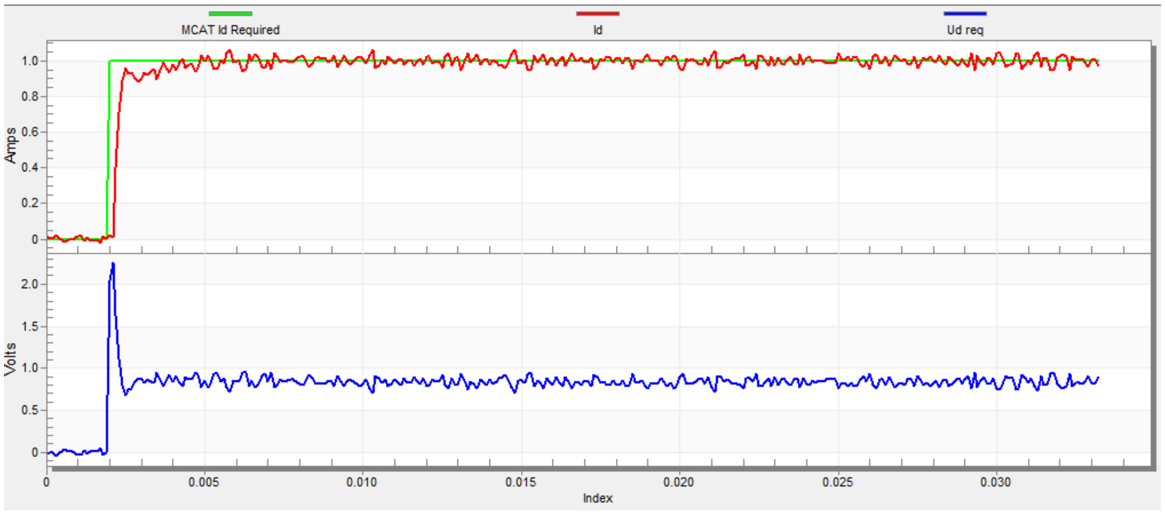


Figure 44. Optimal step response of the Id current controller

– The loop bandwidth is high (700 Hz) and the response time of the I_d current is very fast, but with oscillation and overshoot (see Figure 45).

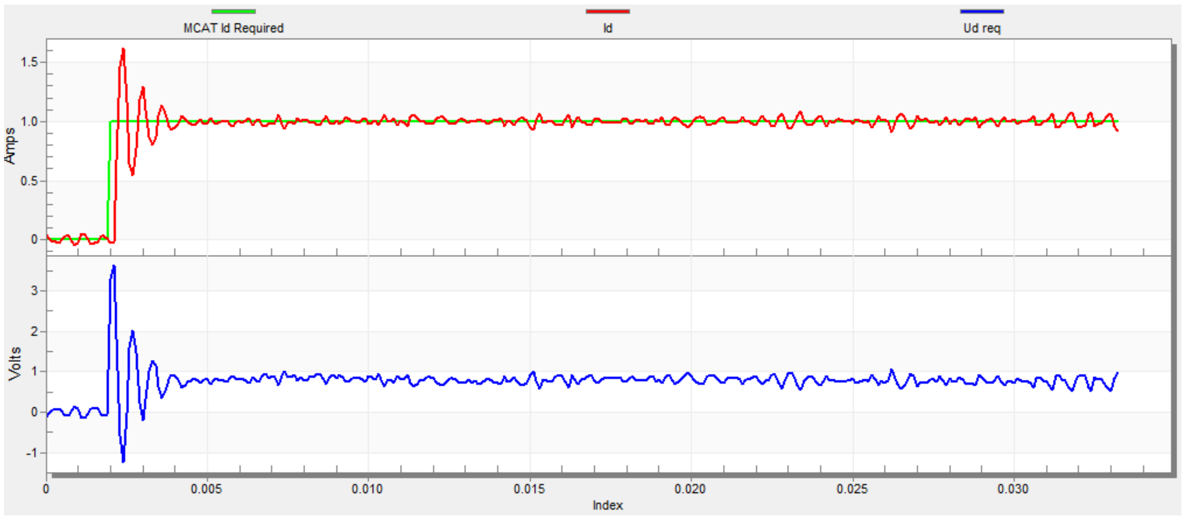


Figure 45. Fast step response of the Id current controller

9.3.9. Speed ramp tuning

1. The speed command is applied to the speed controller through a speed ramp. The ramp function contains two increments (up and down) which express the motor acceleration and deceleration per second. If the increments are very high, they can cause an overcurrent fault during acceleration and an overvoltage fault during deceleration. In the “Speed” scope, you can see whether the “Speed Actual Filtered” waveform shape equals the “Speed Ramp” profile.
2. The increments are common for the scalar and speed control. The increment fields are in the “Speed & Pos” tab and accessible in both tuning modes. Clicking the “Update Target” button applies the changes to the MCU. An example speed profile is shown in Figure 46. The ramp increment down is set to 500 rpm/sec and the increment up is set to 3000 rpm/sec.
3. The start-up ramp increment is in the “Sensorless” tab and its value is usually higher than that for the speed loop ramp.

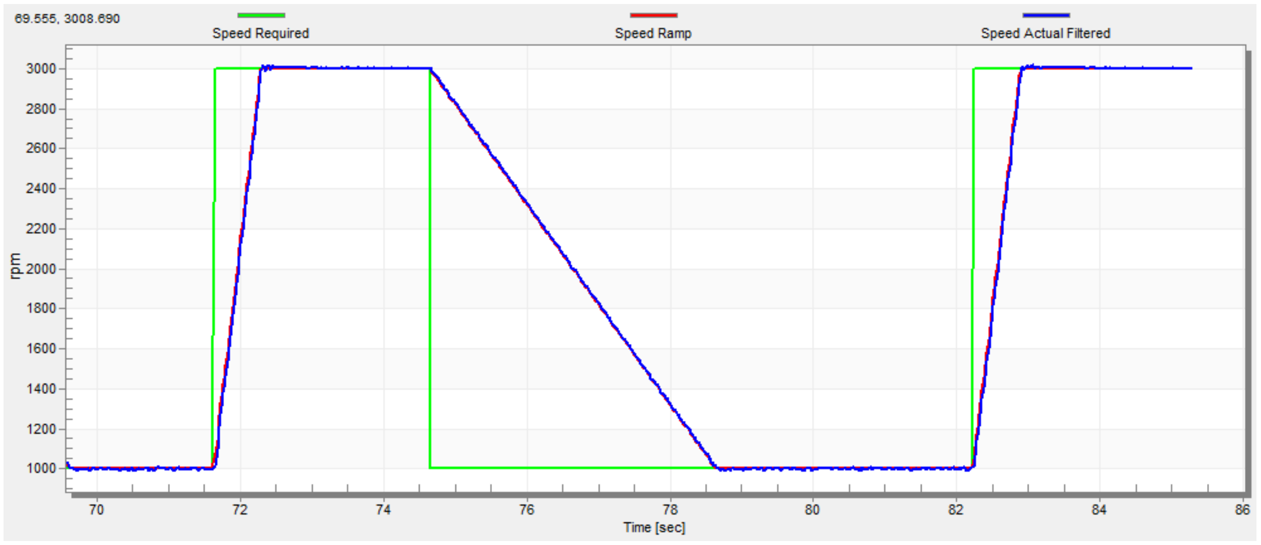


Figure 46. Speed profile

9.3.10. Open loop startup

1. The start-up process can be tuned by a set of parameters located in the “Sensorless” tab. Two of them (ramp increment and current) are accessible in both tuning modes. The start-up tuning can be processed in all control modes besides the scalar control. Setting the optimal values results in a proper motor startup. An example start-up state of low-dynamic drives (fans, pumps) is shown in Figure 47.
2. Select the “Startup” recorder from the FreeMASTER project tree.
3. Set the start-up ramp increment typically to a higher value than the speed-loop ramp increment.
4. Set the start-up current according to the required start-up torque. For drives such as fans or pumps, the start-up torque is not very high and can be set to 15 % of the nominal current.
5. Set the required merging speed—when the open-loop and estimated position merging starts, the threshold is mostly set in the range of 5 % ~ 10 % of the nominal speed.
6. Set the merging coefficient—in the position merging process duration, 100 % corresponds to a half of an electrical revolution. The higher the value, the faster the merge is done. The values close to 1 % are set for the drives where a high start-up torque and smooth transitions between the open loop and the closed loop are required.
7. Click the “Update Target” button to apply the changes to the MCU.
8. Switch to the “Control Structure” tab, and enable the “Speed FOC”.
9. Set the required speed higher than the merging speed.
10. Check the start-up response in the recorder.
11. Tune the start-up parameters until you achieve an optimal response.
12. If the rotor does not start running, increase the start-up current.
13. If the merging process fails (the rotor is stuck or stopped), decrease the start-up ramp increment, increase the merging speed, and set the merging coefficient to 5 %.

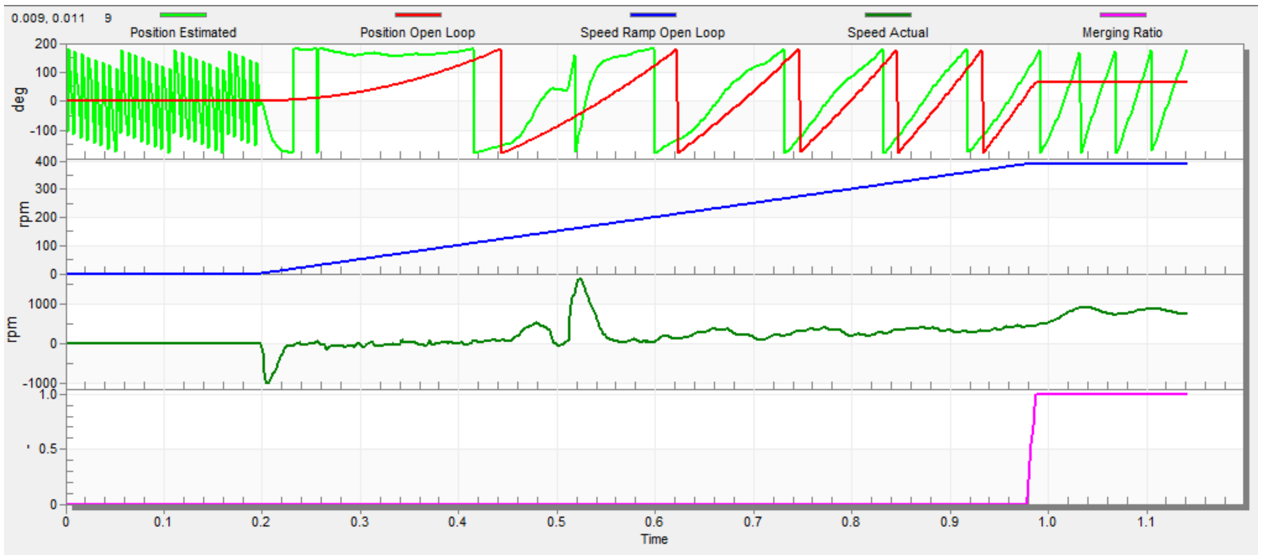


Figure 47. Motor startup

9.3.11. BEMF observer tuning

1. In the “Basic” mode, the parameters of the BEMF observer and the tracking observer are fully calculated using the motor parameters and no action is required. If the calculated loop parameters do not correspond to the optimal response, the bandwidth and attenuation parameters can be tuned.
2. Switch the tuning mode to “Expert”.
3. Select the “Observer” recorder from the FreeMASTER project tree.
4. Set the required bandwidth and attenuation of the BEMF observer—the bandwidth is typically set to a value close to the current loop bandwidth.
5. Set the required bandwidth and attenuation of the tracking observer—the bandwidth is typically set in the range of 10 – 20 Hz for most low-dynamic drives (fans, pumps).
6. Click the “Update Target” button to apply the changes to the MCU.
7. Check the observer response in the recorder.

9.3.12. 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. Obtaining these mechanical constants using the PMSM electrical and mechanical parameter measurement process is described in Section 9.3.1.6, “Mechanical parameters measurement” and Section 9.3.2, “PMSM electrical and mechanical parameters measurement process”. If these mechanical constants are available, the PI controller constants can be tuned using the loop bandwidth and attenuation. Otherwise, the 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 48). There are dozens of approaches available to tune the PI controller constants. The following steps provide one of these examples to set and tune the speed PI controller for a PM synchronous motor:

1. Select the “Speed Controller” option from the FreeMASTER project tree.
2. Select the “Speed & Pos” tab.
3. Check the “Manual Constant Tuning” option—that is, the “Bandwidth” and “Attenuation” fields are disabled and the “SL_Kp” and “SL_Ki” fields are enabled.
4. Tune the proportional gain:
   - Set the “SL_Ki” integral gain to 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 (about 30 % of the nominal speed).
   - Set a step in the required speed to 40 % of N_{nom}.
   - Switch back to the “Speed loop” tab.
   - Adjust the proportional gain “SL_Kp” until the system responds to the required value properly and without any oscillations or excessive overshoot:
   - If the “SL_Kp” field is set low, then the system response is slow.

- If the “SL_Kp” field is set high, then the system response is tighter.
- When the “SL_Ki” field equals 0, then the system most probably does not achieve the required speed.
- Click the “Update Target” button to apply the changes to the MCU.

5. Tune the integral gain:
   - Increase the “SL_Ki” field slowly to minimize the difference between the required and actual speeds to 0.
   - Adjust the “SL_Ki” field such that you do not see any oscillation or large overshoot of the actual speed value while the required speed step is applied.
   - Click the “Update Target” button to apply the changes to the MCU.
6. Tune the loop bandwidth and attenuation until the required response is received. The example waveforms with the correct and incorrect settings of the speed loop parameters are shown in the following figures:
   - The “SL_Ki” value is low, and the “Speed Actual Filtered” does not achieve the “Speed Ramp” (see Figure 48).

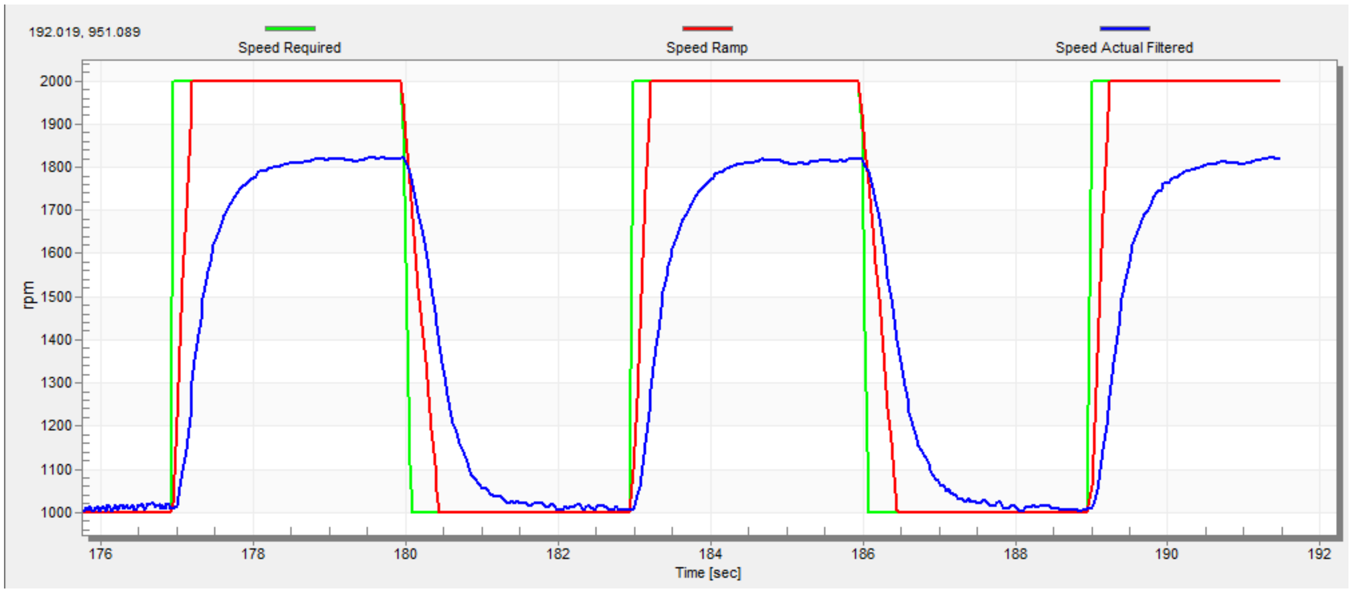


Figure 48. Speed Controller Response—SL_Ki value is low, Speed Ramp is not achieved

- The “SL_Kp” value is low, the “Speed Actual Filtered” greatly overshoots, and the long settling time is unwanted (see Figure 49).

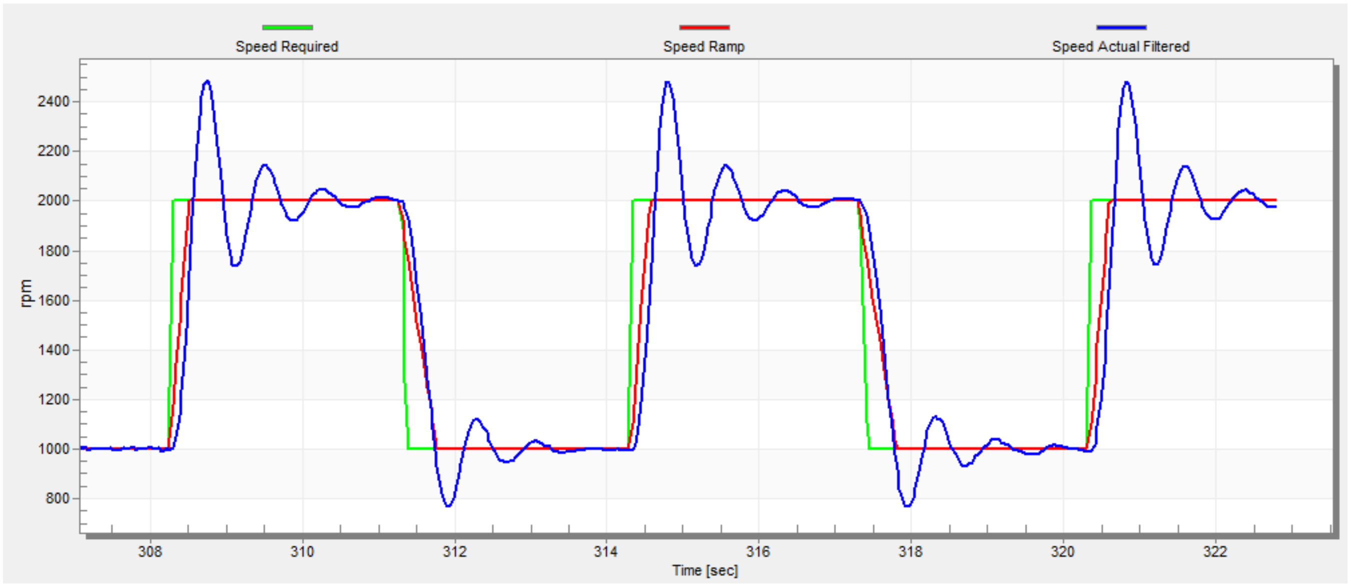


Figure 49. Speed Controller Response—SL_Kp value is low, Speed Actual Filtered greatly overshoots

- The speed loop response has a small overshoot and the "Speed Actual Filtered" settling time is sufficient. Such response can be considered optimal (see Figure 50).

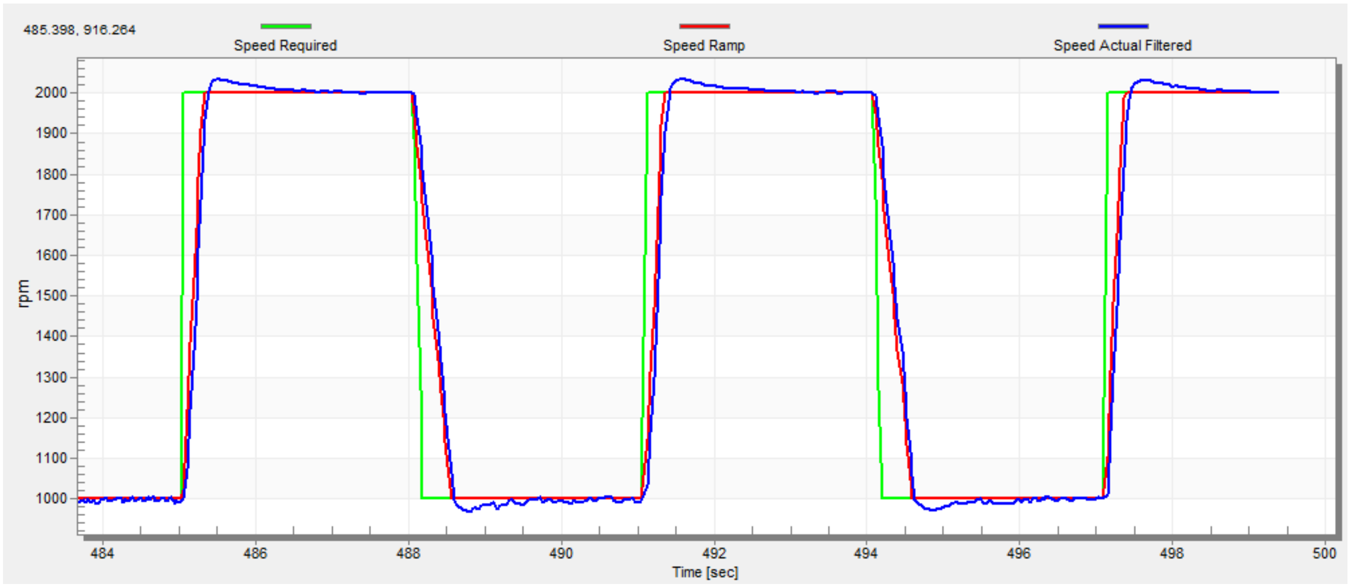


Figure 50. Speed Controller Response—speed loop response with a small overshoot

9.3.13. MCAT output file generation

When you successfully finish tuning the application and want to store all calculated parameters to an embedded application, navigate to the "Output File" tab. The list of all definitions generated by the MCAT can be viewed there. Clicking the "Generate Configuration File" button overwrites the old version of the *m1_pmsm_appconfig.h* file, which contains these definitions.

10. Conclusion

This document describes the implementation of the sensor and sensorless FOC of a 3-phase PMSM on the NXP FRDM-KV31F board with the FRDM-MC-LVPMSM NXP Freedom development platform. The hardware setup used is described in Section 2, “Hardware setup”. In the application, it is possible to use two predefined motors with two types of feedback sensors (hall sensor and encoder) or use your own motor. The MCU features and peripheral settings (including application timing) are described in Section 3, “KV31F MCU features and peripheral settings”. Sections 4, 5, and 6 describe the project file structure, the tools used, and how to build and debug the application. The FreeMASTER user interface and its web application (MCAT) used to control and tune the application are described in sections 8 and 9. It is also described how to identify the parameters and how to run your own motor.

11. Acronyms and abbreviations

Table 9. Acronyms and abbreviations

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

12. References

These references are available on www.nxp.com:

1. *Sensorless PMSM Field-Oriented Control* (document DRM148).
2. Motor *Control Application Tuning (MCAT) Tool for 3-Phase PMSM* (document AN4642).
3. Sensorless *PMSM Field-Oriented Control on Kinetis KV* (document AN5237).
4. PMSM *Field-Oriented Control on MIMXRT10xx EVK* (document PMSMFOCRT10xxUG)
5. *PMSM Field-Oriented Control on MIMXRT10xx EVK* (document AN12214)

13. Useful links

1. PMSM Control Reference Design www.nxp.com/motorcontrol_pmsm
2. BLDC Control Reference Design www.nxp.com/motorcontrol_bldc
3. ACIM Control Reference Design www.nxp.com/motorcontrol_acim
4. FRDM-KV31F Freedome Development Platform
5. FRDM-MC-PMSM Freedome Development Platform
6. MCUXpresso SDK Builder (SDK examples in several IDEs)
   www.mcuxpresso.nxp.com/en/builder

Document Number: AN12374
Rev. 0
03/2019