UG10263

MCUXpresso SDK Field-Oriented Control of 3-Phase PMSM and BLDC Motors (i.MX943-EVK)

Rev. 1.0 — 3 July 2025 User guide

Document information

Information	Content
Keywords	i.MX943-EVK, PMSM, FOC, MCAT, MID, Motor control, Sensorless control, Speed control, Servo control, Position control
Abstract	This user guide describes the implementation of the motor-control software for 3-phase Permanent Magnet Synchronous Motors.

1 Introduction

SDK motor control example user guide describes the implementation of the motor-control software for 3-phase Permanent Magnet Synchronous Motors (PMSM) using following NXP platforms:

- i.MX943-EVK
- Freedom Development Platform for Low-Voltage, 3-Phase PMSM Motor Control - Factory automation (FRDM-LVPMSM-FA)
  Note: *This platform is currently available for internal use only.*

The document is divided into several parts. Hardware setup, processor features, and peripheral settings are described at the beginning of the document. The next part contains the PMSM project description and motor control peripheral initialization. The last part describes user interface and additional example features.

Available motor control examples types with supported motors, and possible control methods are listed in Table 1.

Table 1. Available example type, supported motors and control methods

Example type	Supported motor	Possible control methods in SDK example				
		Scalar and Voltage	Current FOC (Torque)	Sensorless Speed FOC	Sensored Speed FOC	Sensored Position FOC
pmsm_servo	TGT3-0065-60-48	✓	✓	✓	✓	✓
pmsm_servo_dual	HMD06-011-048 (Motor Control 1)	✓	✓	✓	✓	✓
	TGT3-0065-60-48 (Motor Control 2)	✓	✓	✓	✓	✓

SDK motor control example description:

- **pmsm_servo** - pmsm servo example uses float arithmetic, the example contains improved position Field Oriented Control (FOC) loops with feed-forward and also sensorless FOC. Digital encoder is used instead of the quadrature encoder. This example can be used for single motor control application. Default motor configuration is tuned for the TGT3-0065-60-48 motor.
- **pmsm_servo_dual** - pmsm servo dual example uses float arithmetic, the example contains improved position Field Oriented Control (FOC) loops with feed-forward and also sensorless FOC. Digital encoder is used instead of the quadrature encoder. This example can be used for dual motor control application. Default motor configurations are tuned for the Hedrive HMD06-011-048 motor (Arduino Motor Control 1 connector) and for the TGT3-0065-60-48 motor (Arduino Motor Control 2 connector).
  Note: *For performance optimization, the macro `SERVO_OPTIM` is defined in the project settings by default. Only position control is available in this case. Remove this macro to enable all control modes.*

The SDK motor control example contains several additional features:

- **FreeMASTER** pmsm_float_servo_dual.pmpx project provides a simple and user-friendly way for algorithm tuning, software control, debugging, and diagnostics.
- **MCAT** - Motor Control Application Tuning page based on the FreeMASTER runtime debugging tool.

The control software and the PMSM control theory, in general, are described in *Sensorless PMSM Field-Oriented Control (FOC)* (document DRM148).

2 Hardware setup

The following chapter describes the used hardware and the setup needed for proper example working

2.1 TGT motor

The TG drives TGT3-0065-60-48/T5PUXS4-H09 motor is a low-voltage 3-phase permanent-magnet motor with EnDat2.2 high resolution encoder used in PMSM servo applications. The motor parameters are listed bellow.

Table 2. TGT3-0065-60-48 motor parameters

Characteristic	Symbol	Value	Units
Rated voltage	Vt	48	V
Rated speed	-	6000	RPM
Rated torque	T	0.62	Nm
Rated power	P	346	W
Continuous current	Ics	11.1	A
Number of pole-pairs	pp	3	-

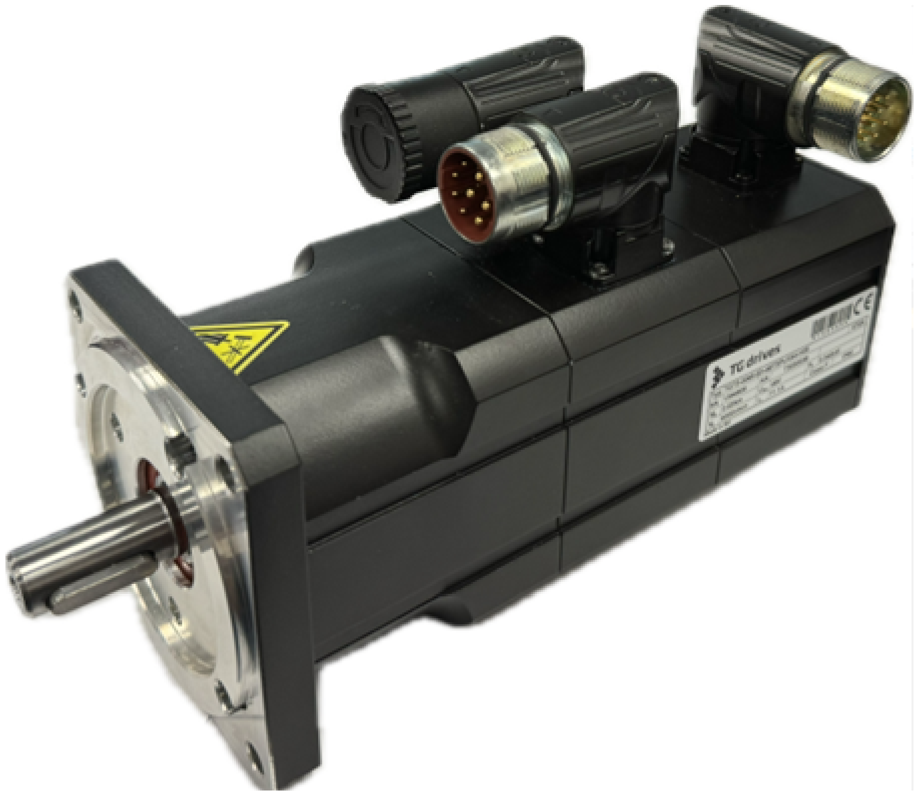


Figure 1. TGT3-0065-60-48 permanent magnet synchronous motor

The motor has two types of connectors (cables). The first cable has eight wires and is designated to power and brake the motor. The second cable has twelve wires and is designated for the EnDat encoder communication.

2.2 Heidrive motor

The Heidrive HMD06-011-048-60-OPB1MW230 motor is a low-voltage 3-phase permanent-magnet motor with BiSS high resolution encoder used in PMSM servo applications. The motor parameters are listed bellow.

Table 3. HMD06-011-048 motor parameters

Characteristic	Symbol	Value	Units
Rated voltage	Vt	27.2	V
Rated speed	-	6000	RPM
Rated torque	T	0.77	Nm
Rated power	P	630	W
Continuous current	Ics	12.4	A
Number of pole-pairs	pp	5	-

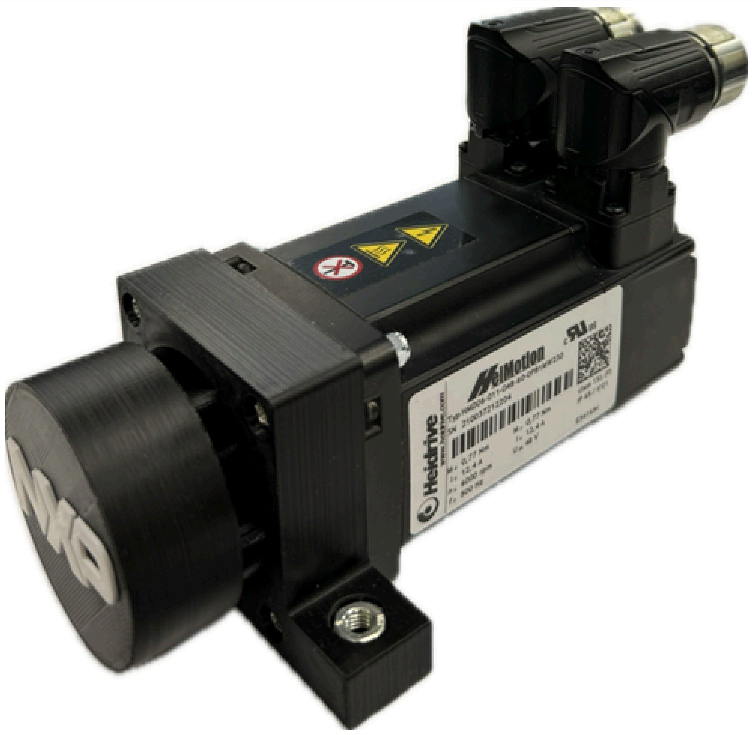


Figure 2. HMD06-011-048 permanent magnet synchronous motor

The motor has two types of connectors (cables). The first cable has eight wires and is designated to power and brake the motor. The second cable has twelve wires and is designated for the BiSS encoder communication.

2.3 FRDM-LVPMSM-FA

In a shield form factor, this evaluation board effectively turns an NXP Freedom development board or an evaluation board into a complete motor-control reference design. It is compatible with existing NXP Freedom development boards and evaluation boards. The Freedom motor-control headers are compatible with the Arduino R3 pin layout.

Freedom Development Platform for Low-Voltage, 3-Phase PMSM Motor Control - Factory automation (FRDM-LVPMSM-FA) development platform board features:

- Up to 48V/12A output.
- Phase current and DCbus voltage measurement using Sigma Delta modulators OR external/internal OpAmps.
- Interface for Quadrature encoder or Digital encoders via RS-485.
- Over voltage, over current fault protection.

Note: *The FRDM-LVPMSM-FA board is currently available for internal use only.*

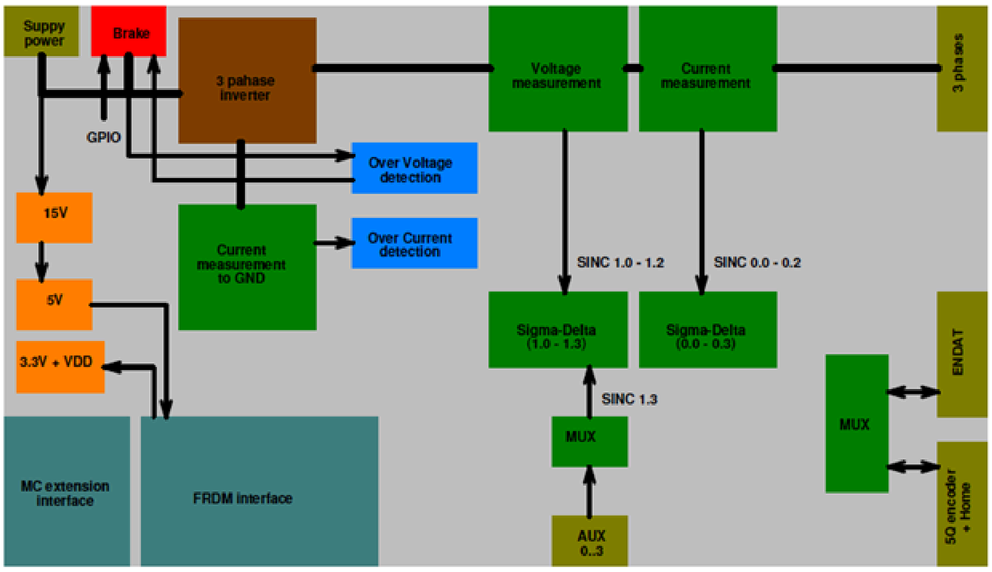


Figure 3. FRDM-LVPMSM-FA block diagram

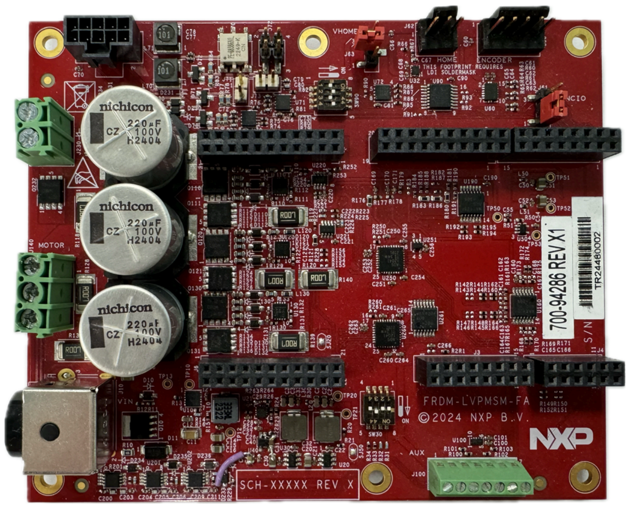


Figure 4. FRDM-LVPMSM-FA board

Switches SW30 and SW90 should be set with regards to used encoder type. SW30 setting - Select the appropriate voltage based on the properties of the sensor. For the SW90 setting, see the tables below or set base on used encoder.

Table 4. SW90 setting for BiSS encoder

SW position	SW state	Description
1:2	ON:OFF	Full duplex
3	OFF	Echo Enabled

Table 4. SW90 setting for BiSS encoder*...continued*

SW position	SW state	Description
4	ON	ENDAT is used

Table 5. SW90 setting for EnDat2.2 encoder

SW position	SW state	Description
1:2	OFF:ON	Half duplex
3	OFF	Echo Enabled
4	ON	ENDAT is used

Connect sensor signals to the J70 connector on the FRDM-LVPMSM-FA as follow:

Table 6. BiSS point-to-point connection

Pin name (FRDM-LVPMSM-FA)	Pin location (FRDM-LVPMSM-FA)	Sensor signal name
ENC_CLK_P	J70-2	MASTER_CLOCK_INPUT_P
ENC_CLK_N	J70-7	MASTER_CLOCK_INPUT_N
ENC_DATA_IN_P	J70-3	SLAVE_DATA_P
ENC_DATA_IN_N	J70-8	SLAVE_DATA_N
VENC	J70-1	VDC - Power supply voltage
GND	J70-6	GND

Table 7. EnDat2.2 connection

Pin name (FRDM-LVPMSM-FA)	Pin location (FRDM-LVPMSM-FA)	Sensor signal name
ENC_CLK_P	J70-2	CLOCK_P
ENC_CLK_N	J70-7	CLOCK_N
ENC_DATA_IO_P	J70-4	DATA_P
ENC_DATA_IO_N	J70-9	DATA_N
VENC	J70-1	UP - Power supply voltage
GND	J70-6	GND

2.4 i.MX943-EVK board

The i.MX943-EVK board offers a rich set of peripherals targeting industrial, automotive telematics, and consumer IoT market segments. The i.MX 943 applications processors integrate up to four Arm Cortex-A55 cores and supports functional safety with built-in 2x Arm Cortex -M33 and -M7 cores.

Table 8. i.MX943-EVK switch settings

Switch	Setting
SW4	based on used BOOT_MODE
SW7 [1:4]	ON:OFF:ON:ON
SW8	all to OFF

All others jumpers are open.

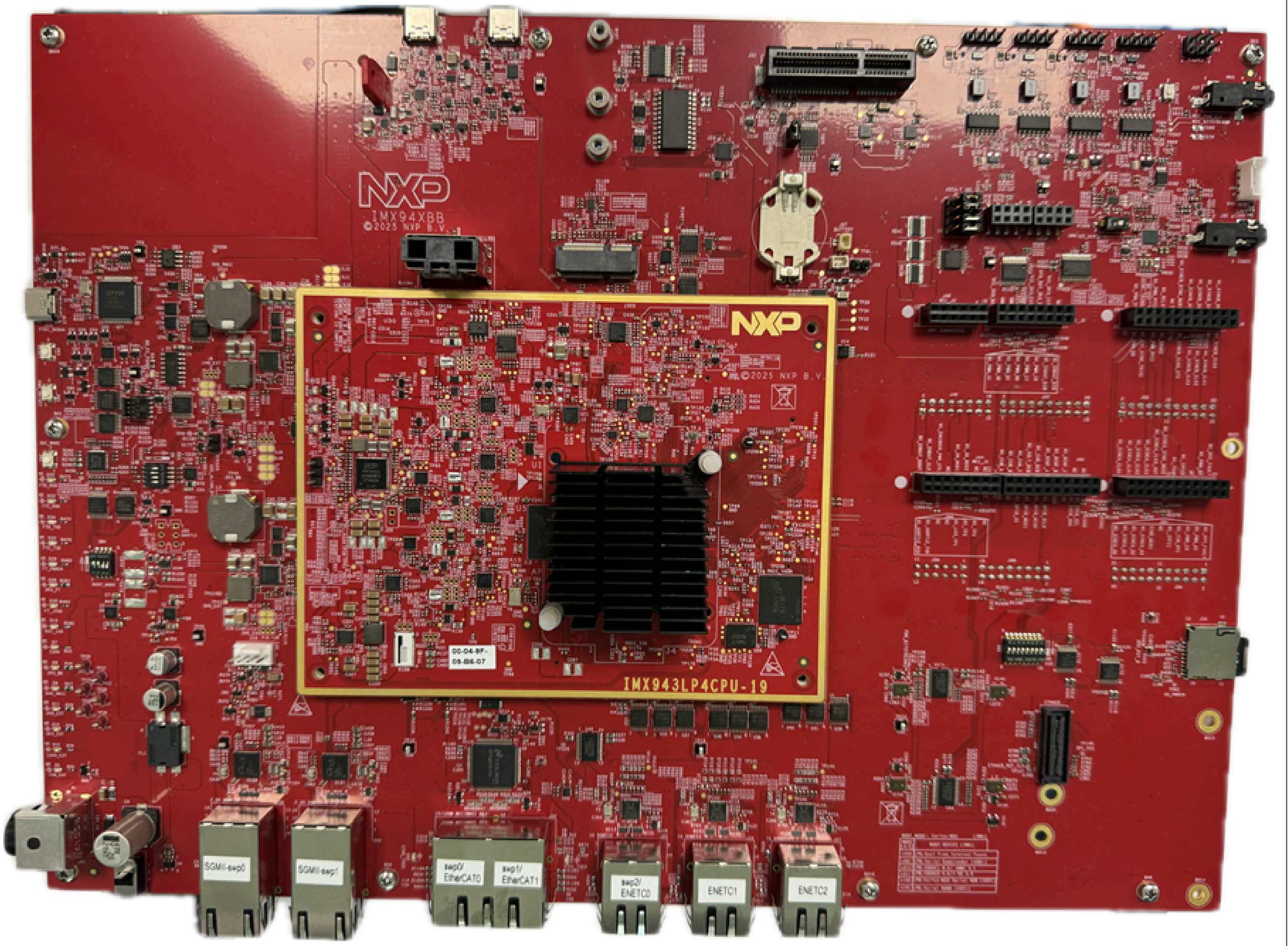


Figure 5. i.MX943-EVK board

2.4.1 Hardware assembling

1. Connect the FRDM-LVPMSM-FA shield on Motor Control 1 or Motor Control 2 connector of the i.MX943-EVK board (with regard to used example).
2. Connect the 3-phase motor wires to the screw terminals (J140) on the FRDM-LVPMSM-FA board.
3. Connect the sensor signal wires to the J70 connector on the FRDM-LVPMSM-FA board.
4. Plug the USB cable from the USB host to the FTDI_DEBUG USB connector J15 on the i.MX943-EVK board.
5. Plug the 12-V DC power supply to the P1 connector on the i.MX943-EVK board.
6. Plug the 24-V DC power supply to the J10 connector on the FRDM-LVPMSM-FA board.

2.4.2 Additional information

- *pmsm_servo* example is available on Arduino Motor Control 1 connector. EnDat2.2 encoder is used.
- *pmsm_servo_dual* example - BiSS encoder is used on Arduino Motor Control 1 connector. EnDat2.2 encoder is used on Arduino Motor Control 2 connector. MCAT interface is available only on Arduino Motor Control 1 connector.

3 Processors features and peripheral settings

This chapter describes the peripheral settings and application timing.

3.1 i.MX 94x

For more information about i.MX94 processor family, see product web pages.

3.1.1 i.MX94X - Hardware timing and synchronization

Correct and precise timing is crucial for motor-control servo applications. Therefore, the motor-control-dedicated peripherals take care of the timing and synchronization on the hardware layer. PWM frequency is 32 kHz. Fast loop frequency is 64 kHz.

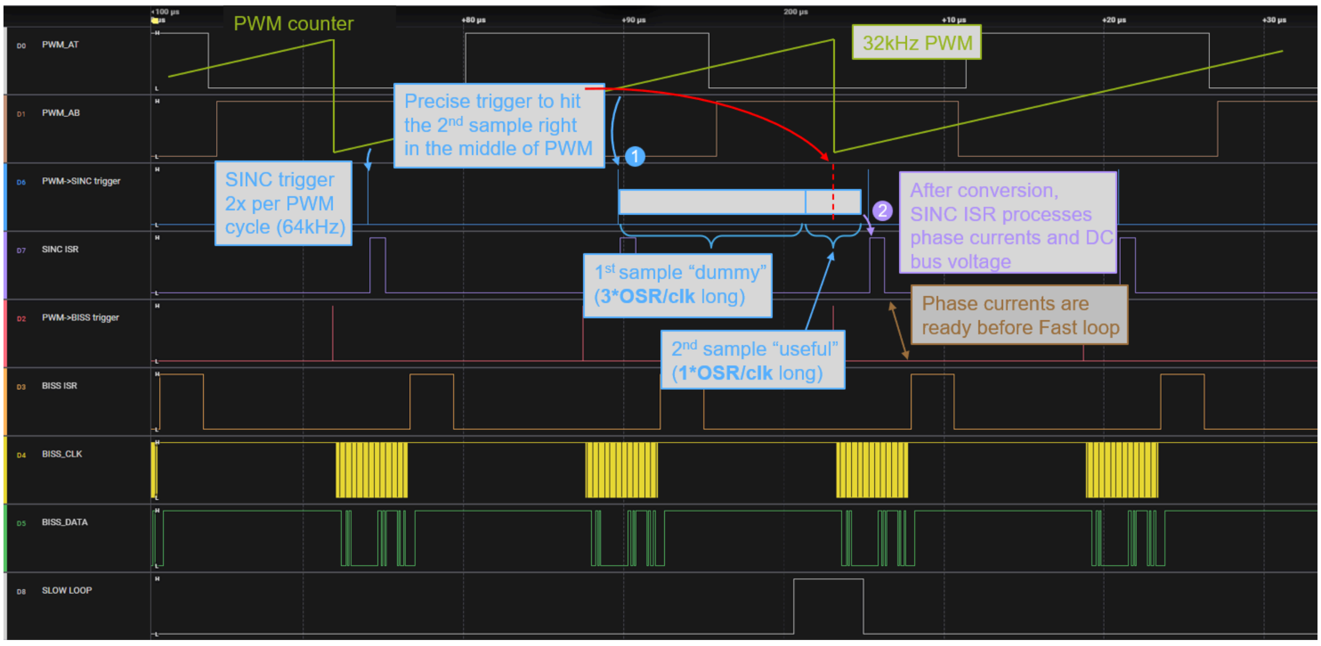


Figure 6. Timing on i.MX94X - SINC trigger and ISR

- The top signals show the PWM_AT (PWM phase A - top) and PWM_AB (PWM phase A - bottom).
- The eFlexPWM submodule SM0 generates trigger 0 (SINC trigger) to hit the second sample right in the middle/top of PWM.
- When the SINC conversion is completed, the SINC ISR (SINC interrupt) is entered. The phase currents and DC bus voltage are process in this interrupt to be ready for the Fast loop.

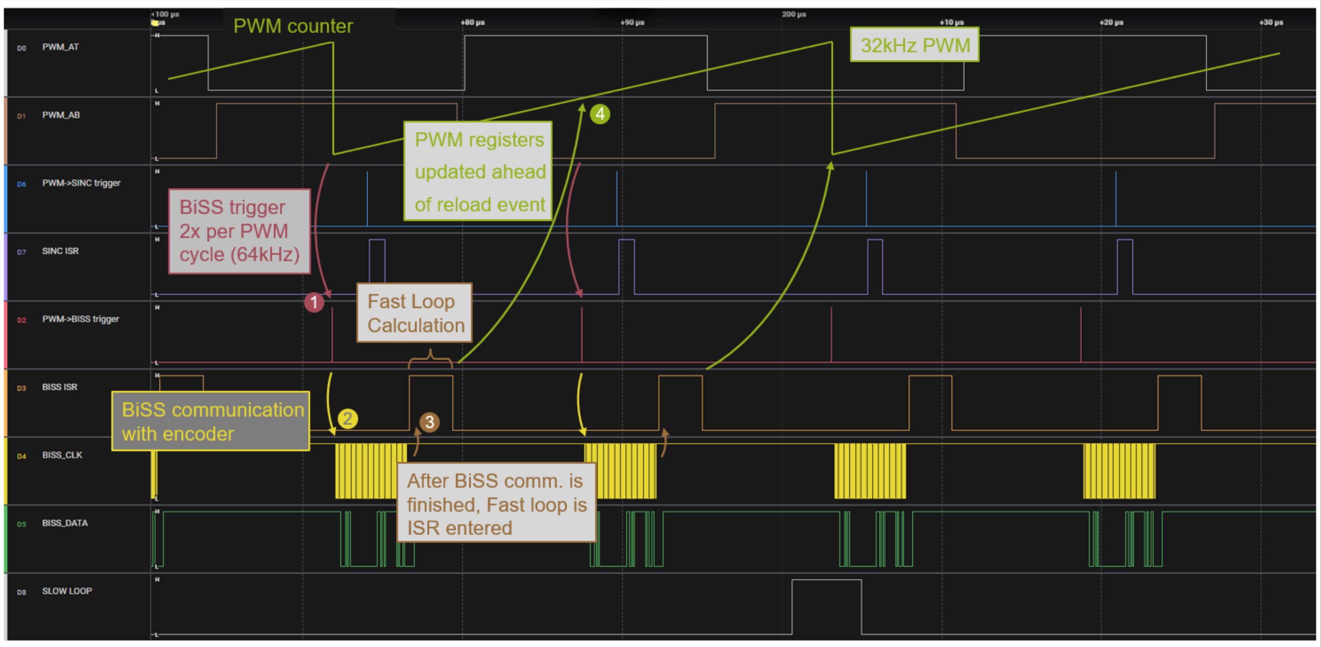


Figure 7. Timing i.MX94X - BiSS trigger and Fast loop ISR

- The eFlexPWM submodule 1 generates trigger 0 (BiSS trigger) to request data from the BiSS encoder sensor.
- After BiSS communication is finisned, BiSS ISR is entered and Fast loop is process.
- Similar approach is used for the EnDat2.2 encoder triggering.

High-level timing diagram for the dual motor-control servo application can be seen on the folowing image.

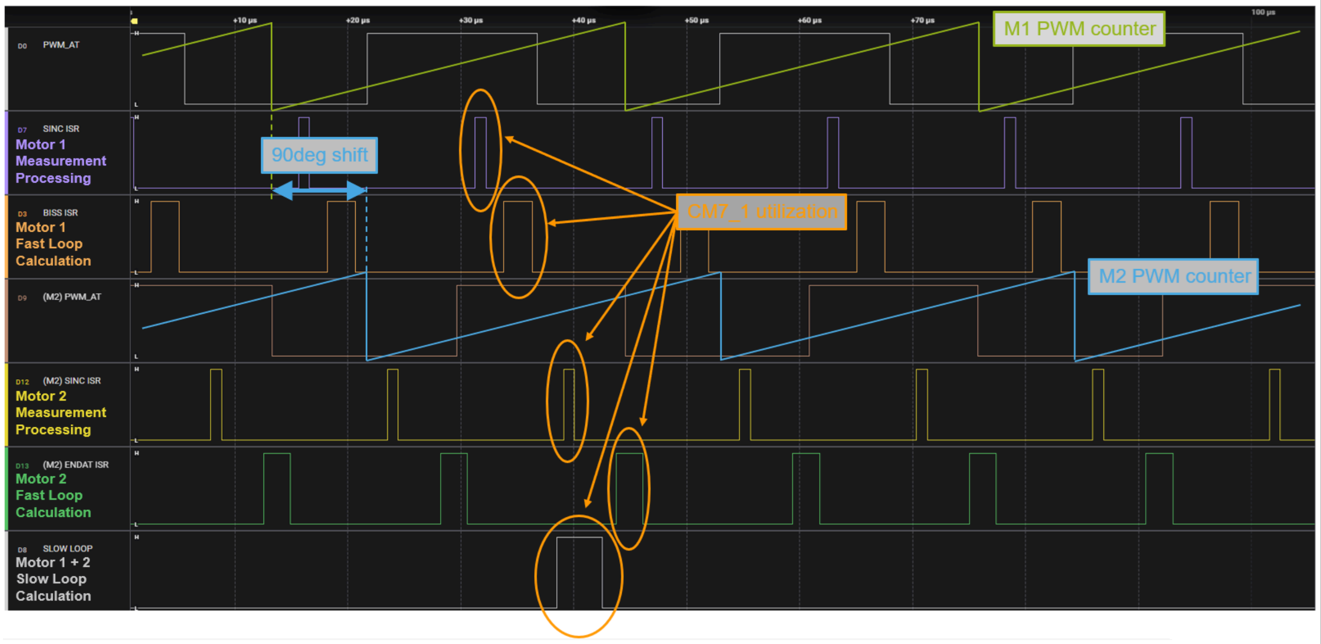


Figure 8. Timing i.MX94X - dual servo application

3.1.2 i.MX94X - Peripheral settings

This section describes the peripherals used for the motor control applications. On i.MX94X, three submodules from the enhanced FlexPWM (eFlexPWM) are used for 6-channel PWM generation and four channels of SINC filter for the phase currents and DC-bus voltage measurement. The eFlexPWM and SINC are synchronized via submodule 0 from the eFlexPWM. Position feedback is ensured by digital encoders communicating via BiSS or EnDat2.2 interface. The following settings are located in the `mc_periph_init.c` and `pin_mux.c` files and their header files.

3.1.2.1 PWM generation

- Six channels from three submodules are used for the 3-phase PWM generation.
- Submodules 1 and 2 get their clocks from submodule 0.
- The counters at submodules 1 and 2 are synchronized with the master reload signal from submodule 0.
- Submodule 0 trigger 0 is used for synchronization with the `SINC` filter. The submodule generates the output trigger to hit the second sample of SINC filter conversion right in the the middle/top of PWM cycle.
- Submodule 1 trigger 0 is used for trigger communication with digital encoders.
- Submodule 0 trigger 1 is used for synchronization with second instance of the PWM module. Second PWM is shifted by 90 degrees.
  Note: *This setting only applies to dual motor application.*
- Fault mode is enabled for channels A and B at submodules 0, 1, and 2 with automatic fault clearing.
  Note: *The PWM outputs are re-enabled at the first PWM reload after the fault input returns to zero.*
- The PWM period (frequency) is determined by how long the counter takes to count from INIT to VAL1. By default, INIT = -MODULO/2 and VAL1 = MODULO/2 -1 where MODULO = FastPeripheralClock / `M1_PWM_FREQ`. PWM frequency is set to 32 kHz by default.
- Dead time insertion is enabled. Define the dead time length in the `M1_PWM_DEADTIME` macro.
- PWM registers are updated on both half and full cycle reload events.

3.1.2.2 SINC filter

- SINC filter is used for the MC sensing of phase currents and DC-bus voltage. SINC converts an external ADC sigma-delta modulator bitstream to a data stream.
- SINC operates as a 3^{rd}-order filter. Filter order is defined by `SINC1_ORD` macro.
- SINC Oversampling ration (OSR) is set to 64. OSR value is defined as `OSR = SINC1_OSR + 1`, where `SINC1_OSR` macro is set to 63.
- SINC operates in a continuous conversion mode. Hardware trigger is used.

3.1.2.3 BiSS interface

- BiSS interface is used for a communication with the digital encoder.
- BiSS communication runs at 10 MHz.
- 16-bits single-turn resolution, 12-bits multi-turn resolution.
- Available only for dual motor application.
- For mor information about BiSS interface navigate to BiSS Association e.V. website.

3.1.2.4 EnDat2.2 interface

- EnDat2.2 interface is used for a communication with the digital encoder.
- EnDat2.2 communications runs at 8.333 MHz.
- 25-bits single-turn resolution.
- For mor information about EnDat2.2 interface navigate to HEIDENHAIN website.

3.1.2.5 XBAR - peripheral interconnection

Inter-Peripheral Crossbar Switch (XBAR) is used to interconnect signals. The following signals are interconnected using XBAR:

- The PWM output trigger 0 (generated by submodule 0) to the SINC input hardware trigger.
- The PWM output trigger 0 (generated by submodule 1) to the BiSS GETSENS signal.
- The PWM output trigger 0 (generated by submodule 1) to the EnDat2.2 nstr signal.
- The PWM output trigger 1 (generated by submodule 0) to the PWM ExtSync0 signal.
- The GPIO inputs to the PWM Fault signals (over current and over voltage fault protection).

3.1.2.6 Slow-loop interrupt generation

The QuadTimer module TMR1 is used to generate the slow-loop interrupt.

- The interrupt is generated after the counter counts from LOAD = 0 to COMP1 = FastPeripheralClock / (16U * Speed Loop Freq). The speed loop frequency is set in the `M1_SPEED_LOOP_FREQ` macro and equals 4000 Hz.
- An interrupt (which serves the slow-loop period) is enabled and generated at the reload event.

3.2 CPU load and memory usage

The following information applies to the application built using one of the following IDE: MCUXpresso IDE, IAR, Keil MDK or CodeWarrior. The memory usage is calculated from the *.map linker file, including FreeMASTER recorder buffer allocated in RAM. In the MCUXpresso IDE, the memory usage can be also seen after project build in the Console window. The table below shows the maximum CPU load of the supported examples. The CPU load is measured using the SYSTICK timer. The CPU load is dependent on the fast-loop (FOC calculation) and slow-loop (speed loop) frequencies. The total CPU load is calculated using the following equations:

$$CPU_{fast} = cycles_{fast}\frac{f_{fast}}{f_{CPU}}100\left[\%\right] \tag{1}$$

$$CPU_{slow} = cycles_{slow}\frac{f_{slow}}{f_{CPU}}100\left[\%\right] \tag{2}$$

$$CPU_{total} = CPU_{fast} + CPU_{slow}\left[\%\right] \tag{3}$$

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

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

CPU_{total} = the total CPU load consumed by the motor control

Table 9. Maximum CPU load (fast loop)

		Debug configuration
Device	Example	Position Control
i.MX943-EVK	pmsm_servo	22.2 %
i.MX943-EVK	pmsm_servo_dual	45.4 %

Table 10. Memory usage

	Debug configuration	Release configuration
Readonly code memory	60 449 B	44 436 B
Readwrite code memory	80 B	80 B
Readonly data memory	22 528 B	18 074 B
Readwrite data memory	11 239 B	11 248 B
Parameters	pmsm_servo_dual example, IAR IDE, `SERVO_OPTIM` macro defined	

CPU load measured with defined `RAM_RELOCATION` and `SERVO_OPTIM` macros. Measured CPU load applies to the application built using IAR IDE.

Note: *Memory usage and maximum CPU load can differ depending on the used IDEs and settings.*

4 SDK package project file and IDE workspace structure

All the necessary files are included in one package, which simplifies the distribution and decreases the size of the final package. The directory structure of this package is simple, easy to use, and organized logically. The folder structure used in the IDE differs from the structure of the PMSM package installation, but it uses the same files. The different organization is chosen due to better manipulation of folders and files in workplaces and the possibility of adding or removing files and directories. The `pack_motor_<board_name>` project includes all the available functions and routines. This project serves for development and testing purposes.

4.1 PMSM project structure

The directory tree of the PMSM project is shown in below.

```
└── pack_motor_<board_name>/
    ├── boards/
    │   └── imx943evk/
    │       └── demo_apps/
    │           └── mc_pmsm/
    │               ├── pmsm_servo/
    │               └── pmsm_servo_dual/
    │                   └── cm7_core1/
    │                       ├── iar/
    │                       └── armgcc/
    ├── CMSIS/
    ├── components/
    ├── devices/
    ├── docs/
    ├── examples/
    ├── middleware/
    │   ├── freemaster/
    │   ├── motor_control/
    │   │   ├── freemaster/
    │   │   ├── licenses/
    │   │   └── pmsm/
    │   │       ├── pmsm_float/
    │   │       │   ├── mc_algorithms/
    │   │       │   ├── mc_cfg_template/
    │   │       │   ├── mc_drivers/
    │   │       │   ├── mc_identification/
    │   │       │   └── mc_state_machine/
    │   │       └── state_machine/
    │   └── rtcesl/
    └── tools/
```

The main project folder `pack_motor_<board_name>\boards\imx943evk\demo_apps\mc_pmsm\pmsm_servo` contains the following folders and files (the same is included in `pmsm_servo_dual` folder):

- `iar`— for the IAR Embedded Workbench IDE.
- `armgcc`—for the GNU Arm IDE.
- `m1_pmsm_appconfig.h`—contains the definitions of constants for the application control processes, parameters of the motor and regulators, and the constants for other vector control-related algorithms. When you tailor the application for a different motor using the Motor Control Application Tuning (MCAT) tool, the tool generates this file at the end of the tuning process.

- `m2_pmsm_appconfig.h`—contains constants and parameters tuned for another motor. In the dual motor application, this parameters set is used for the second motor.
- `main.c`—contains the basic application initialization (enabling interrupts), subroutines for accessing the MCU peripherals, and interrupt service routines. The FreeMASTER communication is performed in the background infinite loop.
- `board.c`—contains the functions for the UART, GPIO, and SysTick initialization.
- `board.h`—contains the definitions of the board LEDs, buttons, UART instance used for FreeMASTER, and so on.
- `clock_config.c and .h`—contains the CPU clock setup functions. These files are going to be generated by the clock tool in the future.
- `mc_periph_init.c`—contains the motor-control driver peripherals initialization functions that are specific for the board and MCU used.
- `mc_periph_init.h`—header file for `mc_periph_init.c`. This file contains the macros for changing the PWM period, and for reading the phase currents, DC bus voltage, position and speed.
- `freemaster_cfg.h`—the FreeMASTER configuration file containing the FreeMASTER communication and features setup.
- `pin_mux.c and .h`—port configuration files. It is recommended to generate these files in the pin tool.
- `peripherals.c and .h`—MCUXpresso Config Tool configuration files.

The main motor-control folder `pack_motor_<board_name>\middleware\motor_control\` contains these subfolders:

- `pmsm`—contains main pmsm motor-control functions
- `freemaster`—contains the FreeMASTER project file `pmsm_float_servo_dual.pmpx`. Open this file in the FreeMASTER tool and use it to control the application. The folder also contains the auxiliary files for the MCAT tool.

The `pack_motor_<board_name>\middleware\motor_control\pmsm\pmsm_float` folder contains the following subfolders common to the other motor-control projects:

- `mc_algorithms`—contains the main control algorithms used to control the FOC and speed control loop. Folder also contains MCAA library.
- `mc_cfg_template`—contains templates for MCUXpresso Config Tool components.
- `mc_drivers`—contains the source and header files used to initialize and run motor-control applications.
- `mc_identification`—contains the source code for the automated parameter-identification routines of the motor.
- `mc_state_machine`—contains the software routines that are executed when the application is in a particular state or state transition.
- `state_machine`—contains the state machine functions for the FAULT, INITIALIZATION, STOP, and RUN states.

5 Github project structure

More informations for examples released through NXP MCUXpresso Github are available on link MCUXpresso SKD Documentation.

6 Motor-control peripheral initialization

The motor-control peripherals are initialized by calling the `MCDRV_Init()` function during MCU startup and before the peripherals are used. All initialization functions are in the `mc_periph_init.c` source file and the `mc_periph_init.h` header file. The definitions specified by the user are also in these files. The features provided by the functions are the 3-phase PWM generation and 3-phase current measurement and the DC-bus voltage measurement. Moreover, the reading and processing data from the position sensors are provided as well. 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 `mc_periph_init.h` header file provides the following macros defined by the user:

- `M1_MCDRV_SINC_INIT`: this macro calls SINC peripheral initialization.
- `M1_MCDRV_PWM_PERIPH_INIT`: this macro calls PWM peripheral initialization.
- `M1_MCDRV_ENDAT2P2_PERIPH_INIT`: this macro calls EnDat2.2 peripheral initialization. Available in the single motor application.
- `M1_MCDRV_BISS_PERIPH_INIT`: this macro calls BiSS peripheral initialization. Available in the dual motor application.
- `M2_MCDRV_ENDAT2P2_PERIPH_INIT`: this macro calls EnDat2.2 peripheral initialization. Available in the dual motor application.
- `M1_PWM_FREQ`: the value of this definition sets the PWM frequency.
- `M1_FOC_FREQ_VS_PWM_FREQ`: enables you to call the fast-loop interrupt at every first, second, third, or n^{th} PWM reload. This is convenient when the PWM frequency must be higher than the maximal fast-loop interrupt.
- `M1_SPEED_LOOP_FREQ`: the value of this definition sets the speed loop frequency.
- `M1_PWM_DEADTIME`: the value of the PWM dead time in nanoseconds.
- The macros related to the second motor are prefixed with `M2_`.

In the motor-control software, the following API-serving SINC and PWM peripherals are available:

- The available APIs for the SINC are:
  - `mcdrv_sinc_t`: MCDRV SINC structure data type.
  - `void M1_MCDRV_SINC_INIT()`: this function is by default called during the SINC peripheral initialization procedure invoked by the `MCDRV_Init()` function and should not be called again after the peripheral initialization is done.
  - `void M1_MCDRV_SINC_GET(mcdrv_sinc_t*)`: this function reads and process the actual values of the 3-phase currents and DC-bus voltage.
  - The APIs for SINC control of the second motor are prefixed with `M2_`.
- The available APIs for the PWM are:
  - `mcdrv_pwm3ph_pwma_t`: MCDRV PWM structure data type.
  - `void M1_MCDRV_PWM_PERIPH_INIT`: this function is by default called during the PWM periphery initialization procedure invoked by the `MCDRV_Init()` function.
  - `void M1_MCDRV_PWM3PH_SET(mcdrv_pwm3ph_pwma_t*)`: this function updates the PWM phase duty cycles.
  - `void M1_MCDRV_PWM3PH_EN(mcdrv_pwm3ph_pwma_t*)`: this function enables all PWM channels.
  - `void M1_MCDRV_PWM3PH_DIS(mcdrv_pwm3ph_pwma_t*)`: this function disables all PWM channels.
  - `bool_t M1_MCDRV_PWM3PH_FLT_GET(mcdrv_pwm3ph_pwma_t*)`: this function returns the state of the overcurrent fault flag and automatically clears the flag (if set). This function returns true when an overcurrent event occurs. Otherwise, it returns false.
  - `bool_t M1_MCDRV_PWM3PH_FLT_OV_GET(mcdrv_pwm3ph_pwma_t*)`: this function returns the state of the overvoltage fault flag and automatically clears the flag (if set). This function returns true when an overvoltage event occurs. Otherwise, it returns false.

  - The APIs for PWM control of the second motor are prefixed with `M2_`.
- The available APIs for the BiSS sensor are:
  - `BISSC_Type`: MCDRV BiSS structure data type.
  - `void M1_MCDRV_BISS_PERIPH_INIT`: this function is by default called during the BiSS periphery initialization procedure invoked by the `MCDRV_Init()` function.
  - `void M1_MCDRV_ENC_GET_DATA_FAST(BISSC_Type*)`: this function reads and process the actual electrical position from BiSS sensor.
  - `void M1_MCDRV_ENC_GET_DATA_SLOW(BISSC_Type*)`: this function reads and process the actual mechanical position and speed from BiSS sensor.
  - `void M1_MCDRV_ENC_CLEAR(BISSC_Type*)`: this function clears the internal variables and revolutions.
  - `void M1_MCDRV_ENC_SET_OFFSET(BISSC_Type*)`: this function sets the revolutions offset.
  - The APIs related to the BiSS sensor are available only in the dual motor application.
- The available APIs for the EnDat2.2 sensor are:
  - `mcdrv_endat2p2_t`: MCDRV EnDat2.2 structure data type.
  - `void M2_MCDRV_ENDAT2P2_PERIPH_INIT`: this function is by default called during the EnDat2.2 periphery initialization procedure invoked by the `MCDRV_Init()` function.
  - `void M2_MCDRV_ENC_GET_DATA_FAST(mcdrv_endat2p2_t*)`: this function reads and process the actual electrical position from EnDat2.2 sensor.
  - `void M2_MCDRV_ENC_GET_DATA_SLOW(mcdrv_endat2p2_t*)`: this function reads and process the actual mechanical position and speed from EnDat2.2 sensor.
  - `void M2_MCDRV_ENC_CLEAR(mcdrv_endat2p2_t*)`: this function clears the internal variables and revolutions.
  - `void M2_MCDRV_ENC_SET_OFFSET(mcdrv_endat2p2_t*)`: this function sets the revolutions offset.
  - The APIs related to the EnDat2.2 sensor are available in the single motor application and are prefixed with `M1_`.

Note: *Not all macros are available for every motor control example type. The structure data types may vary across the platforms.*

7 User interface

The application contains the demo mode to demonstrate motor rotation. You can operate it either using the user button, or using FreeMASTER. The NXP development 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. The serial interface transfers data between the PC and the embedded application. This interface is provided by the debugger included in the boards.

The application can be controlled using the following two interfaces:

- The user button on the development board (controlling the demo mode):
  - i.MX943-EVK - N/A
- Remote control using FreeMASTER (Following chapter):
  - Setting a variable in the FreeMASTER Variable Watch. See chapter Section 8.4

Identify all motor parameters if you are using your own motor (different from the default motors). The automated parameter identification is described in the following sections.

8 Remote control using FreeMASTER

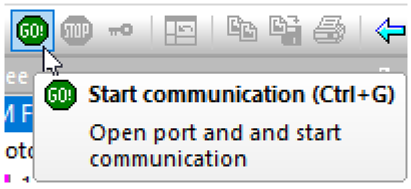
This section provides information about the tools and recommended procedures to control the sensor/sensorless PMSM Field-Oriented Control (FOC) application using FreeMASTER. The application contains the embedded-side driver of the FreeMASTER real-time debug monitor and data visualization tool for communication 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 the latest version of FreeMASTER at www.nxp.com/freemaster. To run the FreeMASTER application including the MCAT tool, double-click the **`pmsm_float_servo_dual`** file located in the `middleware\motor_control\freemaster` folder. The FreeMASTER application starts and the environment is created automatically, as defined in the *.pmpx file.

Note: *In MCUXpresso, the FreeMASTER application can run directly from IDE in `motor_control/freemaster` folder.*

8.1 Establishing FreeMASTER communication

The remote operation is provided by FreeMASTER via the USB interface. To control a PMSM motor using FreeMASTER, perform the steps below:

1. Download the project from your chosen IDE to the MCU and run it.
2. Open the FreeMASTER project `pmsm_float_servo_dual.pmpx` . The PMSM project uses the TSA by default, so it is not necessary to select a symbol file for FreeMASTER.
3. To establish the communication, click the communication button (the green "GO" button in the top left-hand corner).



Start communication (Ctrl+G)

Open port and and start communication

Figure 9. Green “GO” 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 "RS-232 UART Communication; COMxx; speed=115200". Otherwise, the FreeMASTER warning pop-up window appears.

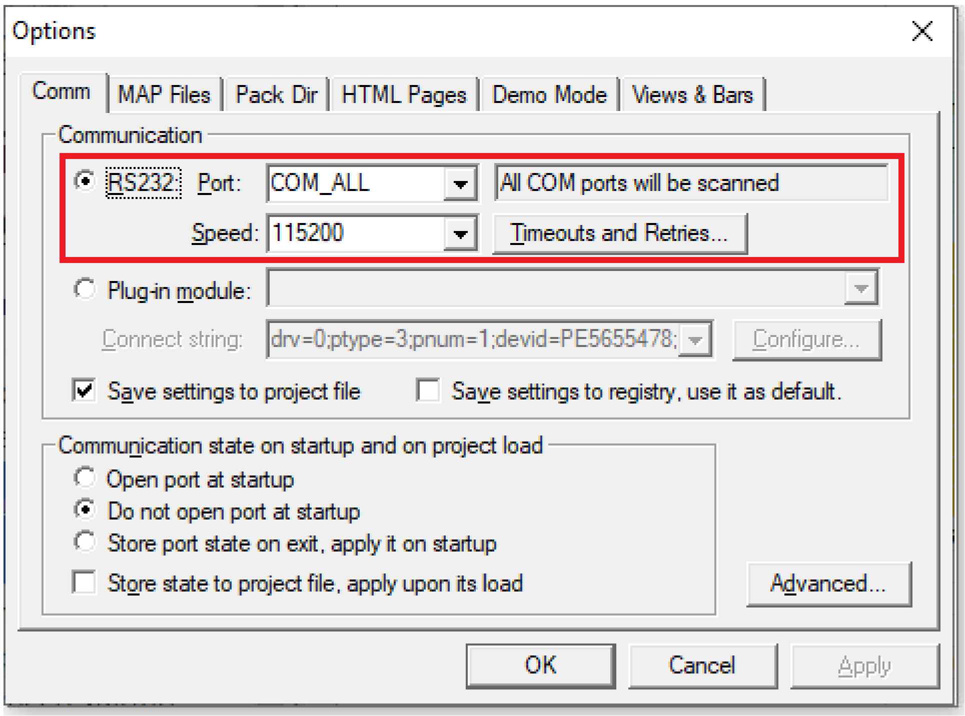
RS232 UART Communication; COM5; speed=115200

Figure 10. FreeMASTER—communication is established successfully

5. To reload the MCAT HTML page and check the App ID, press F5.
6. Control the PMSM motor by writing to a control variable in a variable watch.
7. If you rebuild and download the new code to the target, turn the FreeMASTER application off and on.

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

1. Go to the **Project** > **Options** > **Comm** tab and make sure that the correct COM port is selected and the communication speed is set to 115200 bps.



Communication

[✓ RS232] Port: COM_ALL — All COM ports will be scanned

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 11. FreeMASTER communication setup window

2. Ensure, that your computer is communicating with the plugged board. Unplug and then plug in the USB cable and reopen the FreeMASTER project.

8.2 TSA replacement with ELF file

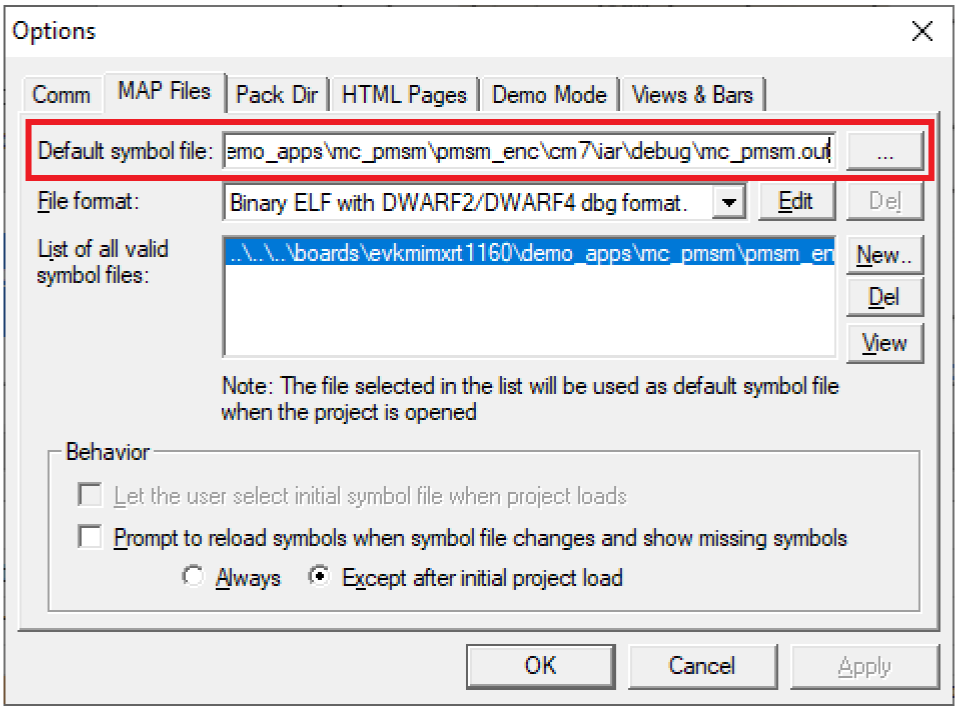
The FreeMASTER project for motor control example uses Target-Side Addressing (TSA) information about variable objects and types to be retrieved from the target application by default. With the TSA feature, you can describe the data types and variables directly in the application source code and make this information available to the FreeMASTER tool. The tool can then use this information instead of reading symbol data from the application's ELF/Dwarf executable file.

FreeMASTER reads the TSA tables and uses the information automatically when an MCU board is connected. A great benefit of using the TSA is no issues with the correct path to ELF/Dwarf file. The variables described by TSA tables may be read-only, so even if FreeMASTER attempts to write the variable, the target MCU side denies the value. The variables not described by any TSA tables may also become invisible and protected even for read-only access.

The use of TSA means more memory requirements for the target. If you do not want to use the TSA feature, you must modify the example code and FreeMASTER project.

To modify the example code, follow the steps below:

1. Open motor control project and rewrite macro `FMSTR_USE_TSA` from 1 to 0 in `freemaster_cfg.h` file.
2. Build, download, and run motor control project.
3. Open FreeMASTER project and click to **Project** > **Options** (or use shortcut Ctrl+T).
4. Click to **MAP Files** tab and find Default symbol file (ELF/Dwarf executable file) located in IDE output folder.



Default symbol file: emo_apps\mc_pmsm\pmsm_enc\cm7\iar\debug\mc_pmsm.out

File format: Binary ELF with DWARF2/DWARF4 dbg format.

List of all valid symbol files: ..\..\..\boards\evkmimxrt1160\demo_apps\mc_pmsm\pmsm_en

Note: The file selected in the list will be used as default symbol file when the project is opened

Behavior

☐ Let the user select initial symbol file when project loads

☐ Prompt to reload symbols when symbol file changes and show missing symbols

Always [✓ Except after initial project load]

Figure 12. Default symbol file

5. Click **OK** and restart the FreeMASTER communication.

For more information, check FreeMASTER User Guide.

8.3 Motor Control Aplication Tuning interface (MCAT)

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 page, which runs within the FreeMASTER. The tool consists of the tab menu and workspace as shown in Figure 13. Each tab from the tab menu (4) represents one submodule which enables tuning or controlling different application aspects. Besides the MCAT page for PMSM, several scopes, recorders, and variables in the project tree (5) are predefined in the FreeMASTER project file to further the motor parameter tuning and debugging simplify.

When the FreeMASTER is not connected to the target, the "Board found" line (2) shows "Board ID not found". When the communication with the target MCU is established, the "Board found" line is read from Board ID variable watch and displayed. If the connection is established and the board ID is not shown, press F5 to reload the MCAT HTML page.

There are three action buttons in MCAT (3):

- **Load data** - MCAT input fields (for example, motor parameters) are loaded from `mX_pmsm_appconfig.h` file (JSON formatted comments). Only `existing mX_pmsm_appconfig.h` files can be selected for loading. Loaded `mX_pmsm_appcofig.h` file is displayed in grey field (7).
- **Save data** - MCAT input fields (JSON formatted comments) and output macros are saved to `mX_pmsm_appconfig.h` file. Up to 9 files (`m1-9_pmsm_appconfig.h`) can be selected. A pop-up window with the user motor ID and description appears when a different `mX_pmsm_appcofig.h` file is selected. The motor ID and description are also saved in `mX_pmsm_appcofig.h` as a JSON comment. The embedded code includes `m1_pmsm_appcofig.h` only at single motor control application. Therefore, saving to higher indexed `mX_pmsm_appcofig.h` files has no effect at the compilation stage.
- **Update target** - writes the MCAT calculated tuning parameters to FreeMASTER Variables, which effectively updates the values on target MCU. These tuning parameters are updated in MCU's RAM. To write these

tuning parameters to MCU's flash memory, `m1_pmsm_appcofig.h` must be saved, code recompiled, and downloaded to MCU.

Every time the communication with the board is established or MCAT is refreshed by pressing F5, MCAT looks for the `m1_pmsm_appconfig.h` file. A path to `m1_pmsm_appconfig` is relative to *.pmpx file. The path is composed from the fixed defined folders name and from the following FreeMASTER variables: *User Path 1, User Path 2, Board ID, Example ID*. Variables *User Path* 1 and *User Path* 2 are intended for user's custom path to the `m1_pmsm_appconfig.h`. These variables are defined in example's main.c file where user can modify them. There are several possible paths with different priority (higher to lower) and typical use case:

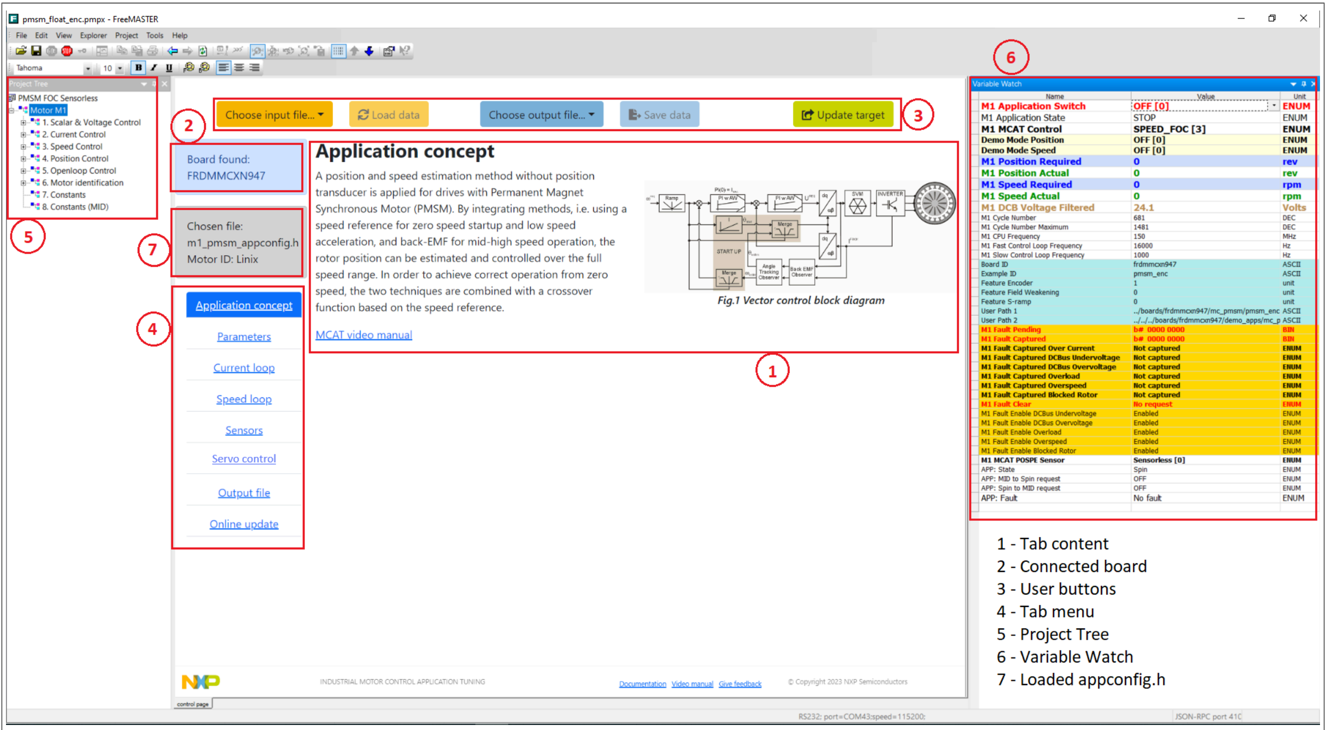
1. *`User Path 1/`*
2. *`User Path 2/`*
3. `../boards/`*`Board ID`*`/mc_pmsm/`*`Example ID/`*
   - typical use case during SDK example development
4. `../../../boards/`*`Board ID`*`/demo_apps/mc_pmsm/`*`Example ID`*`/`
   - typical use case in the SDK package
5. `../../../boards/`*`Board ID`*`/demo_apps/mc_pmsm/`*`Example ID`*`/cm7/`
   - typical use case in the SDK package
6. `../../../boards/`*`Board ID`*`/demo_apps/mc_pmsm/`*`Example ID`*`/cm33_core0/`
   - typical use case in the SDK package
7. `../../source/`
   - typical use case in the workspace

When the `m1_pmsm_appconfig.h` is found in one of the above defined path, it is loaded into the MCAT. Then, MCAT uses this directory for further operations (Load data, Save data). When `m1_pmsm_appconfig.h` is not found even in one of the path above, the default `m1_pmsm_appconfig.h` is loaded. The default `m1_pmsm_appconfig.h` is located in the `mcat` folder. In case of default file is loaded to the MCAT, newly saved `mX_pmsm_appconfig.h` files will be placed next to the *.pmpx file.

Note: *Since the path to `mX_pmsm_appconfig.h` is composed from User Path 1, User Path 2, Board ID and Example ID, FreeMASTER must be connected to the target, and FreeMASTER variables value read prior using Save/Load buttons.*

Note: *Only **Update target** button updates values on the target in real time. Load/Save buttons operate with `mX_pmsm_appcofig.h` file only.*

Note: *MCAT may require Internet connection. If no Internet connection is available, CSS and icons may not be loaded properly.*


pmsm_float_enc.pmpx - FreeMASTER
File Edit View Explorer Project Tools Help
Project Tree
Choose input file...
Load data
Choose output file...
Save data
Update target
Variable Watch
Fig.1 Vector control block diagram
1
2
3
4
5
6
7

PMSM FOC Sensorless
- Motor M1
  - 1. Scalar & Voltage Control
  - 2. Current Control
  - 3. Speed Control
  - 4. Position Control
  - 5. Openloop Control
  - 6. Motor identification
  - 7. Constants
  - 8. Constants (MID)

Board found:
FRDMMCXN947

Chosen file:
m1_pmsm_appconfig.h
Motor ID: Linix

Application concept
Parameters
Current loop
Speed loop
Sensors
Servo control
Output file
Online update

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.

MCAT video manual

Name	Value	Unit
M1 Application Switch	OFF [0]	ENUM
M1 Application State	STOP	ENUM
M1 MCAT Control	SPEED_FOC [3]	ENUM
Demo Mode Position	OFF [0]	ENUM
Demo Mode Speed	OFF [0]	ENUM
M1 Position Required	0	rev
M1 Position Actual	0	rev
M1 Speed Required	0	rpm
M1 Speed Actual	0	rpm
M1 DCB Voltage Filtered	24.1	Volts
M1 Cycle Number	681	DEC
M1 Cycle Number Maximum	1481	DEC
M1 CPU Frequency	150	MHz
M1 Fast Control Loop Frequency	16000	Hz
M1 Slow Control Loop Frequency	1000	Hz
Board ID	frdmmcxn947	ASCII
Example ID	pmsm_enc	ASCII
Feature Encoder	1	unit
Feature Field Weakening	0	unit
Feature S-ramp	0	unit
User Path 1	../boards/frdmmcxn947/mc_pmsm/pmsm_enc	ASCII
User Path 2	../../../boards/frdmmcxn947/demo_apps/mc_p	ASCII
M1 Fault Pending	b# 0000 0000	BIN
M1 Fault Captured	b# 0000 0000	BIN
M1 Fault Captured Over Current	Not captured	ENUM
M1 Fault Captured DCBus Undervoltage	Not captured	ENUM
M1 Fault Captured DCBus Overvoltage	Not captured	ENUM
M1 Fault Captured Overload	Not captured	ENUM
M1 Fault Captured Overspeed	Not captured	ENUM
M1 Fault Captured Blocked Rotor	Not captured	ENUM
M1 Fault Clear	No request	ENUM
M1 Fault Enable DCBus Undervoltage	Enabled	ENUM
M1 Fault Enable DCBus Overvoltage	Enabled	ENUM
M1 Fault Enable Overload	Enabled	ENUM
M1 Fault Enable Overspeed	Enabled	ENUM
M1 Fault Enable Blocked Rotor	Enabled	ENUM
M1 MCAT POSPE Sensor	Sensorless [0]	ENUM
APP: State	Spin	ENUM
APP: MID to Spin request	OFF	ENUM
APP: Spin to MID request	OFF	ENUM
APP: Fault	No fault	ENUM

1 - Tab content
2 - Connected board
3 - User buttons
4 - Tab menu
5 - Project Tree
6 - Variable Watch
7 - Loaded appconfig.h

INDUSTRIAL MOTOR CONTROL APPLICATION TUNING
Documentation Video manual Give feedback
© Copyright 2023 NXP Semiconductors
RS232; port=COM43;speed=115200;
JSON-RPC port 41C

Figure 13. FreeMASTER + MCAT layout

In the default configuration, the following tabs (4) are available:

- Application concept: welcome page with the PMSM sensor/sensorless FOC diagram and a short application description.
- Parameters: this page enables you to modify the motor parameters, hardware and application scales specification, alignment, and fault limits.
- Current loop: current loop PI controller gains and output limits.
- Speed loop: this tab contains fields for the specification of the speed controller proportional and integral gains, as well as the output limits and parameters of the speed ramp. .
- Servo control: this tab contains fields for the specification of the position controller proportional gain, as well as the output limits.
- Sensorless: this page enables you to tune the parameters of the BEMF observer, tracking observer, and open-loop startup.
- Output file: this tab shows all the calculated constants that are required by the PMSM sensor/sensorless FOC application. It is also possible to generate the `mX_pmsm_appconfig.h` file, which is then used to preset all application parameters permanently at the project rebuild.
- Online update : this tab shows actual values of variables on target and new calculated values, which can be used to update the target variables.

Every sublock in FreeMASTER project tree (5) has defined several variables in variable watch (6).

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

8.3.1 MCAT tabs description

This chapter describes MCAT input parameters and equations used to calculate MCAT output (generated) parameters. In the default configuration, the below described tabs are available. Some tabs may be missing if not supported in the embedded code. There are general constants used at MCAT calculations listed in the following table:

Table 11. Constants used in equations

Constant	Value	Unit
UmaxCoeff	1.73205	-
DiscMethodFactor	2	-
k_factor	100	-
pi	3.1416	-

8.3.1.1 Application concept

This tab is a welcome page with the PMSM sensor/sensorless FOC diagram and a short description of the application.

8.3.1.2 Parameters

This tab enables modification of motor parameters, specification of hardware and application scales, alignment, and fault limits. All inputs are described in the following table. MCAT group and MCAT name help to locate the parameter in MCAT layout. Equation name represents the input parameter in equations below.

Table 12. Parameters tab inputs

MCAT group	MCAT name	Equation name	Description	Unit
Motor parameters	PP	parametersPp	Motor number of pole-pairs. Obtain from motor manufacturer or use the pole-pair assistant to determine and then fill manually.	-
	Rs	parametersRs	Stator phase resistance. Obtain from motor manufacturer or use the electrical parameters identification and then fill manually.	[Ω]
	Ld	parametersLd	Stator direct inductance. Obtain from motor manufacturer or use the electrical parameters identification and then fill manually.	[H]
	Lq	parametersLq	Stator quadrature inductance. Obtain from motor manufacturer or use the electrical parameters identification and then fill manually.	[H]
	Kt	parametersKt	Motor torque constant. Obtain from motor manufacturer or use the Kt identification and then fill manually.	[Nm/A]
	J	parametersJ	Drive inertia (motor + plant). Use the mechanical identification and then fill manually.	[kg.m2]
	Iph nom	parametersIphNom	Nominal motor current. Obtain from motor manufacturer.	[A]

Table 12. Parameters tab inputs*...continued*

MCAT group	MCAT name	Equation name	Description	Unit
	Uph nom	parametersUphNom	Nominal motor voltage. Obtain from motor manufacturer.	[V]
	N nom	parametersNnom	Nominal motor speed. Obtain from motor manufacturer.	[rpm]
Hardware scales	I max	parametersImax	Current sensing HW scale. Keep as-is in case of standard NXP HW or recalculate according to own schematic.	[A]
	U DCB max	parametersUdcbMax	DCBus voltage sensing HW scale. Keep as-is in case of standard NXP HW or recalculate according to own schematic.	[V]
Fault limits	U DCB trip	parametersUdcbTrip	DCBus braking resistor threshold. Braking resistor's transitor is turned on when DCbus voltage exceeds this threshold.	[V]
	U DCB under	parametersUdcbUnder	DCBus under voltage fault threshold	[V]
	U DCB over	parametersUdcbOver	DCBus over voltage fault threshold	[V]
	N over	parametersNover	Over speed fault threshold	[rpm]
	N min	parametersNmin	Minimal closed loop speed. When the required speed ramps down under this threshold, the motor control state machine goes to freewheel state where top and bottom transistors are turned off and motor speeds down freely. Applies only for sensorless operation.	[rpm]
	E block	parametersEblock	Blocked rotor detection. When BEMF voltage drops under *E block* threshold for more than *E block per* (fast loop ticks), the blocked rotor fault is detected.	[V]
	E block per	parametersEblockPer		-
Application scales	N max	parametersNmax	Application speed scale. Keep about 10 % margin above *N over*.	[rpm]
	U DCB IIR F0	parametersUdcbIIRf0	Cut-off frequency of DCBus IIR filter	[Hz]
	Calibration duration	parametersCalibDuration	ADC (phase current offset) calibration duration. Done every time transitioning from STOP to RUN.	[sec]
	Fault duration	parametersFaultDuration	After fault condition disappears, wait defined time to clear pending faults bitfield and transition to STOP state.	[sec]

Table 12. Parameters tab inputs...*continued*

MCAT group	MCAT name	Equation name	Description	Unit
	Freewheel duration	parametersFreewheelDuration	Free-wheel state duration. Freewheel state in entered when ramped speed drops under *N min*.	[sec]
	Scalar Uq min	parametersScalarUqMin	Scalar control voltage minimal value.	[V]
	Scalar V/Hz factor ratio	parametersScalarVHzRatio	Scalar V/Hz ratio gain.	[%]
Alignment	Align voltage	parametersAlignVoltage	Motor alignment voltage.	[V]
	Align duration	parametersAlignDuration	Motor alignment duration.	[sec]

Output equations (applies for saving to `mX_pmsm_appconfig.h` and also for updating a corresponding FreeMASTER variable):

- `M1_U_MAX` = parametersUdcbMax / UmaxCoeff
- `M1_MOTOR_PP` = parametersPp
- `M1_I_PH_NOM` = parametersIphNom
- `M1_I_MAX` = parametersImax
- `M1_U_DCB_MAX` = parametersUdcbMax
- `M1_U_DCB_TRIP` = parametersUdcbTrip
- `M1_U_DCB_UNDERVOLTAGE` = parametersUdcbUnder
- `M1_U_DCB_OVERVOLTAGE` = parametersUdcbOver
- `M1_FREQ_MAX` = parametersNmax / 60 * parametersPp
- `M1_ALIGN_DURATION` = parametersAlignDuration / speedLoopSampleTime
- `M1_CALIB_DURATION` = parametersCalibDuration / speedLoopSampleTime
- `M1_FAULT_DURATION` = parametersFaultDuration / speedLoopSampleTime
- `M1_FREEWHEEL_DURATION` = parametersFreewheelDuration / speedLoopSampleTime
- `M1_E_BLOCK_TRH` = parametersEblock
- `M1_E_BLOCK_PER` = parametersEblockPer
- `M1_N_MIN` = parametersNmin / 60 * (parametersPp * 2 * pi)
- `M1_N_MAX` = parametersNmax / 60 * (parametersPp * 2 * pi)
- `M1_N_ANGULAR_MAX` = (60 / (parametersPp * 2 * pi))
- `M1_N_NOM` = parametersNnom / 60 * (parametersPp * 2 * pi)
- `M1_N_OVERSPEED` = parametersNover / 60 * (parametersPp * 2 * pi)
- `M1_UDCB_IIR_B0` = (2 * pi * parametersUdcbIIRf0 * currentLoopSampleTime) / (2 + (2 * pi * parametersUdcbIIRf0 * currentLoopSampleTime))
- `M1_UDCB_IIR_B1` = (2 * pi * parametersUdcbIIRf0 * currentLoopSampleTime) / (2 + (2 * pi * parametersUdcbIIRf0 * currentLoopSampleTime))
- `M1_UDCB_IIR_A1` = -(2 * pi * parametersUdcbIIRf0 * currentLoopSampleTime - 2) / (2 + (2 * pi * parametersUdcbIIRf0 * currentLoopSampleTime))
- `M1_SCALAR_UQ_MIN` = parametersScalarUqMin
- `M1_ALIGN_VOLTAGE` = parametersAlignVoltage
- `M1_SCALAR_VHZ_FACTOR_GAIN` = parametersUphNom * parametersScalarVHzRatio / 100 / (parametersNnom * parametersPp / 60)
- `M1_SCALAR_INTEG_GAIN` = 2*pi*parametersPp*parametersNmax/60*currentLoopSampleTime/pi
- `M1_SCALAR_RAMP_UP` = speedLoopIncUp*currentLoopSampleTime/60*parametersPp
- `M1_SCALAR_RAMP_DOWN` = speedLoopIncDown*currentLoopSampleTime/60*parametersPp

8.3.1.3 Current loop

This tab enables current loop PI controller gains and output limits tuning. All inputs are described in the following table. MCAT group and MCAT name help to locate the parameter in MCAT layout. Equation name represents the input parameter in equations bellow.

Table 13. Current loop tab input

MCAT group	MCAT name	Equation name	Description	Unit
Loop parameters	Sample time	currentLoopSampleTime	Fast control loop period. This disabled value is read from target via FreeMASTER because application timing is set in embedded code by peripherals setting. This value is accessible only if target is not connected and value cannot be obtained from target.	[sec]
	F0	currentLoopF0	Current controller's bandwidth	[Hz]
	ξ	currentLoopKsi	Current controller's attenuation	-
Current PI controller limits	Output limit	currentLoopOutputLimit	Current controllers' output voltage limit = Duty cycle limit. Be careful setting this limit above 95 % because it affects current sensing (Some minimal bottom transistors on time is required).	[%]

Output equations (applies for saving to `mX_pmsm_appconfig.h` and also for updating a corresponding FreeMASTER variable):

- `M1_CLOOP_LIMIT` = currentLoopOutputLimit / UmaxCoeff / 100
- `M1_D_KP_GAIN` = (2 * currentLoopKsi * 2 * pi * currentLoopF0 * parametersLd) - parametersRs
- `M1_D_KI_GAIN` = (2 * pi * currentLoopF0)^2 * parametersLd * currentLoopSampleTime / DiscMethodFactor
- `M1_Q_KP_GAIN` = (2 * currentLoopKsi * 2 * pi * currentLoopF0 * parametersLq) - parametersRs
- `M1_Q_KI_GAIN` = (2 * pi * currentLoopF0)^2 * parametersLq * currentLoopSampleTime / DiscMethodFactor
- kp_q = (4*currentLoopKsi*pi*currentLoopF0*parametersLq-parametersRs)
- ki_q = (parametersLq*(2*pi*currentLoopF0)^2)
- Ti = kp_q/ki_q
- `M1_Q_IIR_ZC_B0`= currentLoopSampleTime/(currentLoopSampleTime+2*Ti)
- `M1_Q_IIR_ZC_B1`= currentLoopSampleTime/(currentLoopSampleTime+2*Ti)
- `M1_Q_IIR_ZC_A1`= -(currentLoopSampleTime-2*Ti)/(currentLoopSampleTime+2*Ti)

8.3.1.4 Speed loop

This tab enables speed loop PI controller gains and output limits tuning, required speed ramp parameters and feedback speed filter tuning. MCAT group and MCAT name help to locate the parameter in MCAT layout. Equation name represents the input parameter in equations bellow.

Table 14. Speed loop tab input

MCAT group	MCAT name	Equation name	Description	Unit
Loop parameters	Sample time Ts	speedLoopSampleTime	Slow control loop period. This disabled value is read from target via FreeMASTER because	[sec]

Table 14. Speed loop tab input*...continued*

MCAT group	MCAT name	Equation name	Description	Unit
			application timing is set in embedded code by peripherals setting. This value is accessible only if target is not connected and value cannot be obtained from target.	
	Bandwidth F0	speedLoopF0	Speed controller's bandwidth	[Hz]
	Attenuation ξ	speedLoopKsi	Speed controller's attenuation	-
Speed ramp	Inc up	speedLoopIncUp	Required speed maximal acceleration	[rpm/sec]
	Inc down	speedLoopIncDown	Required speed maximal acceleration	[rpm/sec]
Actual speed filter	Cut-off freq	speedLoopCutOffFreq	Speed feedback (before entering PI subtraction) filter bandwidth.	[Hz]
Speed PI controller limits	Upper limit	speedLoopUpperLimit	Maximal required Q-axis current (Speed controller's output). Q-axis current limitation equals to motor torque limitation.	[A]
	Lower limit	speedLoopLowerLimit	Minimal required Q-axis current (Speed controller's output). Q-axis current limitation equals to motor torque limitation.	[A]

Output equations (applies for saving to `mX_pmsm_appconfig.h` and also for updating a corresponding FreeMASTER variable):

- `M1_SPEED_PI_PROP_GAIN` = (4 * speedLoopKsi * pi * speedLoopF0 * parametersJ) / (parametersKt * parametersPp))
- `M1_SPEED_PI_INTEG_GAIN` = ((2 * pi * speedLoopF0) * (2 * pi * speedLoopF0) * parametersJ) / (parametersKt * parametersPp) * (speedLoopSampleTime / DiscMethodFactor))
- `M1_SPEED_RAMP_UP` = (speedLoopIncUp * speedLoopSampleTime / (60 / (parametersPp * 2 * pi)))
- `M1_SPEED_RAMP_DOWN` = (speedLoopIncDown * speedLoopSampleTime / (60 / (parametersPp * 2 * pi)))
- `M1_SPEED_IIR_B0`= (2 * pi * speedLoopCutOffFreq * speedLoopSampleTime) / (2 + (2 * pi * speedLoopCutOffFreq * speedLoopSampleTime))
- `M1_SPEED_IIR_B1` = (2 * pi * speedLoopCutOffFreq * speedLoopSampleTime) / (2 + (2 * pi * speedLoopCutOffFreq * speedLoopSampleTime))
- `M1_SPEED_IIR_A1` = -(2 * pi * speedLoopCutOffFreq * speedLoopSampleTime - 2) / (2 + (2 * pi * speedLoopCutOffFreq * speedLoopSampleTime))
- `M1_SPEED_LOOP_HIGH_LIMIT` = speedLoopUpperLimit
- `M1_SPEED_LOOP_LOW_LIMIT` = speedLoopLowerLimit
- Tfilt = `M1_SPEED_PI_PROP_GAIN`/(`M1_SPEED_PI_INTEG_GAIN` * (2/speedLoopSampleTime))
- `M1_SPEED_IIR_ZC_B0`=(speedLoopSampleTime / (speedLoopSampleTime + 2 * Tfilt))
- `M1_SPEED_IIR_ZC_B1`=(speedLoopSampleTime / (speedLoopSampleTime + 2 * Tfilt))
- `M1_SPEED_IIR_ZC_A1`=(-1)*(speedLoopSampleTime - 2 * Tfilt) / (speedLoopSampleTime + 2 * Tfilt)

8.3.1.5 Servo control

This tab enables position loop P/PI controller gains and output limits tuning. MCAT group and MCAT name help to locate the parameter in MCAT layout. Equation name represents the input parameter in equations bellow.

Table 15. Servo control tab input

MCAT group	MCAT name	Equation name	Description	Unit
Servo control parameters	Sample time Ts	positionLoopSampleTime	Slow control loop period. This disabled value is read from target via FreeMASTER because application timing is set in embedded code by peripherals setting. This value is accessible only if target is not connected and value cannot be obtained from target.	[sec]
	Bandwidth F0	positionLoopF0	Servo controller's bandwidth	[Hz]
	Attenuation ξ	positionLoopKsi	Servo controller's attenuation	-
Position loop - P controller limits	Upper limit	positionPControllerHighLimit	Maximal required speed (Position controller's output)	[rpm/sec]
	Lower limit	positionPControllerLowLimit	Minimal required speed (Position controller's output)	[rpm/sec]
Speed loop - PI controller limits	Upper limit	servo_speedLoopUpperLimit	Maximal required Q-axis current (Speed controller's output). Q-axis current limitation equals to motor torque limitation.	[A]
	Lower limit	servo_speedLoopLowerLimit	Minimal required Q-axis current (Speed controller's output). Q-axis current limitation equals to motor torque limitation.	[A]

Output equations (applies for saving to `mX_pmsm_appconfig.h` and also for updating a corresponding FreeMASTER variable):

- k10 = 1/(2 * pi * parametersPp)
- `M1_SERVO_POSITION_P_PROP_GAIN`= (2 * pi * positionLoopF0)/(2 * positionLoopKsi + 1) * k10
- `M1_SERVO_POSITION_P_HIGH_LIMIT`=(2 * pi * positionPControllerHighLimit * parametersPp / 60.0)
- `M1_SERVO_POSITION_P_LOW_LIMIT`=(2 * pi * positionPControllerLowLimit * parametersPp / 60.0)
- `M1_SERVO_FEED_FRWD_K1`= (2 * positionLoopKsi * `M1_SERVO_POSITION_P_PROP_GAIN` / (2 * pi * positionLoopF0))
- `M1_SERVO_FEED_FRWD_K2`= (`M1_SERVO_POSITION_P_PROP_GAIN` / ((2 * pi * positionLoopF0) * (2 * pi * positionLoopF0)))
- Tfilt = (2 * positionLoopKsi + 1) / ((2 * positionLoopKsi + 1) * 2 * pi * positionLoopF0)
- `M1_SERVO_IIR_ZC_B0` = (positionLoopSampleTime/(positionLoopSampleTime + 2 * Tfilt))
- `M1_SERVO_IIR_ZC_B1`= (positionLoopSampleTime/(positionLoopSampleTime + 2 * Tfilt))
- `M1_SERVO_IIR_ZC_A1`= (-1) * (positionLoopSampleTime - 2 * Tfilt) / (positionLoopSampleTime + 2 * Tfilt)
- `M1_SERVO_SPEED_PI_PROP_GAIN` = (((2 * positionLoopKsi + 1) * (2 * pi * positionLoopF0) * parametersJ) / (parametersKt * parametersPp))
- `M1_SERVO_SPEED_PI_INTEG_GAIN` = (((parametersJ * (2 * positionLoopKsi + 1) * (2 * pi * positionLoopF0) * (2 * pi * positionLoopF0)) / (parametersKt * parametersPp)) * (positionLoopSampleTime / DiscMethodFactor)

- `M1_SERVO_SPEED_PI_HIGH_LIMIT` = servo_speedLoopUpperLimit
- `M1_SERVO_SPEED_PI_LOW_LIMIT` = servo_speedLoopLowerLimit

8.3.1.6 Sensorless

This tab enables BEMF observer and Tracking observer parameters tuning and open-loop startup tuning. MCAT group and MCAT name help to locate the parameter in MCAT layout. Equation name represents the input parameter in equations bellow.

Table 16. Sensorless tab input

MCAT group	MCAT name	Equation name	Description	Unit
BEMF observer parameters	F0	sensorlessBemfObsrvF0	BEMF observer bandwidth	[Hz]
	ξ	sensorlessBemfObsrvKsi	BEMF observer attenuation	-
Tracking observer parameters	F0	sensorlessTrackObsrvF0	Tracking observer bandwidth	[Hz]
	ξ	sensorlessTrackObsrvKsi	Tracking observer attenuation	-
Open loop startup parameters	Startup ramp	sensorlessStartupRamp	Open loop startup ramp	[rpm/sec]
	Startup current	sensorlessStartupCurrent	Open loop startup current	[A]
	Merging Speed	sensorlessMergingSpeed	Merging speed	[rpm]
	Merging Coefficient	sensorlessMergingCoeff	Merging coefficient (100 % = merging is done within one electrical revolution)	[%]

Output equations (applies for saving to `mX_pmsm_appconfig.h` and also for updating a corresponding FreeMASTER variable):

- `M1_I_SCALE` = (parametersLd / (parametersLd + currentLoopSampleTime * parametersRs))
- `M1_U_SCALE` = (currentLoopSampleTime / (parametersLd + currentLoopSampleTime * parametersRs))
- `M1_E_SCALE` = (currentLoopSampleTime / (parametersLd + currentLoopSampleTime * parametersRs))
- `M1_WI_SCALE` = (parametersLq * currentLoopSampleTime / (parametersLd + currentLoopSampleTime * parametersRs))
- `M1_BEMF_DQ_KP_GAIN` = ((2 * sensorlessBemfObsrvKsi * 2 * pi * sensorlessBemfObsrvF0 * parametersLd - parametersRs))
- `M1_BEMF_DQ_KI_GAIN` = (parametersLd * (2 * pi * sensorlessBemfObsrvF0)^ 2 * currentLoopSampleTime)
- `M1_TO_KP_GAIN` = 2 * sensorlessTrackObsrvKsi * 2 * pi * sensorlessTrackObsrvF0
- `M1_TO_KI_GAIN` = ((2 * pi * sensorlessTrackObsrvF0)^ 2) * currentLoopSampleTime
- `M1_TO_THETA_GAIN` = (currentLoopSampleTime / pi)
- `M1_OL_START_RAMP_INC` = (sensorlessStartupRamp * currentLoopSampleTime / (60 / (parametersPp * 2 * pi)))
- `M1_OL_START_I` = sensorlessStartupCurrent
- `M1_MERG_SPEED_TRH` = (sensorlessMergingSpeed / (60 / (parametersPp * 2 * pi)))
- `M1_MERG_COEFF` = ((sensorlessMergingCoeff / 100) * sensorlessMergingSpeed * parametersPp * currentLoopSampleTime) / 60
- `TO_IIR_cutoff_freq` = 1 / (2 * speedLoopSampleTime) * 0.8
- `M1_TO_SPEED_IIR_B0` = (2 * pi * TO_IIR_cutoff_freq * currentLoopSampleTime) / (2 + (2 * pi * TO_IIR_cutoff_freq * currentLoopSampleTime))
- `M1_TO_SPEED_IIR_B1` = (2 * pi * TO_IIR_cutoff_freq * currentLoopSampleTime) / (2 + (2 * pi * TO_IIR_cutoff_freq * currentLoopSampleTime))

- `M1_TO_SPEED_IIR_A1` = -(2 * pi * TO_IIR_cutoff_freq * currentLoopSampleTime - 2) / (2 + (2 * pi * TO_IIR_cutoff_freq * currentLoopSampleTime))

8.4 Motor Control Modes - How to run motor

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

Before motor starts, several conditios have to be completed:

1. Connected power supply to the inverter with the correct voltage value.
2. If you want to use sensored control (encoder feedback), connect the encoder to the inverter.
3. No pending fault. Check variable "M1 Fault Pending" in "Motor M1" project tree subblock. If there is some value, first remove the cause of the fault, or disable fault checking. (for example in variable "M1 Fault Enable Blocked Rotor")

8.4.1 Scalar control

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

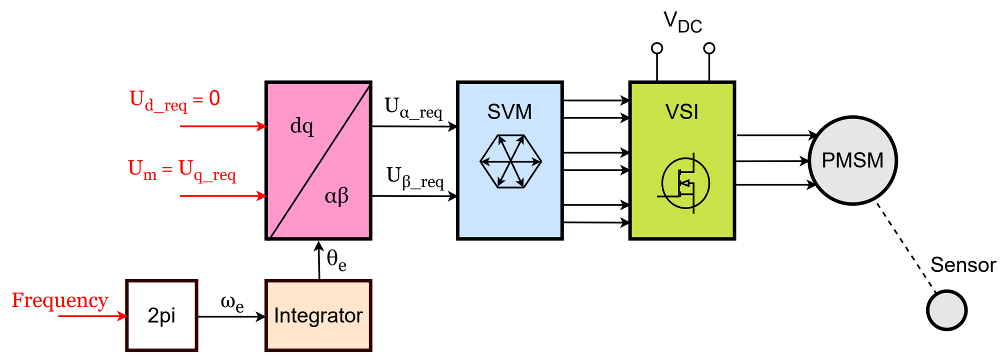


Figure 14. Scalar control mode

For run motor in scalar control, follow these steps:

1. Switch project tree subblock on "Scalar & Voltage Control".
2. Switch variable "M1 MCAT Control" on "SCALAR_CONTROL".
3. In variable "M1 Scalar Freq Required" set required frequency. (i.e. 20Hz)
4. Set variable "M1 Application Switch" to "1". Motor start spinning.
5. Observe motor speed, position, phase currents and other graphs predefined in subblock scopes and recorders.

8.4.2 Open loop control mode

Open loop mode (its diagram is shown in figure below) is similar in function to the Scalar control mode. However, it provides more flexibility in specifying required parameters. This mode allows you to set specific angle and frequency, according to the following equation:

$$\theta_{el} = \theta_{init} + \int_{t_0}^{t} 2\pi f \, dt \tag{4}$$

Besides setting voltage in DQ axis, when using this mode you can also enable current controllers and specify required currents in D and Q axis. Therefore, this function can be utilized for current controller parameter tuning. Please, bear in mind that current controllers cannot be enabled/disabled in SPIN state (user must turn the Application Switch OFF before enabling/disabling current controllers).

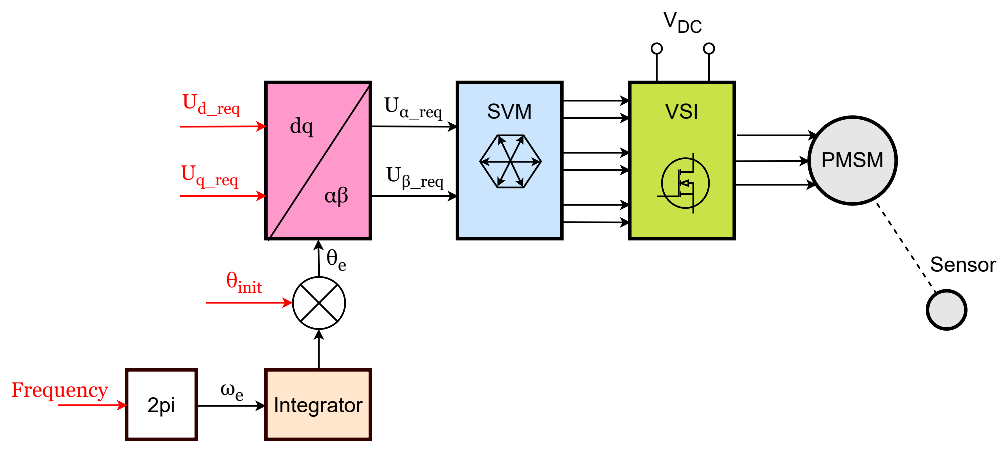


Figure 15. Voltage - Open loop control

For run motor in Voltage - Open loop control, follow these steps:

1. Switch project tree subblock on "Openloop Control".
2. Switch variable "M1 MCAT Control" on "OPEN_LOOP".
3. In variable "M1 Openloop Required Ud" and "M1 Openloop Required Uq" set required values.
4. In variable "M1 Openloop Theta Electrical" set required initial position.
5. In variable "M1 Openloop Required Frequency Electrical" set required frequency.
6. Set variable "M1 Application Switch" to "1". Motor start spinning.
7. Observe motor speed, position, phase currents and other graphs predefined in subblock scopes and recorders.

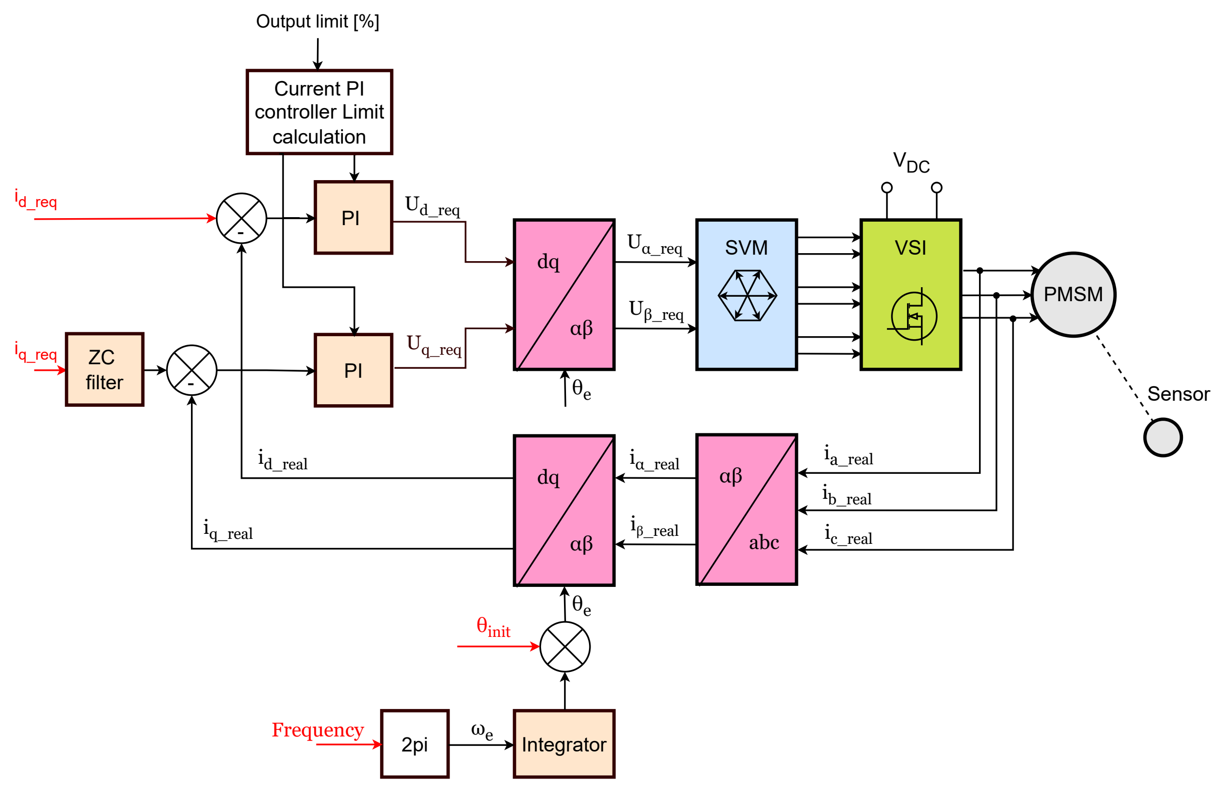


Figure 16. Current - Open loop control

For run motor in Current - Open loop control, follow these steps:

1. Switch project tree subblock on "Openloop Control".
2. Switch variable "M1 MCAT Control" on "OPEN_LOOP".
3. Set variable "M1 Openloop Use I Control" to "1".
4. In variable "M1 Openloop Required Id" and "M1 Openloop Required Iq" set required values.
5. In variable "M1 Openloop Theta Electrical" set required initial position.
6. In variable "M1 Openloop Required Frequency Electrical" set required frequency.
7. Set variable "M1 Application Switch" to "1". Motor start spinning.
8. Observe motor speed, position, phase currents and other graphs predefined in subblock scopes and recorders.

8.4.3 Voltage control

The block diagram of the voltage FOC is shown in Figure 17. Unlike the scalar control, the position feedback is closed using the BEMF observer and the stator voltage magnitude is not dependent on the motor speed. Both the d-axis and q-axis stator voltages can be specified in the "M1 MCAT Ud Required" and "M1 MCAT Uq Required" fields. This control method is useful for the BEMF observer functionality check.

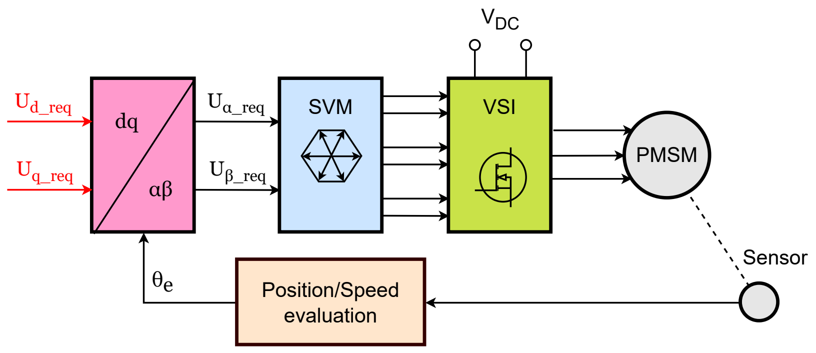


Figure 17. Voltage FOC control mode

For run motor in voltage control, follow these steps:

1. Switch project tree subblock on "Scalar & Voltage Control".
2. Switch variable "M1 MCAT Control" on "VOLTAGE_FOC".
3. In variable "M1 MCAT Uq Required" and "M1 MCAT Ud Required" set required voltages.
4. Set variable "M1 Application Switch" to "1". Motor start spinning.
5. Observe motor speed, position, phase currents and other graphs predefined in subblock scopes and recorders.

8.4.4 Current/Torque control

The current FOC (or torque) control requires the rotor position feedback and the currents transformed into a d-q reference frame. There are two reference variables ("M1 MCAT Id Required" and "M1 MCAT Iq Required") available for the motor control, as shown in Figure 18. The d-axis current component "M1 MCAT Id Required" controls the rotor flux. The q-axis current component of the current "M1 MCAT Iq Required" generates torque and, by its application, the motor starts running. By changing the polarity of the current "M1 MCAT Iq Required", the motor changes the direction of rotation. Supposing the BEMF observer is tuned correctly, the current PI controllers can be tuned using the current FOC control structure.

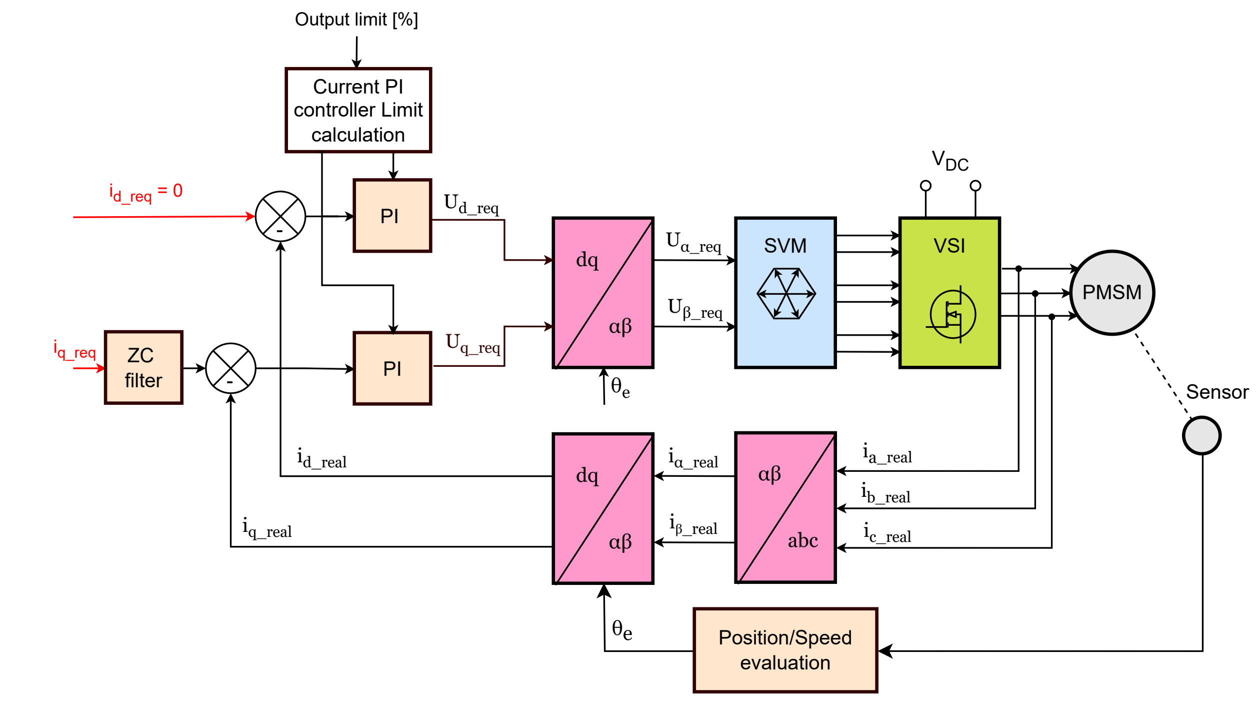


Figure 18. Current/Torque control mode (sensored)

For run motor in current control, follow these steps:

1. Switch project tree subblock on "Current Control".
2. Switch variable "M1 MCAT Control" on "CURRENT_FOC".
3. In variable "M1 MCAT Iq Required" and "M1 MCAT Id Required" set required currents.
4. Set variable "M1 Application Switch" to "1". Motor start spinning.
5. Observe motor speed, position, phase currents and other graphs predefined in subblock scopes and recorders.

8.4.5 Speed FOC control

As shown in Figure 19, the speed PMSM sensor/sensorless FOC is activated by enabling the speed FOC control structure. Enter the required speed into the "M1 Speed Required" field. The d-axis current reference is held at 0 during the entire FOC operation.

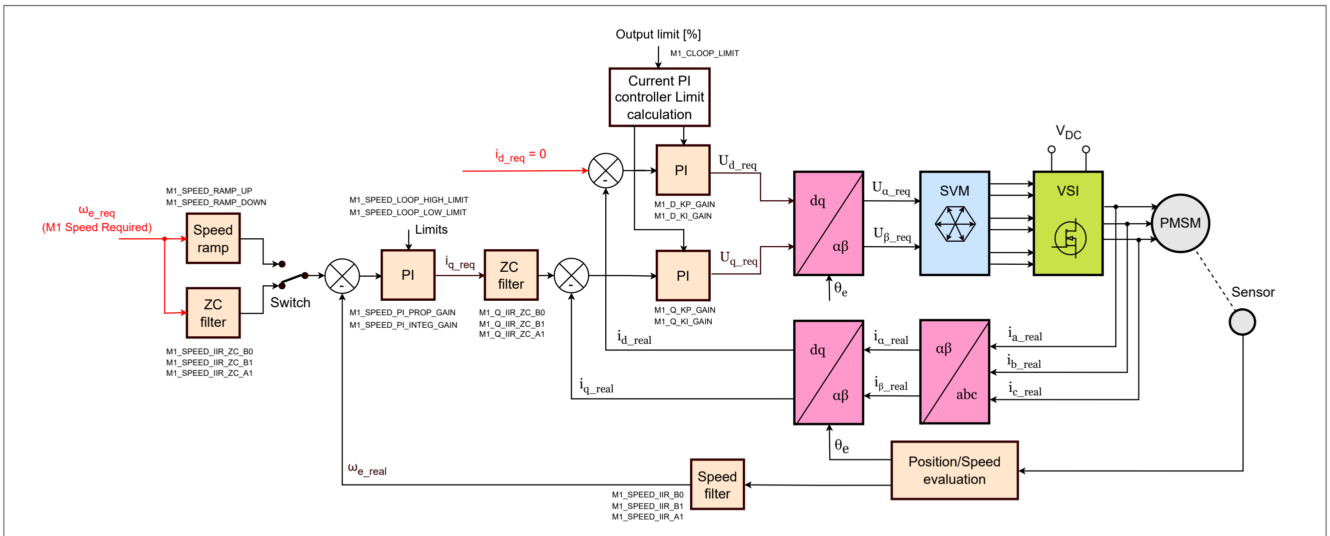


Figure 19. Speed FOC control mode (sensored)

For run motor in speed FOC control, follow these steps:

1. Switch project tree subblock on "Speed Control".
2. Switch variable "M1 MCAT Control" on "SPEED_FOC".
3. Choose between sensored and sensorless control in variable "M1 MCAT POSPE Sensor".
4. In variable "M1 Speed Required" set the required speed. (i.e. 1000rpm). The motor automatically starts spinning.
5. Observe motor speed, position, phase currents and other graphs predefined in subblock scopes and recorders.

8.4.6 Position (servo) control

The position of PMSM sensor FOC is shown in Figure 20 (available for sensored/encoder based applications only). The position control using the P controller can be tuned in the "Servo control" menu tab. An encoder sensor is required for the feedback. Without the sensor, the position control does not work. A braking resistor is missing on the FRDM-MC-LVPMSM board. Therefore, it is necessary to set a soft parameters of P controller because the voltage on the DC-bus can rise when braking the quickly spinning shaft. It may cause an overvoltage fault.

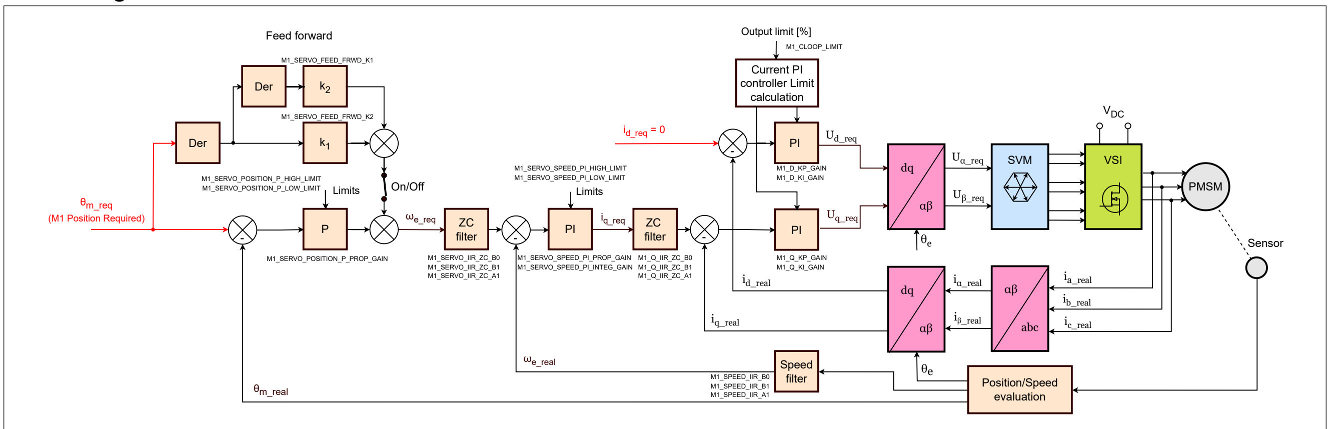


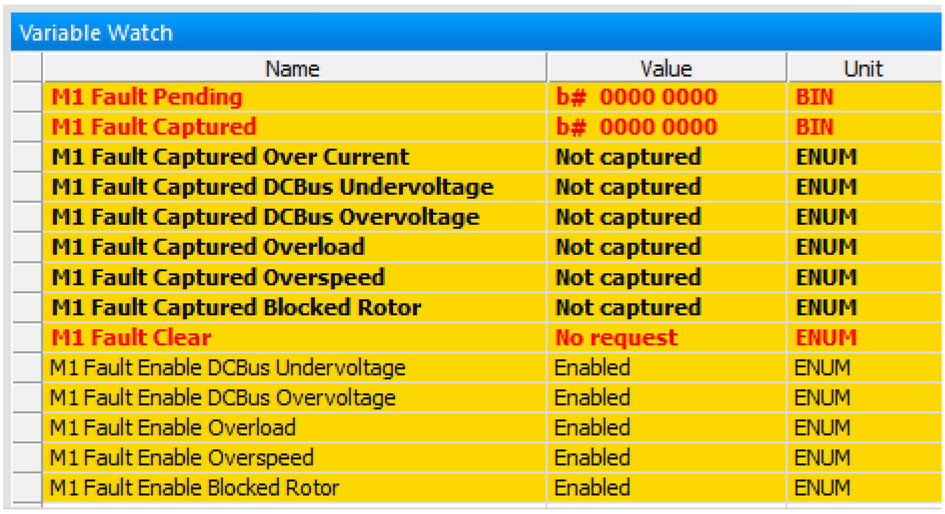
Figure 20. Position control mode

For run motor in position (servo) control, follow these steps:

1. Switch project tree subblock on "Position Control".
2. Switch variable "M1 MCAT Control" on "POSITION_CNTRL".
3. Swich variable "M1 MCAT POSPE Sensor" to "Encoder [1]".
4. In variable "M1 Position Required" set the required psition. (i.e. 10 revs).
5. Set variable "M1 Application Switch" to "1". The motor starts and automatically stops in the required position.
6. Change "M1 Encoder Direction" if the motor does not spin. (See chapter Section 8.7.1)
7. Observe motor speed, position, phase currents and other graphs predefined in subblock scopes and recorders.

8.5 Faults explanation

When the motor is running or during the tuning process, there may be several fault conditions. Therefore, the motor-control example has an integrated fault indication located in the variable watch of the "Motor M1" FreeMASTER subblock. If a fault is indicated, state machine enters the FAULT state.



Variable Watch

Name	Value	Unit
M1 Fault Pending	b# 0000 0000	BIN
M1 Fault Captured	b# 0000 0000	BIN
M1 Fault Captured Over Current	Not captured	ENUM
M1 Fault Captured DCBus Undervoltage	Not captured	ENUM
M1 Fault Captured DCBus Overvoltage	Not captured	ENUM
M1 Fault Captured Overload	Not captured	ENUM
M1 Fault Captured Overspeed	Not captured	ENUM
M1 Fault Captured Blocked Rotor	Not captured	ENUM
M1 Fault Clear	No request	ENUM
M1 Fault Enable DCBus Undervoltage	Enabled	ENUM
M1 Fault Enable DCBus Overvoltage	Enabled	ENUM
M1 Fault Enable Overload	Enabled	ENUM
M1 Fault Enable Overspeed	Enabled	ENUM
M1 Fault Enable Blocked Rotor	Enabled	ENUM

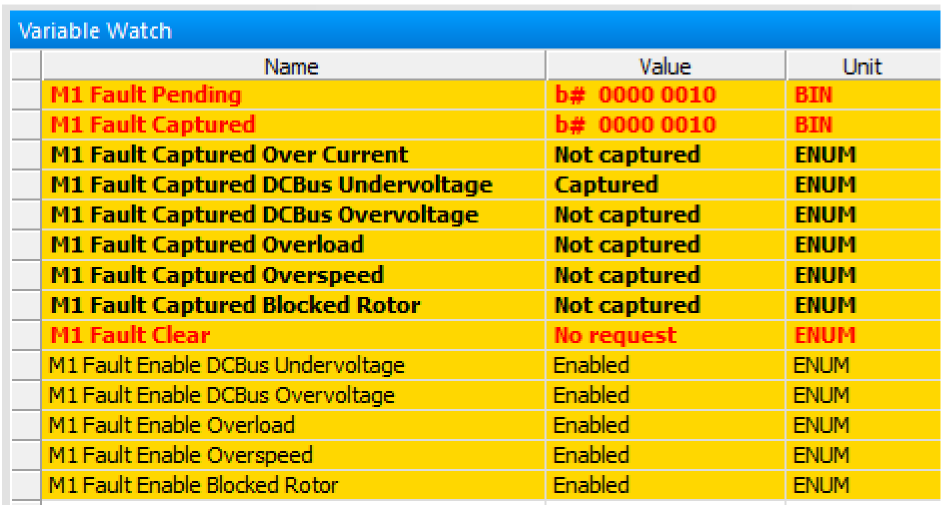
Figure 21. Faults in variable watch located in "Motor M1" subblock

8.5.1 Variable "M1 Fault Pending"

It shows actually persisting faults, which means that the fault indicated during fault conditions is accomplished. For example, if the source voltage is still under the undervoltage fault threshold, the undervoltage pending fault is shown. If the fault condition disappears, the fault pending is cleared automatically. "M1 Fault Pending" is shown in a binary format in the FreeMASTER variable watch. Each place in the variable denotes a different fault condition.

- b 0000 0001 - the overcurrent fault is indicated. If the overcurrent fault is present, the PWMs are automatically disabled. The fault occurs when the DC-Bus current exceeds the **Imax** value (current-sensing HW scale).
- b 0000 0010 - the undervoltage fault is indicated. The undervoltage fault occurs when the UDCBus voltage (source voltage) is lower than the **U DCB under** threshold.
- b 0000 0100 - the overvoltage fault is indicated. The overvoltage fault occurs when the UDCBus voltage (source voltage) is higher than the **U DCB over** threshold.
- b 0000 1000 - the overload fault is indicated. The overload fault occurs when the rotor is overloaded.
- b 0001 0000 - the overspeed fault is indicated. The overspeed fault occurs when the rotor speed exceeds the **N over** threshold.

- b 0010 0000 - the block rotor fault is indicated. The block rotor fault occurs when the back-EMF voltage is lower than the **E block** threshold and the duration of the drop is longer than **E block per**.



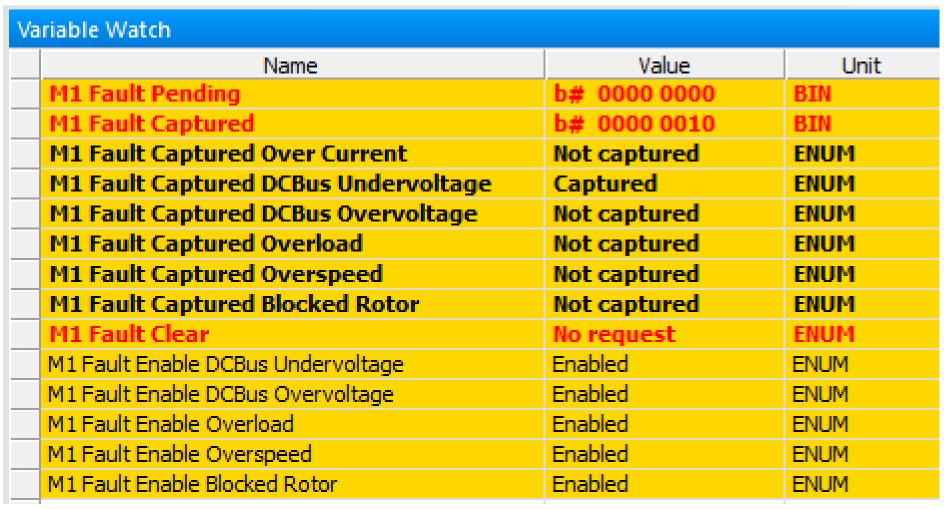
Variable Watch

Name	Value	Unit
M1 Fault Pending	b# 0000 0010	BIN
M1 Fault Captured	b# 0000 0010	BIN
M1 Fault Captured Over Current	Not captured	ENUM
M1 Fault Captured DCBus Undervoltage	Captured	ENUM
M1 Fault Captured DCBus Overvoltage	Not captured	ENUM
M1 Fault Captured Overload	Not captured	ENUM
M1 Fault Captured Overspeed	Not captured	ENUM
M1 Fault Captured Blocked Rotor	Not captured	ENUM
M1 Fault Clear	No request	ENUM
M1 Fault Enable DCBus Undervoltage	Enabled	ENUM
M1 Fault Enable DCBus Overvoltage	Enabled	ENUM
M1 Fault Enable Overload	Enabled	ENUM
M1 Fault Enable Overspeed	Enabled	ENUM
M1 Fault Enable Blocked Rotor	Enabled	ENUM

Figure 22. Undervoltage fault is indicated (pending)

8.5.2 Variable "M1 Fault Captured"

If any fault condition appears, the fault captured is indicated. Similar to fault pending, fault captured is shown in the BIN format, but every fault type has its own variable ("M1 Fault Captured Over Curent" and others). For example, if the undervoltage fault condition is accomplished, fault captured is indicated. Fault captured is also indicated after the undervoltage fault condition disappears. The captured faults are cleared manually by writing "Clear [1]" to "M1 Fault Clear".



Variable Watch

Name	Value	Unit
M1 Fault Pending	b# 0000 0000	BIN
M1 Fault Captured	b# 0000 0010	BIN
M1 Fault Captured Over Current	Not captured	ENUM
M1 Fault Captured DCBus Undervoltage	Captured	ENUM
M1 Fault Captured DCBus Overvoltage	Not captured	ENUM
M1 Fault Captured Overload	Not captured	ENUM
M1 Fault Captured Overspeed	Not captured	ENUM
M1 Fault Captured Blocked Rotor	Not captured	ENUM
M1 Fault Clear	No request	ENUM
M1 Fault Enable DCBus Undervoltage	Enabled	ENUM
M1 Fault Enable DCBus Overvoltage	Enabled	ENUM
M1 Fault Enable Overload	Enabled	ENUM
M1 Fault Enable Overspeed	Enabled	ENUM
M1 Fault Enable Blocked Rotor	Enabled	ENUM

Figure 23. Undervoltage fault is captured

8.5.3 Variable "M1 Fault Enable"

The fault indication can be unwanted during the tuning process. Therefore, the fault indication can be disabled by writing "Disabled [0]" to the "M1 Fault Enable" variables.

Note: *The overcurrent fault cannot be disabled.*

Note: *Fault thresholds are located in the "MCAT parameters" tab.*

8.6 Initial motor parameters and harware configuration

Motor control examples contain two or more configuration files: `m1_pmsm_appconfig.h`, `m2_pmsm_appconfig.h`, and so on. Each contains constants tuned for the selected motor (see supported motors in Section 2). The initial motor parameters and the hardware configuration (inverter) are to MCAT loaded from `m1_pmsm_appconfig.h` configuration file. There are tree ways to change motor configuration corresponding to the connected motor.

1. The first way is rename the configuration file:
   - In the project example folder, find configuration file to be used.
   - Rename this configuration file to `m1_pmsm_appconfig.h`.
   - Rebuild project and load the code to the MCU.
2. The second way is to change motor configuration, as described in Section 8.3.
3. The last way is change motor and hardware parameters manually:
   - Open the PMSM control application FreeMASTER project containing the dedicated MCAT plug-in module.
   - Select the "Parameters" tab.
   - 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 17 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 17. MCAT motor parameters

Parameter	Units	Description	Typical range
pp	[-]	Motor pole pairs	1-10
Rs	[Ω]	1-phase stator resistance	0.3-50
Ld	[H]	1-phase direct inductance	0.00001-0.1
Lq	[H]	1-phase quadrature inductance	0.00001-0.1
Ke	[V.sec/rad]	BEMF constant	0.001-1
J	[kg.m^2]	System inertia	0.00001-0.1
Iph nom	[A]	Motor nominal phase current	0.5-8
Uph nom	[V]	Motor nominal phase voltage	10-300
N nom	[rpm]	Motor nominal speed	1000-2000

   - 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.
   - Check the fault limits—these fields are calculated using the motor parameters and hardware scales (see Table 18).

Table 18. 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

- Check the application scales—these fields are calculated using the motor parameters and hardware scales (see Table 19).

Table 19. Application scales

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

- Check the alignment parameters—these fields are calculated using the motor parameters and hardware scales. The parameters express the required voltage value applied to the motor during the rotor alignment and its duration.
- To save the modified parameters into the inner file, click the "Store data" button.

8.7 Control parameters tuning

To check correct current measuring and proper working of back EMF observer, follow the steps below:

1. Select the scalar control in the "M1 MCAT Control" FreeMASTER variable watch.
2. Set the "M1 Application Switch" variable to "ON". The application state changes to "RUN".
3. Set the required frequency value in the "M1 Scalar Freq Required" variable; for example, 15 Hz in the "Scalar & Voltage Control" FreeMASTER project tree. The motor starts running.
4. Select the "Phase Currents" recorder from the "Scalar & Voltage Control" FreeMASTER project tree.
5. The optimal ratio for the V/Hz profile can be found by changing the V/Hz factor directly using MCAT input "Scalar V/Hz factor ratio". The shape of the motor currents should be close to a sinusoidal shape (Figure 24).

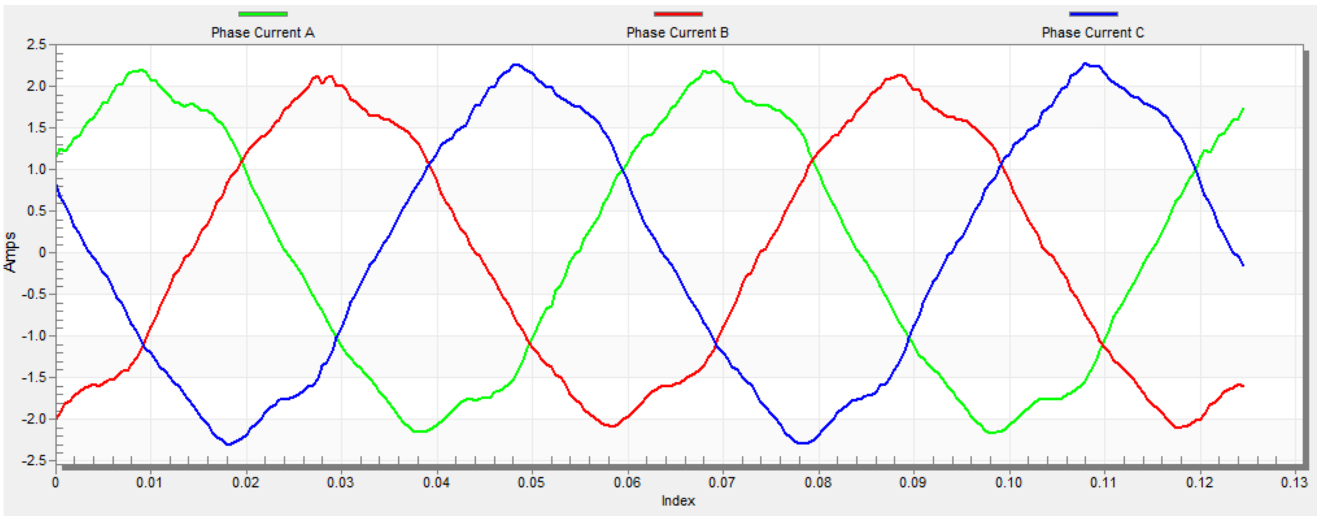


Figure 24. Phase currents

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 25) 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.

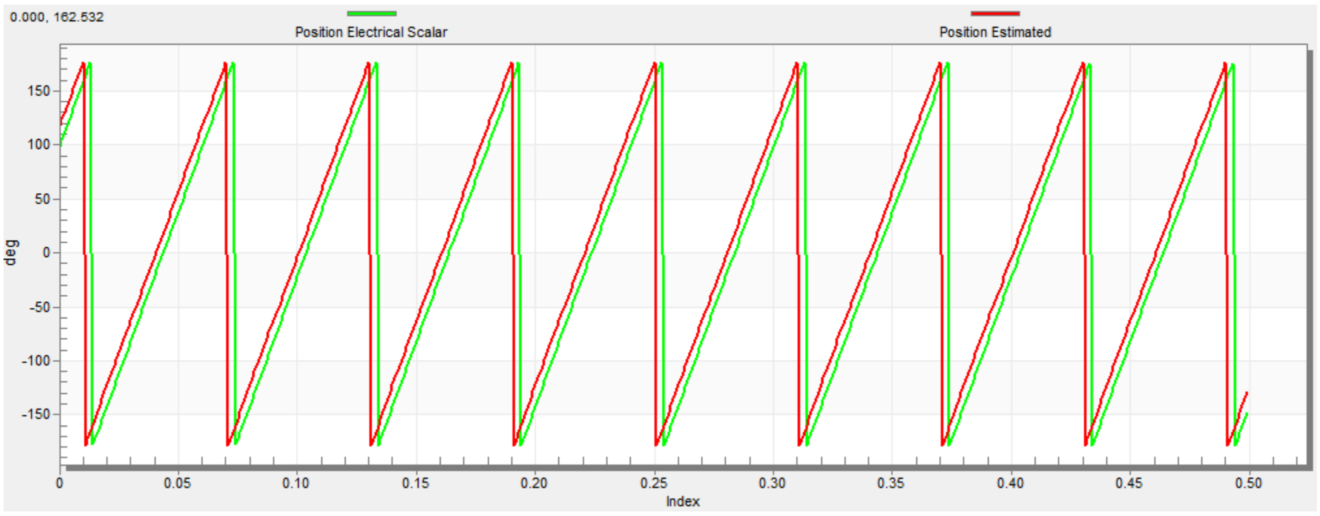


Figure 25. Generated and estimated positions

7. If an opposite speed direction is required, set a negative speed value into the "M1 Scalar Freq Required" variable.
8. The proper observer functionality and the measurement of analog quantities is expected at this step.
9. Enable the voltage FOC mode in the "M1 MCAT Control" variable while the main application switch "M1 Application Switch" is turned off.
10. Switch on the main application switch on and set a non-zero value in the “M1 MCAT Uq Required” variable. The FOC algorithm uses the estimated position to run the motor.

8.7.1 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 and also by connecting the motor phases.

To determine the direction of rotation, follow the steps below:

1. Navigate to the "Scalar & Voltage Control" tab in the project tree and select "SCALAR_CONTROL" in the "M1 MCAT Control" variable.
2. Turn on the application switch. The application state changes to "RUN".
3. Set the required frequency value in the "M1 Scalar Freq Required" variable; for example, 15 Hz. 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. The first way is to change "M1 Encoder Direction" variable - only 0 or 1 value is allowed. The second way is invert the encoder wires—phase A and phase B (or the other way round).

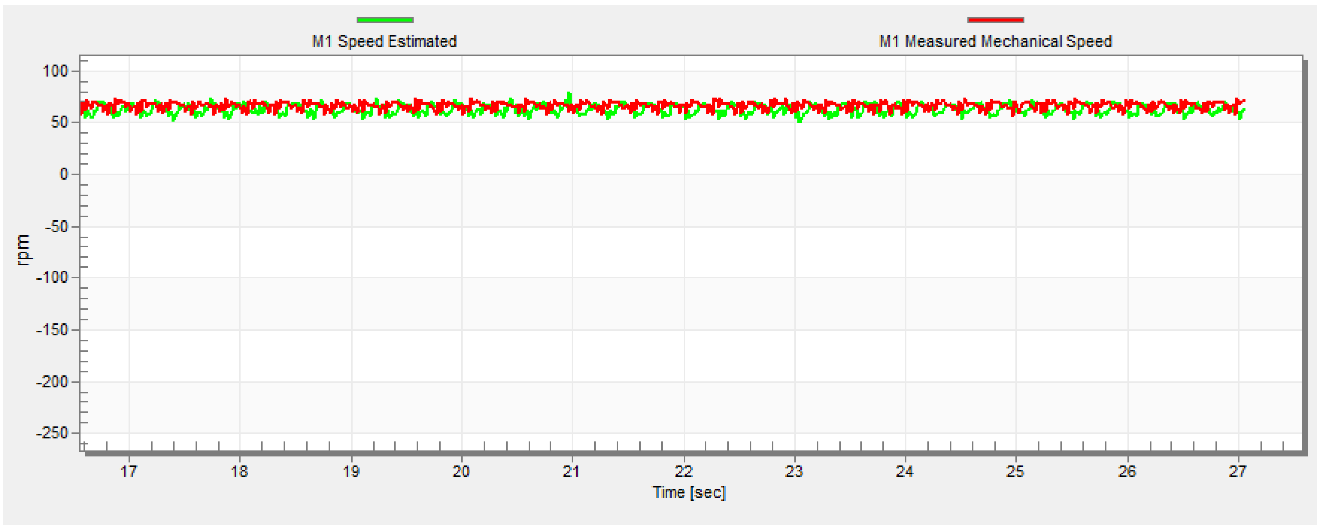


Figure 26. Encoder direction—right direction

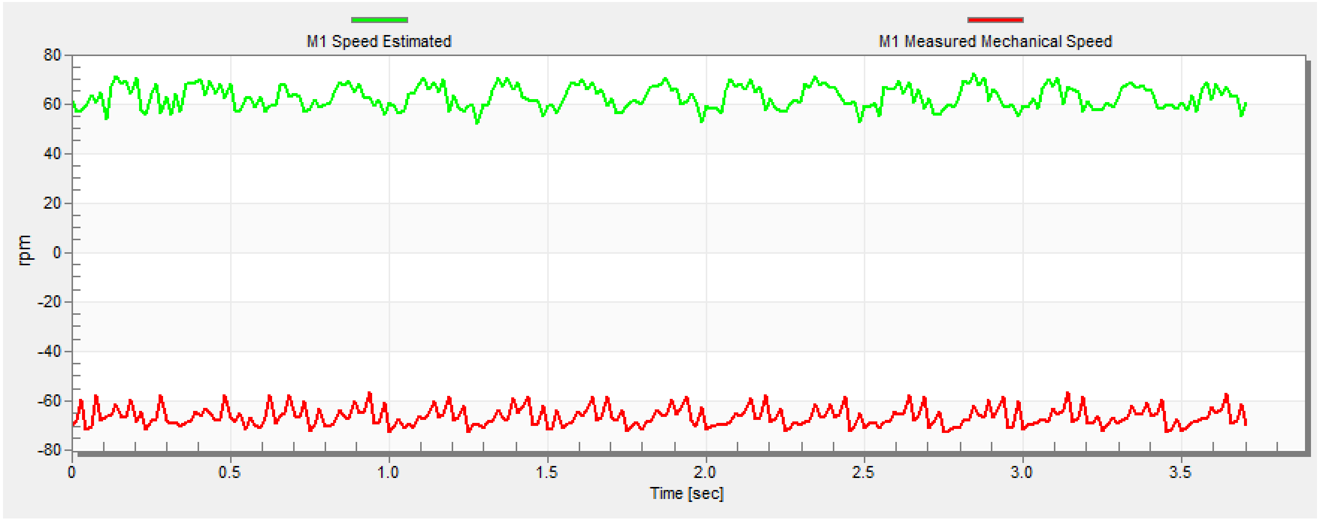


Figure 27. Encoder direction—wrong direction

8.7.2 Alignment tuning

For the alignment parameters, navigate to the "Parameters" MCAT tab. The alignment procedure sets the rotor to an accurate initial position and enables you to apply full startup torque to the motor. A correct initial position is needed mainly for high startup loads (compressors, washers, and so on). The alignment aims to have the rotor in a stable position, without any oscillations before the startup.

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

8.7.3 Current loop tuning

The parameters for the current D, Q, and PI controllers are fully calculated using the motor parameters and no action is required in this mode. If the calculated loop parameters do not correspond to the required response, the bandwidth and attenuation parameters can be tuned.

1. Select “Openloop Control” in the FreeMASTER project tree, set “M1 MCAT Control” to “OPENLOOP_CTRL” and switch “M1 Openloop Use I Control” on.
2. Turn the application on by switching “M1 Application Switch” on and then set “M1 Openloop Requred Id” for rotor alignment. (Rotor alignment always uses Id, even when you are tuning the Q axis regulator)
3. Mechanically lock the motor schaft and turn the application off.
4. Set the required loop bandwidth and attenuation in MCAT “Current loop” tab and then click the “Update target” button. The tuning loop bandwidth parameter defines how fast the loop response is while the tuning loop attenuation parameter defines the actual overshoot magnitude.
5. Select “Current Controller Id” recorder in project tree, turn the application on and set the required step amplitude in “M1 Openloop Requred Id”. Observe the step response in the recorder.
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 (100 Hz) and the settling time of the Id current is long (Figure 28).

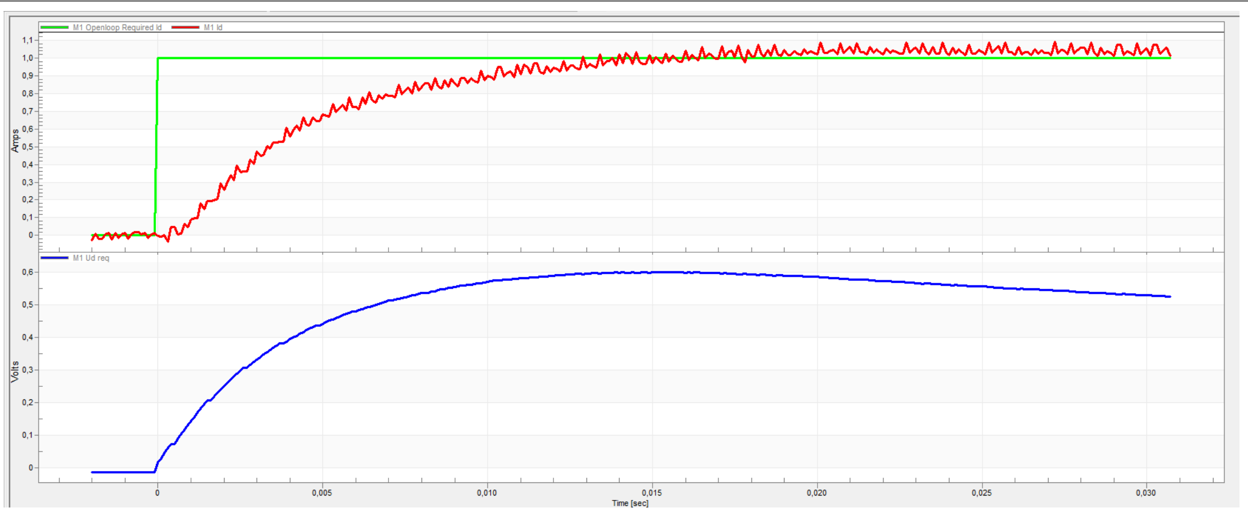


Figure 28. Slow step response of the Id current controller

   - The loop bandwidth (300 Hz) is optimal and the response time of the Id current is sufficient (Figure 29).

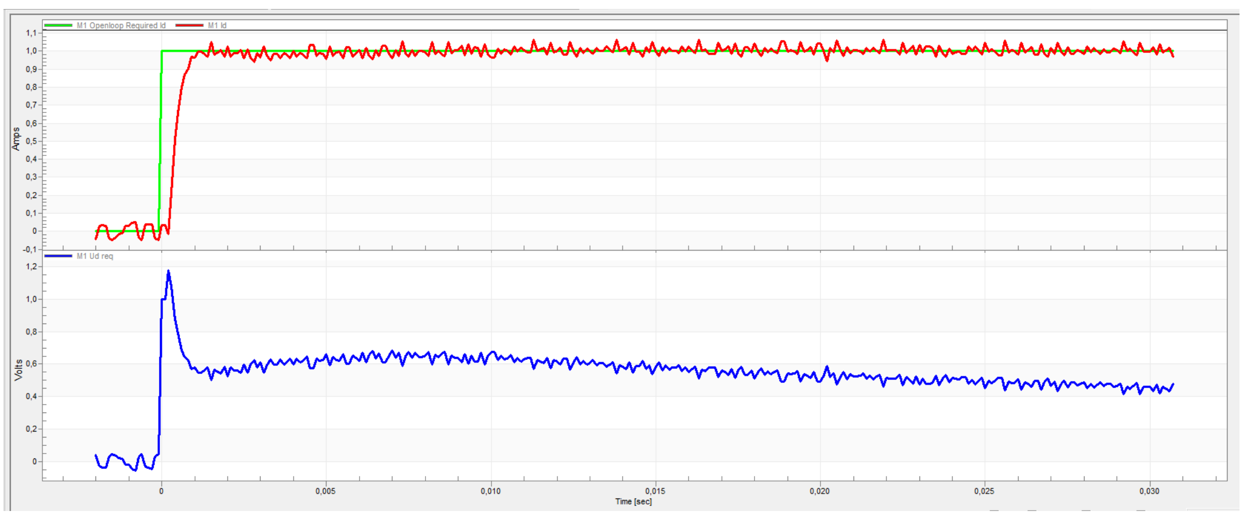


Figure 29. Optimal step response of the Id current controller

- The loop bandwidth is high (700 Hz) and the response time of the Id current is very fast, but with oscillations and overshoot (Figure 30).

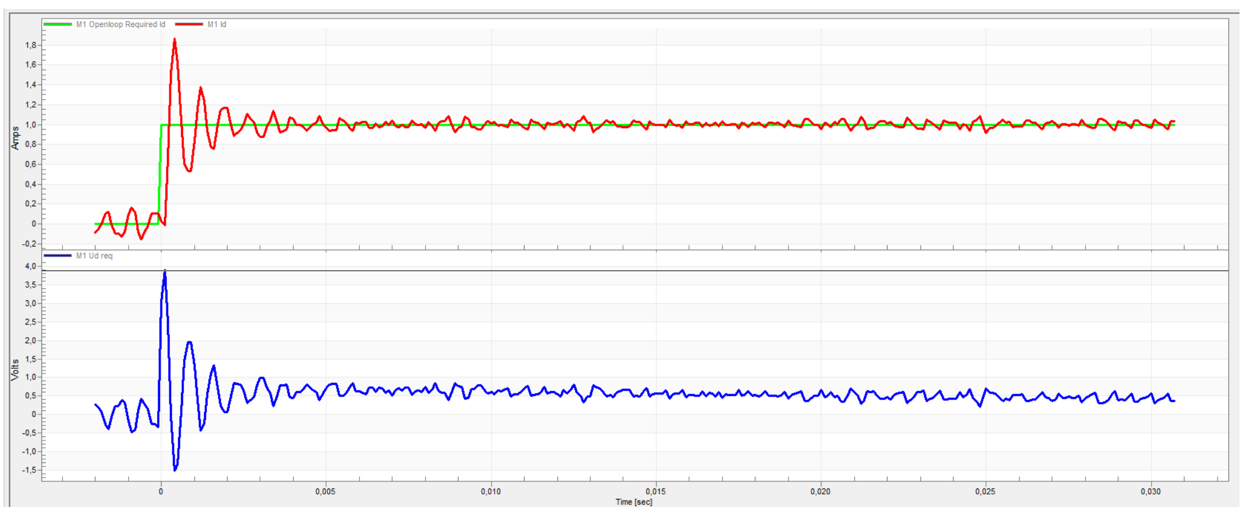


Figure 30. Fast step response of the Id current controller

8.7.4 Speed ramp tuning

To tune speed ramp parameters, follow the steps below:

1. The speed command is applied to the speed controller through a speed ramp. The ramp function contains two increments (up and down) which express the motor acceleration and deceleration per second. If the increments are very high, they can cause an overcurrent fault during acceleration and an overvoltage fault during deceleration. In the "Speed" scope, you can see whether the "Speed Actual Filtered" waveform shape equals the "Speed Ramp" profile.
2. The increments are common for the scalar and speed control. The increment fields are in the "Speed loop" tab and accessible in both tuning modes. Clicking the "Update target" button applies the changes to the MCU. An example speed profile is shown in Figure 31. The ramp increment down is set to 500 rpm/sec and the increment up is set to 3000 rpm/sec.
3. The startup ramp increment is in the "Sensorless" tab and its value is higher than the speed loop ramp.

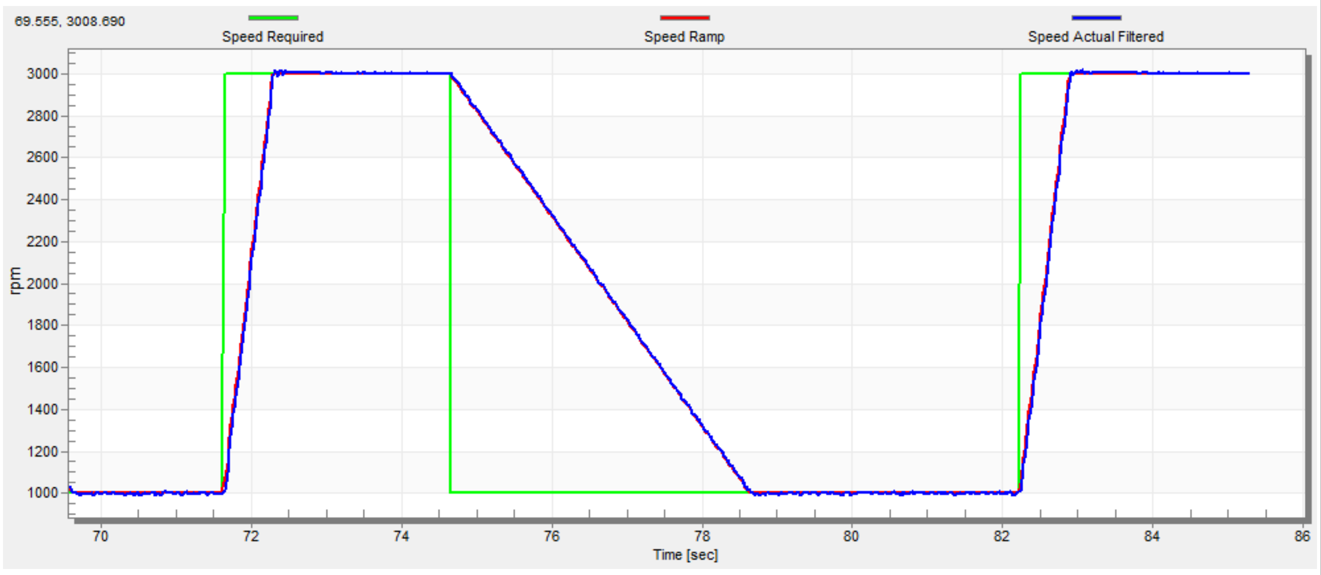


Figure 31. Speed profile

8.7.5 Open loop startup

To tune open loop startup parameters, follow the steps below:

1. The startup 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 startup tuning can be processed in all control modes besides the scalar control. Setting the optimal values results in a proper motor startup. An example startup state of low-dynamic drives (fans, pumps) is shown in Figure 32.
2. Select the "Startup" recorder from the FreeMASTER project tree.
3. Set the startup ramp increment typically to a higher value than the speed-loop ramp increment.
4. Set the startup current according to the required startup torque. For drives such as fans or pumps, the startup 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 one of an electrical revolution. The higher the value, the faster the merge. Values close to 1 % are set for the drives where a high startup torque and smooth transitions between the open loop and the closed loop are required.
7. To apply the changes to the MCU, click the "Update Target" button.
8. Select "SPEED_FOC" in the "M1 MCAT Control" variable.
9. Set the required speed higher than the merging speed.
10. Check the startup response in the recorder.
11. Tune the startup parameters until you achieve an optimal response.
12. If the rotor does not start running, increase the startup current.
13. If the merging process fails (the rotor is stuck or stopped), decrease the startup ramp increment, increase the merging speed, and set the merging coefficient to 5 %.

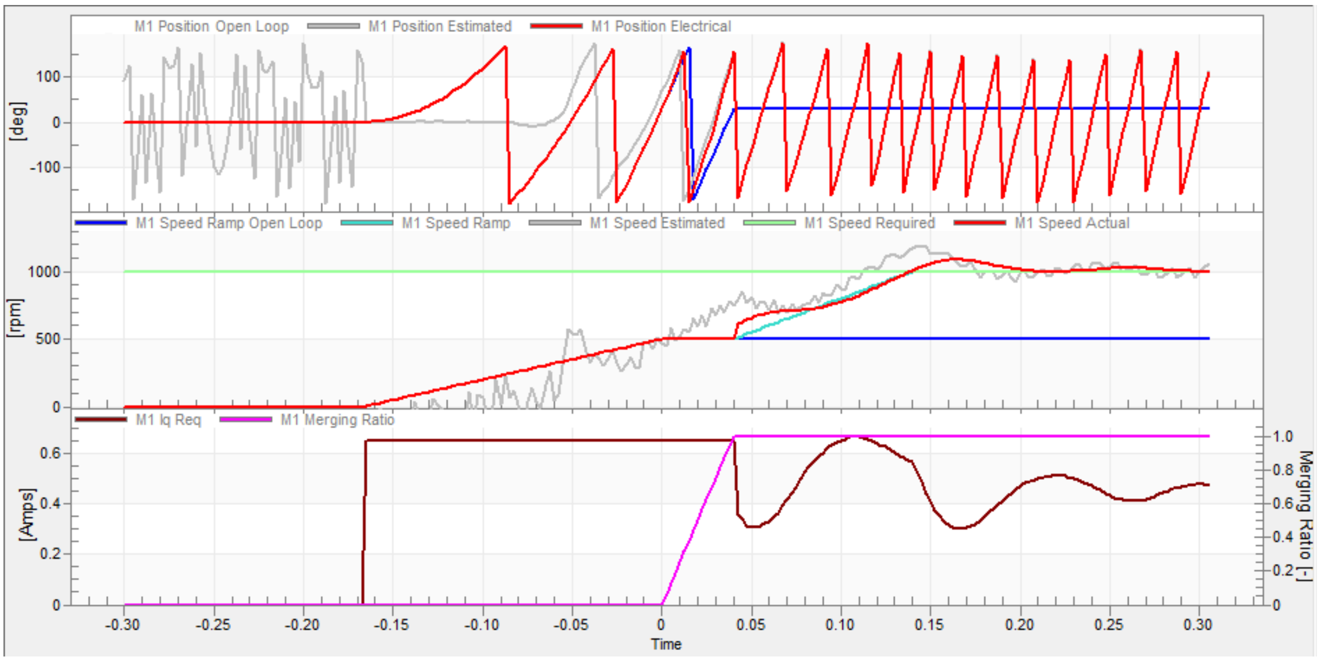


Figure 32. Motor startup

8.7.6 BEMF observer tuning

The bandwidth and attenuation parameters of the BEMF and tracking observer can be tuned. To tune the bandwidth and attenuation parameters, follow the steps below:

1. Navigate to the "Sensorless" MCAT tab.
2. Set the required bandwidth and attenuation of the BEMF observer. The bandwidth is typically set to a value close to the current loop bandwidth.
3. Set the required bandwidth and attenuation of the tracking observer. The bandwidth is typically set in the range of 10 – 20 Hz for most low-dynamic drives (fans, pumps).
4. To apply the changes to the MCU, click the "Update target" button.
5. Select the "Observer" recorder from the FreeMASTER project tree and check the observer response in the "Observer" recorder.

8.7.7 Speed PI controller tuning

The motor speed control loop is a first-order function with a mechanical time constant that depends on the motor inertia and friction. If the mechanical constant is available, the PI controller constants can be tuned using the loop bandwidth and attenuation. Otherwise, the manual tuning of the P and I portions of the speed controllers is available to obtain the required speed response (see Figure 33). There are dozens of approaches to tune the PI controller constants. To set and tune the speed PI controller for a PM synchronous motor, follow the steps below:

1. Select the "Speed Controller" option from the FreeMASTER project tree.
2. Select the "Speed loop" tab.
3. Check the "Manual Constant Tuning" option—that is, the "Bandwidth" and "Attenuation" fields are disabled and the "SL_Kp" and "SL_Ki" fields are enabled.
4. Tune the proportional gain:
   - Set the "SL_Ki" integral gain to 0.
   - Set the speed ramp to 1000 rpm/sec (or higher).

- Run the motor at a convenient speed (about 30 % of the nominal speed).
- Set a step in the required speed to 40 % of N_{nom}.
- Adjust the proportional gain "SL_Kp" until the system responds to the required value properly and without any oscillations or excessive overshoot:
  - If the "SL_Kp" field is set low, the system response is slow.
  - If the "SL_Kp" field is set high, the system response is tighter.
  - When the "SL_Ki" field is 0, the system most probably does not achieve the required speed.
  - To apply the changes to the MCU, click the "Update Target" button.

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.
   - To apply the changes to the MCU, click the "Update target" button.
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".

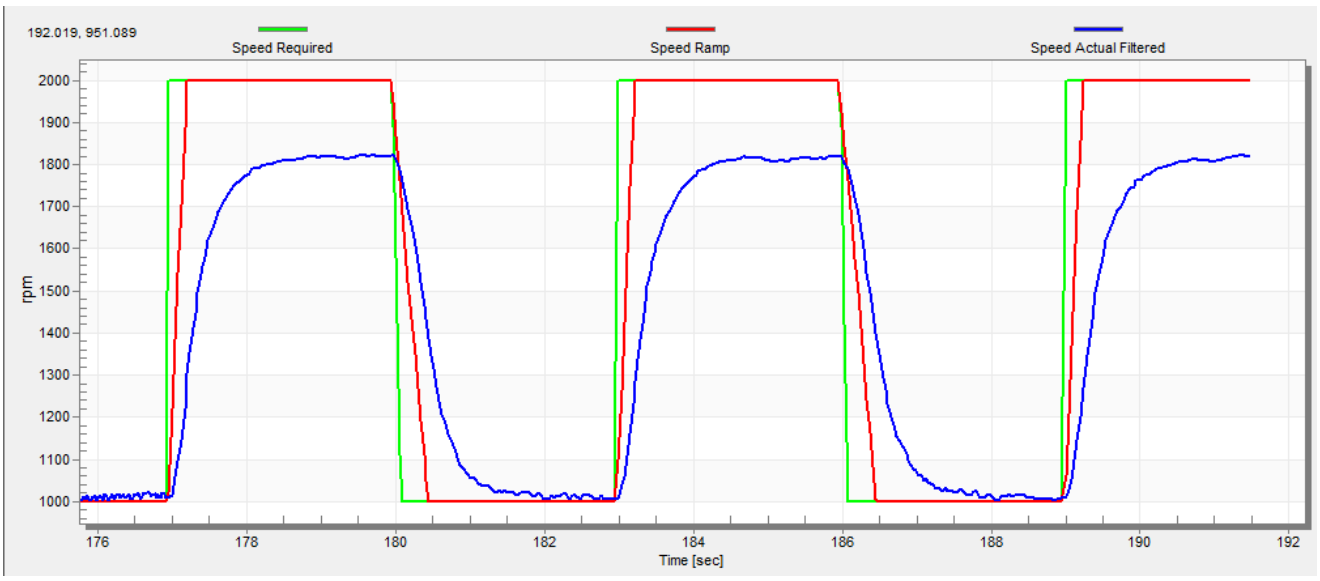


Figure 33. Speed controller response—SL_Ki value is low, Speed Ramp is not achieved

   - The "SL_Kp" value is low, the "Speed Actual Filtered" greatly overshoots, and the long settling time is unwanted.

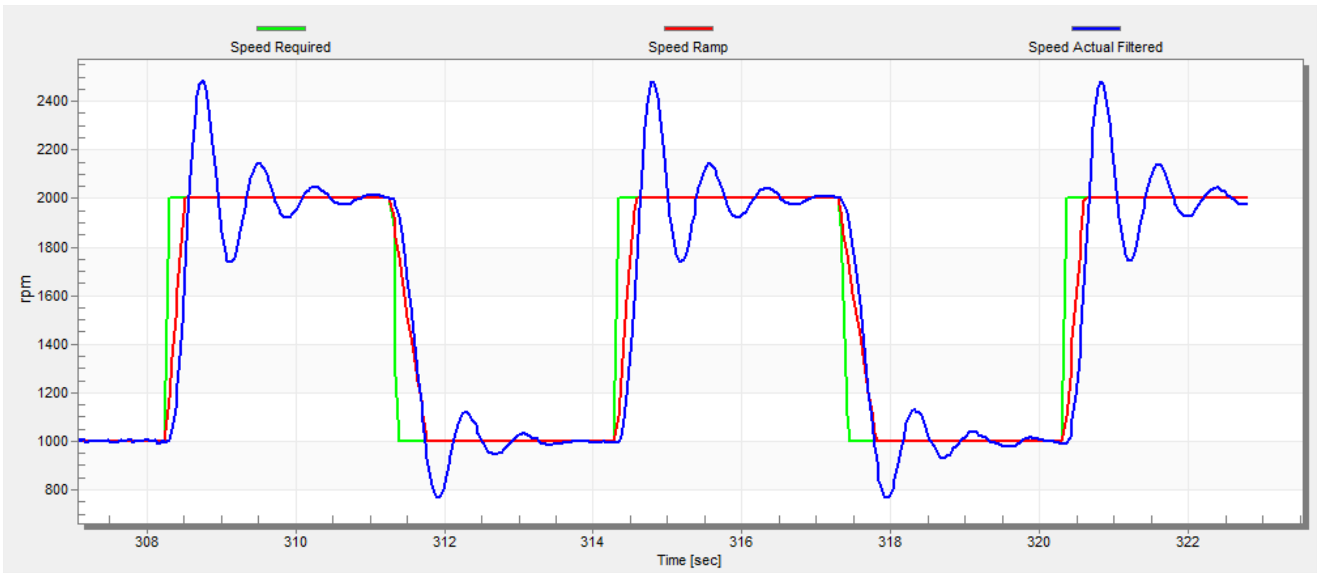


Figure 34. Speed controller response—SL_Kp value is low, Speed Actual Filtered greatly overshoots

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

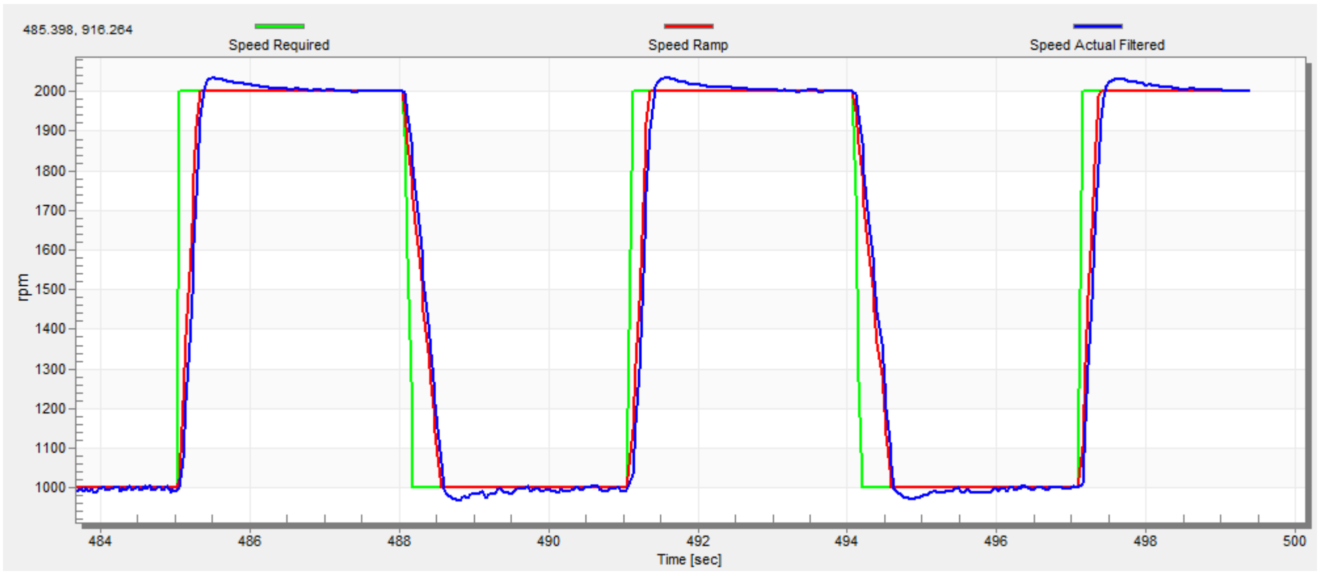


Figure 35. Speed controller response—speed loop response with a small overshoot

8.7.8 Position P controller tuning

The position control loop can be tuned Bandwidth F0 and Attenuation ξ in Servo control tab. A proportional controller can be used to unpretend the position-control systems. The key for the optimal position response is a proper value of the controller, which multiplies the error by the proportional gain (Kp) to get the controller output. An encoder sensor must be used for a working position control. The following steps provide an example of how to set the position P controller for a PM synchronous motor:

1. Select the "Position Controller" scope in "Position Control" tab in the FreeMASTER project tree.
2. Tune the proportional gain by Bandwidth F0 and Attenuation ξ:
   - Set preffered value of Bandwidth F0 and Attenuation ξ. The MCAT automatically compute position controller gain and feed forward constants.
   - Select the position control, and set the required position in "M1 Position Required" variable (for example; 10 revolutions).
   - Select the "Position Controller" scope and watch the actual position response.
3. Repeat the previous steps until you achieve the required position response.

The "PL_Kp" value is low and the actual position response on the required position is very slow.

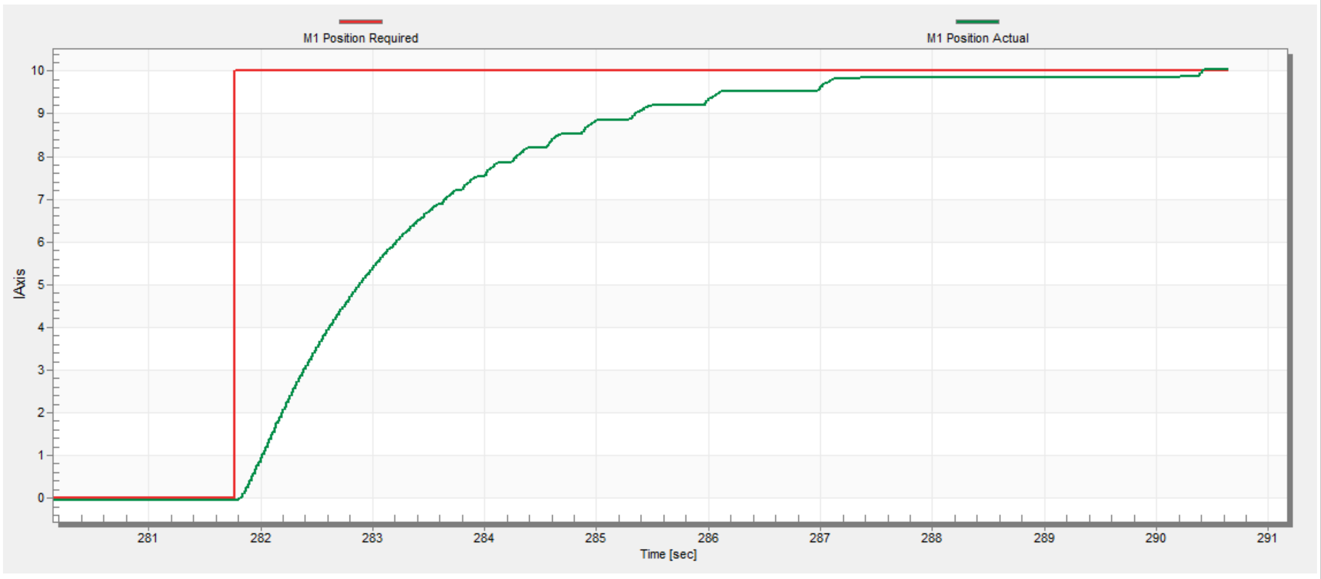


Figure 36. Position controller response—PL_Kp value is low, the actual position response is very slow

The "PL_Kp" value is too high and the actual position overshoots the required position.

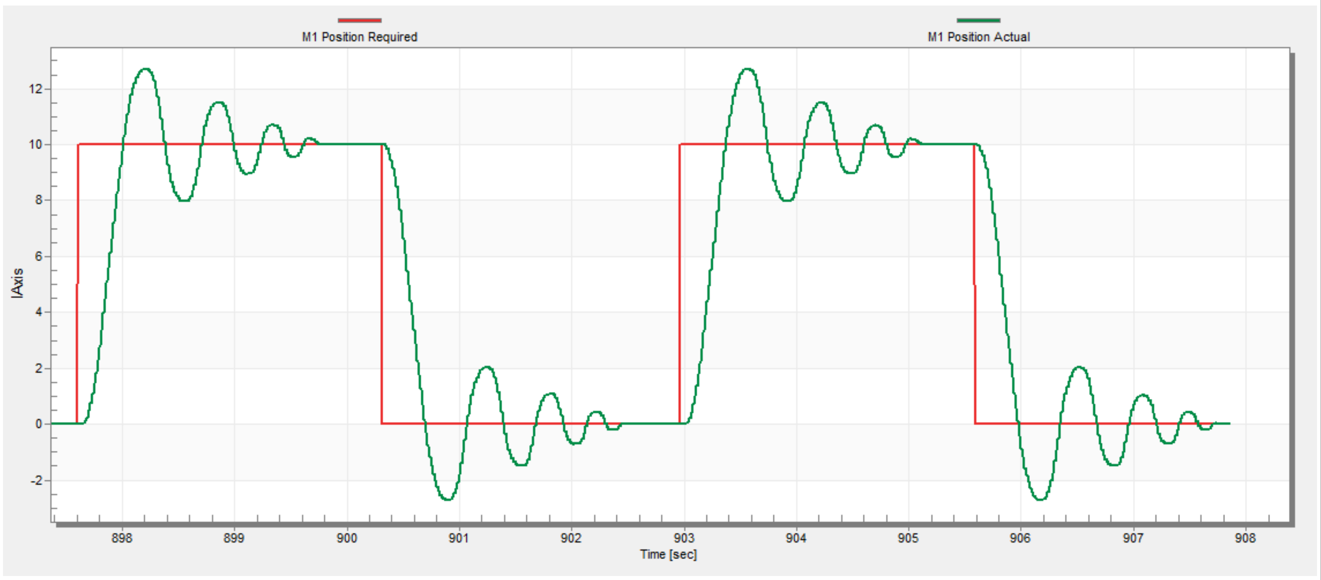


Figure 37. Position controller response—PL_Kp value is too high and the actual position overshoots

The "PL_Kp" value and the actual position response are optimal.

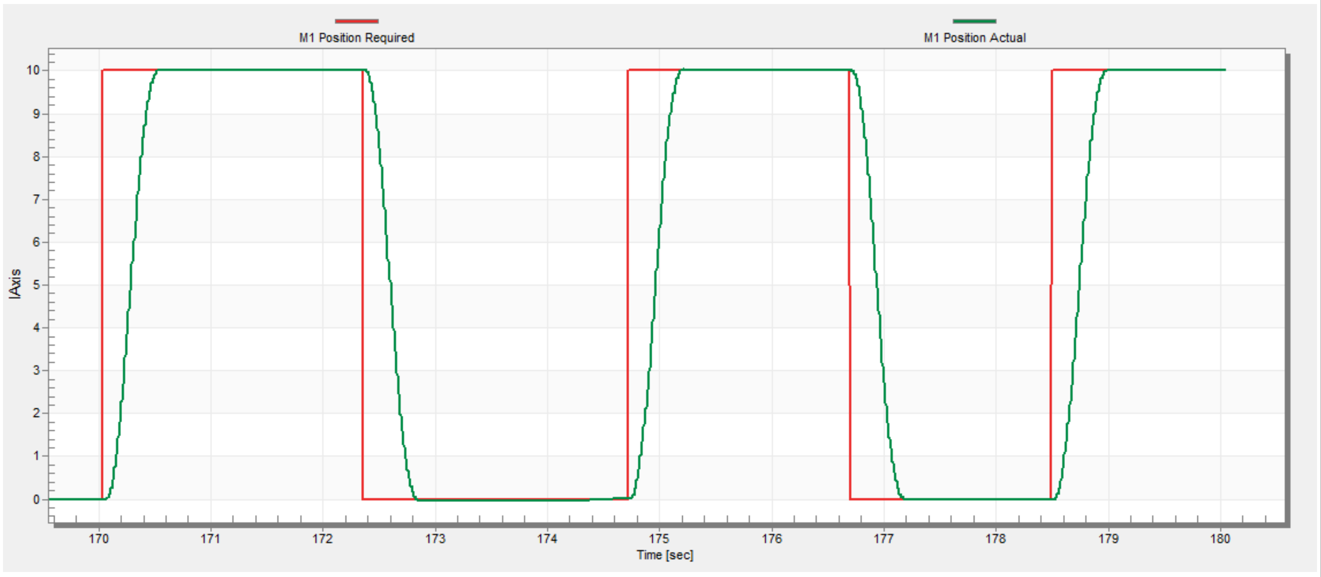


Figure 38. Position controller response—the actual position response is good

9 Conclusion

This application note describes the implementation of the sensor and sensorless field-oriented control of a 3-phase PMSM. The motor control software is implemented on NXP i.MX943-EVK board with the FRDM-LVPMSM-FA NXP Freedom development platform. The hardware-dependent part of the control software is described in Section 2. The motor-control application timing, and the peripheral initialization are described in Section 3. The motor user interface and remote control using FreeMASTER are described in Section 7.

10 Acronyms and abbreviations

Table 20 lists the acronyms and abbreviations used in this document.

Table 20. Acronyms and abbreviations

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

11 References

These references are available on www.nxp.com:

- *Sensorless PMSM Field-Oriented Control* (document DRM148)
- *Motor Control Application Tuning (MCAT) Tool for 3-Phase PMSM* (document AN4642)

12 Useful links

- MCUXpresso SDK for Motor Control www.nxp.com/sdkmotorcontrol
- Motor Control Application Tuning (MCAT) Tool
- MCUXpresso IDE - Importing MCUXpresso SDK
- MCUXpresso Config Tool
- MCUXpresso SDK Builder (SDK examples in several IDEs)
- Model-Based Design Toolbox (MBDT)

13 Revision history

This section summarizes the changes done to the document since the initial release.

Table 21. Revision history

Document ID	Release date	Description
UG10263 v.1.0	03 July 2025	Initial release

Legal information

Tables

Tab. 1. Available example type, supported motors and control methods 2
Tab. 2. TGT3-0065-60-48 motor parameters 3
Tab. 3. HMD06-011-048 motor parameters 4
Tab. 4. SW90 setting for BiSS encoder 5
Tab. 5. SW90 setting for EnDat2.2 encoder 6
Tab. 6. BiSS point-to-point connection 6
Tab. 7. EnDat2.2 connection 6
Tab. 8. i.MX943-EVK switch settings 6
Tab. 9. Maximum CPU load (fast loop) 12
Tab. 10. Memory usage 12
Tab. 11. Constants used in equations 24
Tab. 12. Parameters tab inputs 24
Tab. 13. Current loop tab input 27
Tab. 14. Speed loop tab input 27
Tab. 15. Servo control tab input 29
Tab. 16. Sensorless tab input 30
Tab. 17. MCAT motor parameters 39
Tab. 18. Fault limits 40
Tab. 19. Application scales 40
Tab. 20. Acronyms and abbreviations 52
Tab. 21. Revision history 55

Figures

Fig. 1. TGT3-0065-60-48 permanent magnet synchronous motor 3
Fig. 2. HMD06-011-048 permanent magnet synchronous motor 4
Fig. 3. FRDM-LVPMSM-FA block diagram 5
Fig. 4. FRDM-LVPMSM-FA board 5
Fig. 5. i.MX943-EVK board 7
Fig. 6. Timing on i.MX94X - SINC trigger and ISR 8
Fig. 7. Timing i.MX94X - BiSS trigger and Fast loop ISR 9
Fig. 8. Timing i.MX94X - dual servo application 9
Fig. 9. Green “GO” button placed in top left-hand corner 19
Fig. 10. FreeMASTER—communication is established successfully 19
Fig. 11. FreeMASTER communication setup window 20
Fig. 12. Default symbol file 21
Fig. 13. FreeMASTER + MCAT layout 23
Fig. 14. Scalar control mode 31
Fig. 15. Voltage - Open loop control 32
Fig. 16. Current - Open loop control 33
Fig. 17. Voltage FOC control mode 34
Fig. 18. Current/Torque control mode (sensored) 35
Fig. 19. Speed FOC control mode (sensored) 36
Fig. 20. Position control mode 36
Fig. 21. Faults in variable watch located in "Motor M1" subblock 37
Fig. 22. Undervoltage fault is indicated (pending) 38
Fig. 23. Undervoltage fault is captured 38
Fig. 24. Phase currents 41
Fig. 25. Generated and estimated positions 41
Fig. 26. Encoder direction—right direction 42
Fig. 27. Encoder direction—wrong direction 42
Fig. 28. Slow step response of the Id current controller 43
Fig. 29. Optimal step response of the Id current controller 44
Fig. 30. Fast step response of the Id current controller 44
Fig. 31. Speed profile 45
Fig. 32. Motor startup 46
Fig. 33. Speed controller response—SL_Ki value is low, Speed Ramp is not achieved 47
Fig. 34. Speed controller response—SL_Kp value is low, Speed Actual Filtered greatly overshoots 48
Fig. 35. Speed controller response—speed loop response with a small overshoot 48
Fig. 36. Position controller response—PL_Kp value is low, the actual position response is very slow 49
Fig. 37. Position controller response—PL_Kp value is too high and the actual position overshoots 49
Fig. 38. Position controller response—the actual position response is good 50

Contents

1 Introduction 2
2 Hardware setup 3
2.1 TGT motor 3
2.2 Heidrive motor 4
2.3 FRDM-LVPMSM-FA 4
2.4 i.MX943-EVK board 6
2.4.1 Hardware assembling 7
2.4.2 Additional information 7
3 Processors features and peripheral settings 8
3.1 i.MX 94x 8
3.1.1 i.MX94X - Hardware timing and synchronization 8
3.1.2 i.MX94X - Peripheral settings 9
3.1.2.1 PWM generation 10
3.1.2.2 SINC filter 10
3.1.2.3 BiSS interface 10
3.1.2.4 EnDat2.2 interface 10
3.1.2.5 XBAR - peripheral interconnection 11
3.1.2.6 Slow-loop interrupt generation 11
3.2 CPU load and memory usage 11
4 SDK package project file and IDE workspace structure 13
4.1 PMSM project structure 13
5 Github project structure 15
6 Motor-control peripheral initialization 16
7 User interface 18
8 Remote control using FreeMASTER 19
8.1 Establishing FreeMASTER communication 19
8.2 TSA replacement with ELF file 20
8.3 Motor Control Aplication Tuning interface (MCAT) 21
8.3.1 MCAT tabs description 23
8.3.1.1 Application concept 24
8.3.1.2 Parameters 24
8.3.1.3 Current loop 27
8.3.1.4 Speed loop 27
8.3.1.5 Servo control 29
8.3.1.6 Sensorless 30
8.4 Motor Control Modes - How to run motor 31
8.4.1 Scalar control 31
8.4.2 Open loop control mode 32
8.4.3 Voltage control 33
8.4.4 Current/Torque control 34
8.4.5 Speed FOC control 35
8.4.6 Position (servo) control 36
8.5 Faults explanation 37
8.5.1 Variable "M1 Fault Pending" 37
8.5.2 Variable "M1 Fault Captured" 38
8.5.3 Variable "M1 Fault Enable" 38
8.6 Initial motor parameters and harware configuration 39
8.7 Control parameters tuning 40
8.7.1 Encoder sensor setting 41
8.7.2 Alignment tuning 43
8.7.3 Current loop tuning 43
8.7.4 Speed ramp tuning 44
8.7.5 Open loop startup 45
8.7.6 BEMF observer tuning 46
8.7.7 Speed PI controller tuning 46
8.7.8 Position P controller tuning 48
9 Conclusion 51
10 Acronyms and abbreviations 52
11 References 53
12 Useful links 54
13 Revision history 55
Legal information 56